Philipps Universität Marburg

# SECURITY ANALYSIS OF SYSTEM BEHAVIOUR

## – FROM 'SECURITY BY DESIGN' TO 'SECURITY AT RUNTIME' –

Dissertation
zur Erlangung des Doktorgrades der
Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
(Hochschulkennziffer 1180)
vorgelegt von

Roland Rieke
geboren in Braunschweig

Marburg, 2014

Vom Fachbereich Mathematik und Informatik der
Philipps-Universität Marburg als Dissertation am
12.12.2014 angenommen.

Erstgutachter: Prof. Dr. Bernd Freisleben

Zweitgutachter: Prof. Dr. Bernhard Seeger

Tag der mündlichen Prüfung: 12.12.2014.

Dedicated to my family, my friends,
and my teachers.

## MANY THANKS TO ALL WHO HAVE CONTRIBUTED

First of all, I would like to thank my supervisors *Bernd Freisleben* and *Bernhard Seeger* for supporting my thesis actively and for their collaboration in paving the way to forthcoming research in this area.

This thesis would not have been possible without the cooperation and exchange of ideas with my co-authors, specifically, *Peter Ochsenschläger* and *Jürgen Repp* have been involved in this research over many years.

*Peter Ochsenschläger* and *Carsten Rudolph* have created an inspiring and open-minded working atmosphere within the Fraunhofer SIT department currently named 'Trust and Compliance' and have supported my research interests in challenging times.

Furthermore, I like to thank the colleagues from the national and international research projects that have provided the context for the research in this thesis. In particular, my friends *Hervé Debar*, *Igor Kotenko*, *Andrew Hutchison*, *Elsa Prieto*, *Gunnar Björkman*, *Maria Zhdanova*, *Zaharina Stoynova*, *Jörn Eichler*, *Julian Schütte*, *Michael Jäger*, *Nicolai Kuntze*, *Andreas Fuchs*, and *Sigrid Gürgens* supported my work and encouraged me within the MASSIF project that eventually provided the proof of the applicability of my work.

# ABSTRACT

The Internet today provides the environment for novel applications and processes which may evolve way beyond pre-planned scope and purpose. Security analysis is growing in complexity with the increase in functionality, connectivity, and dynamics of current electronic business processes. Technical processes within critical infrastructures also have to cope with these developments. To tackle the complexity of the security analysis, the application of models is becoming standard practice. However, model-based support for security analysis is not only needed in pre-operational phases but also during process execution, in order to provide situational security awareness at runtime.

This cumulative thesis provides three major contributions to modelling methodology.

Firstly, this thesis provides an approach for model-based analysis and verification of security and safety properties in order to support *fault prevention* and *fault removal* in system design or redesign. Furthermore, some construction principles for the design of well-behaved scalable systems are given.

The second topic is the analysis of the exposition of vulnerabilities in the software components of networked systems to exploitation by internal or external threats. This kind of *fault forecasting* allows the security assessment of alternative system configurations and security policies. Validation and deployment of security policies that minimise the attack surface can now improve *fault tolerance* and mitigate the impact of successful attacks.

Thirdly, the approach is extended to runtime applicability. An observing system monitors an event stream from the observed system with the aim to *detect faults* – deviations from the specified behaviour or security compliance violations – at runtime. Furthermore, knowledge about the expected behaviour given by an operational model is used to *predict faults* in the near future. Building on this, a holistic security management strategy is proposed. The architecture of the observing system is described and the applicability of model-based security analysis at runtime is demonstrated utilising processes from several industrial scenarios.

The results of this cumulative thesis are provided by 19 selected peer-reviewed papers.

# ZUSAMMENFASSUNG

Das Internet bietet heute das Umfeld für neue Anwendungen und Prozesse, die sich weit über den im Voraus geplanten Zweck entwickeln können. Die Komplexität der Sicherheitsanalyse wächst mit der Erhöhung der Funktionalität, Konnektivität und Dynamik der betrachteten Systeme. Die Verwendung von Modellen ist mittlerweile etabliert, um die Komplexität der Sicherheitsanalyse zu bewältigen. Modellbasierte Methoden für den Entwurf sicherer System werden jedoch nicht nur während der Entwicklung von Systemen benötigt. Wie diese Arbeit zeigt, können Modelle auch während der Laufzeit helfen, sicherheitskritische Situationen zu erkennen und zu bewerten.

Diese kumulative Dissertation umfasst drei wichtige Beiträge zur Modellierungsmethodik.

Zum einen wird ein modellbasiertes Konzept vorgestellt, um Sicherheitseigenschaften von Systemen zu verifizieren und so Fehler in der Entwurfsphase zu vermeiden bzw. beim Redesign zu entfernen. Darüber hinaus werden Konstruktionsprinzipien vorgestellt, die beim Enwurf skalierbarer Systeme sicherstellen, dass die Sicherheitseigenschaften eines Systems bei der Erweiterung um gleichartige Komponenten erhalten bleiben.

Das zweite Thema ist die Analyse der Sicherheitslücken in den Software-Komponenten vernetzter Systeme bezüglich der Exposition für interne oder externe Bedrohungen. Diese Art der Fehlervorhersage ermöglicht die Bewertung von alternativen Systemkonfigurationen und Sicherheitsrichtlinien. Der Einsatz von validierten Sicherheitsrichtlinien soll die Angriffsfläche minimieren, die Fehlertoleranz verbessern und Auswirkungen erfolgreicher Angriffe abschwächen.

Das dritte Thema ist die Erweiterung der Methoden, um die Anwendbarkeit zur Laufzeit. Ein Beobachtungssystem überwacht einen Ereignisstrom aus dem beobachteten System mit dem Ziel, Fehler – in Form von Abweichungen vom festgelegten Verhalten oder Verstöße gegen Sicherheitsanforderungen – zur Laufzeit zu erkennen. Darüber hinaus wird das Wissen über das erwartete Verhalten, das durch ein ausführbares Modell spezifiziert wird, verwendet, um Fehler in der nahen Zukunft vorherzusagen. Darauf aufbauend wird eine Strategie für ein ganzheitliches Sicherheitsmanagement vorgeschlagen. Die Architektur des Beobachtungssystems wird beschrieben und die Anwendbarkeit der modellbasierten Sicherheitsanalyse zur Laufzeit wird anhand von Prozessen aus mehreren Industrieszenarien demonstriert.

Die Ergebnisse dieser kumulativen Dissertation wurden in 19 ausgewählten Forschungsarbeiten (peer-reviewed) veröffentlicht.

# ACKNOWLEDGMENTS

## COPYRIGHT REMARKS

## ACKNOWLEDGEMENTS

The author gratefully acknowledges the funding for parts of the work presented in the papers in Part III that have been developed in the context of the projects MASSIF (ID 257475) and EVITA (ID 224275) being co-funded by the European Commission in FP7, the projects ACCEPT (ID 01BY1206D), ADiWa (ID 01IA08006F), VOGUE (ID 01IS09032A), SicAri (ID 01AK062B), and SKe (ID 01AK953A and 01AK953B) being funded by the German Federal Ministry of Education and Research, and the project ProOnline-VSDD being funded by the German Federal Ministry of Health.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACRONYMS

| | |
|---|---|
| ACCEPT | Anomaliemanagement in Computersystemen durch Complex Event Processing Technologie |
| ADAS | Automated Data Acquisition System |
| ADiWa | Alliance Digital Product Flow |
| AMSEC | Attack Modelling and Security Evaluation Component |
| APA | Asynchronous Product Automaton |
| ATM | Air Traffic Management |
| BAM | Business Activity Monitoring |
| BPEL | Business Process Execution Language |
| BPM | Business Process Management |
| CAPEC | Common Attack Pattern Enumeration and Classification |
| CASENET | Computer-Aided solutions to SEcure electroNic commercE Transactions |
| CAST | Competence Center for Applied Security Technology |
| CCE | Common Configuration Enumeration |
| CPE | Common Platform Enumeration |
| CEP | Complex Event Processing |
| CIPC | Critical Infrastructure Process Control |
| COPS | Common Open Policy Service |
| CS | Cooperating Systems |
| CU | Communication Unit |
| CVE | Common Vulnerabilities and Exposures |
| CVSS | Common Vulnerability Scoring System |
| CWE | Common Weakness Enumeration |
| CWSS | Common Weakness Scoring System |
| DoS | Denial of Service |
| eGK | electronic health card |
| eHealth | electronic health |
| EKB | Engineering Knowledge Base |
| EPC | Event-driven Process Chain |
| ESP | Electronic Stability Protection |
| EVITA | E-safety Vehicle Intrusion proTected Applications |
| GPS | Global Positioning System |

| | |
|---|---|
| HMI | Human Machine Interface |
| ICT | Information and Communications Technology |
| IDMEF | Intrusion Detection Message Exchange Format |
| IDS | Intrusion Detection System |
| IFIP | International Federation for Information Processing |
| SPIIRAS | Institution of the Russian Academy of Sciences St.Petersburg Institute for Informatics and Automation of RAS |
| IoT | Internet of Things |
| ISMM | Information Security Measurement Model |
| IT | Information Technology |
| LTL | Linear Temporal Logic |
| LTS | Labeled Transition System |
| MASSIF | MAnagement of Security information and events in Service InFrastructures |
| MESI | Managed Enterprise Service Infrastructures |
| MMT | Mobile Money Transfer |
| MMTS | Mobile Money Transfer Service |
| NIST | National Institute of Standard and Technologies |
| NoW | Network-on-Wheels |
| NVD | National Vulnerability Database |
| OOGG | Olympic Games IT infrastructure |
| OrBAC | Organization Based Access Control |
| OSVDB | Open Sourced Vulnerability Database |
| PDP | Policy Decision Point |
| PDSA | Plan-Do-Study-Act |
| PEP | Policy Enforcement Point |
| PLTL | Propositional Linear Temporal Logic |
| PNML | Petri Net Markup Language |
| PSA | Predictive Security Analyser |
| PSA@R | Predictive Security Analysis at Runtime |
| QoS | Quality of Service |
| RBAC | Role-Based Access Control |
| RFID | Radio Frequency IDentification |
| RG | Reachability Graph |
| RSVP | Resource Reservation Protocol |
| RSU | RoadSide Unit |

| | |
|---|---|
| RQ | Research Question |
| SD | Security Directive |
| SCADA | Supervisory Control And Data Acquisition |
| SeMF | Security Modelling Framework |
| SHVT | Simple Homomorphism Verification Tool |
| SicAri | A security architecture and its tools for ubiquitous internet usage |
| SIEM | Security Information and Event Management |
| SIMM | Security Information Meta Model |
| SKe | Durchgängige Sicherheitskonzeption mit dynamischen Kontrollmechanismen für e-Service-Prozesse |
| SoS | Systems of Systems |
| SPARQL | SPARQL Protocol and RDF Query Language |
| SQUARE | Security Quality Engineering Methodology |
| SREP | Security Requirements Engineering Process |
| SSC | Security Strategy Component |
| SSM | Security Strategy Model |
| SSMM | Security Strategy Meta Model |
| SSPC | Security Strategy Processing Component |
| SWRL | Semantic Web Rule Language |
| TC | Trusted Computing |
| TL | Temporal Logic |
| TS | Transition System |
| VIKING | Vital Infrastructure, Networks, Information and Control Systems Management |
| WAVE | Wireless Access in Vehicular Environments |
| XACML | Xtensible Access Control Markup Language |

Part I

# PRELIMINARIES

*The first question in any scientific research is its subject matter: What are we studying ? The most general answer is a certain kind of system.*

— Sunny Y. Auyang, Foundations of Complex-system Theories [Auyang, 1998]

# 1 THESIS OVERVIEW

> *The acts of the mind, wherein it exerts its power over its simple ideas, are chiefly these three: (1) Combining several simple ideas into one compound one; and thus all complex ideas are made. (2) The second is bringing two ideas, whether simple or complex, together, and setting them by one another, so as to take a view of them at once, without uniting them into one; by which way it gets all its ideas of relations. (3) The third is separating them from all other ideas that accompany them in their real existence: this is called abstraction: and thus all its general ideas are made.*
>
> — John Locke, *An Essay Concerning Human Understanding (1690)*

AIM OF THIS CHAPTER. This chapter provides an overview of the research work presented in this thesis. It gives the background and motivation for the work and describes in detail the research objectives. Furthermore, it provides an overview of the approach taken and of the main results obtained. Finally, it introduces the structure of the thesis.

## 1.1 MOTIVATION AND KEY RESEARCH ISSUES

The transition from systems composed of many isolated, small-scale elements to large-scale, distributed and massively interconnected Systems of Systems (SoS) is a key challenge of modern Information and Communications Technology (ICT). Despite this increased complexity, these new SoS must still be dependable, which means they need to be secure, robust and efficient [Bullock & Cliff, 2004]. The role of *information security* in this context can be defined as follows.

> "The protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability. "
>
> — National Institute of Standard and Technologies (NIST), *2013 [Kissel, 2013]*

Similar definitions are given in Avizienis et al. [2004], Iso Iec [2005] and in Shirey [2007].

The overall aim of this thesis is to provide a modelling framework that is suitable for security and - to some extent - dependability analysis throughout the life cycle of a system: for design, configuration, and monitoring during operation as well as in the context of system adaptations to changing requirements and application context. In this thesis, those security requirements are of particular interest which are strongly related to overall safety goals.

Following the taxonomy of Laprie [1995], the *means to attain security and dependability* can be grouped into four major categories; namely, *fault prevention*, *fault removal*, *fault tolerance*, and *fault forecasting* [Avizienis et al., 2004]. This is illustrated in Figure 1 that is taken from Avizienis, Laprie, Randell & Landwehr [2004].

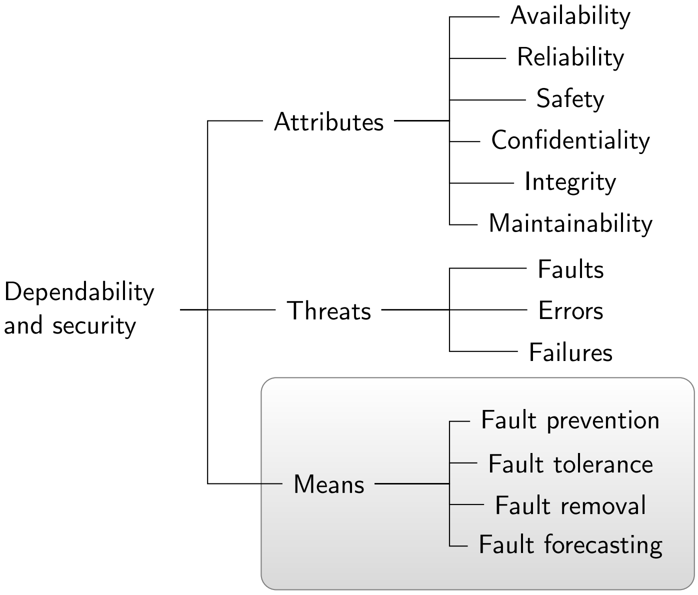


Figure 1: The dependability and security tree by Avizienis, Laprie, Randell, and Landwehr.

In terms of the categories given in Figure 1, the first topic of this thesis is model-based analysis and verification of security and safety properties in order to support *fault prevention* and *fault removal* in system design or redesign.

As the second topic, this thesis considers specific aspects of *fault forecasting* and *fault tolerance* by analysis of networked system configurations with respect to external exploitability of given vulnerabilities.

The third topic is security analysis by observing systems' behaviour at runtime. A given system may fail in the sense that some other system makes a judgement that the activity or inactivity of the given system constitutes failure [Randell, 2003]. Randell introduced the term *judgemental system* for the second system. In this thesis the term *observed system* will be used for the first system and *observing system* or *judgemental system* for the second system. The term *fault detection* will be used for the assessment of the behaviour of the observed system by the judgemental system. The term *fault prediction* will be used to express that the judgemental system forecasts a possible failure of the observed system in the near future.

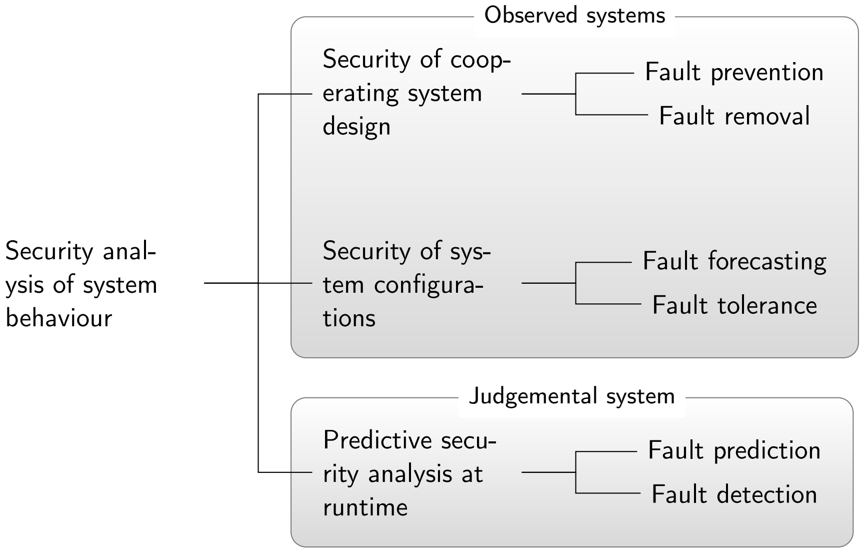

Observed systems
Security of cooperating system design
Fault prevention
Fault removal
Security analysis of system behaviour
Security of system configurations
Fault forecasting
Fault tolerance
Judgemental system
Predictive security analysis at runtime
Fault prediction
Fault detection

Figure 2: Topics of this thesis

In summary, the thesis addresses the following three topics (cf. Figure 2).

SECURITY OF COOPERATING SYSTEM DESIGN
addresses *fault prevention* and *fault removal* by verification of security properties with regard to the design or redesign of a system (cf. Section 1.1.1).

SECURITY OF SYSTEM CONFIGURATIONS
addresses specific aspects of *fault forecasting* and *fault tolerance* by analysis of system configurations (cf. Section 1.1.2).

PREDICTIVE SECURITY ANALYSIS AT RUNTIME
addresses *fault detection* and *fault prediction* by analysis of process specifications and compliance checks at runtime (cf. Section 1.1.3).

Thus, this thesis covers security analysis in three life cycle phases of a system: *design time*, *configuration time*, and *runtime*. The three topics are overlapping to some extend because the configuration of a SoS can be seen as part of the system specification which can be analysed at design time. However, in many cases the combination of systems which form the SoS at runtime is not known at design time. Furthermore, the configuration of SoS can be updated at runtime to match changing fault knowledge and attack situations.

For each of these three topics, research motivations and objectives will be presented in the following subsections. For each objective concrete Research Questions (RQs) will be given which are addressed by this thesis.

### 1.1.1 *Objectives with respect to security of cooperating system design*

Architecting novel SoS requires early consideration of dependability requirements in the system design process. Security engineering is one important aspect of dependability [Avizienis et al., 2004]. This is particularly the case when information security is the business-enabling technology, for example, for electronic health (eHealth) systems. The security engineering process addresses issues such as how to identify and mitigate risks resulting from connectivity and how to integrate security measures into a SoS architecture [Bodeau, 1994]. This chapter addresses specific tasks in the security engineering process.

A currently important example of such novel SoS are *vehicular ad hoc networks*. They will provide a number of complex new features - such as situational awareness - that enable vehicles to act autonomously and intelligently. Because of the fundamental importance of the safety of the users of these SoS - which may evolve to the largest ad hoc networks ever deployed - it is evident that this technology presents major challenges in the secure design of the involved systems and their communication protocols [Gerlach, 2005]. Similarly, future Air Traffic Management (ATM) systems will become distributed and highly interconnected SoS across organisational boundaries. ATM systems need to *collaborate for a common purpose* - such as the smooth running of an airport - and this includes continual update and improvement to security [Hawley et al., 2013].

In this thesis, SoS that collaborate for a common purpose are called Cooperating Systems (CS). CS are specific distributed SoS which are characterised by freedom of decision and loose coupling of their components [Ochsenschläger et al., 1998]. This causes a high degree of nondeterminism which has to be handled by the analysis methods. Typical examples of CS besides those mentioned above are telephone systems, smartcard systems, electronic money, and contract systems.

There is an important dependability requirement for systems like these: They must not only be secure, they must be *demonstrably* so. *Formal security models* are thus needed to be able to convince others of the security of a system [Landwehr, 1981]. Because of the success of the *Internet* and *embedded systems* in vehicles, airplanes and other safety critical systems, it will become even more important to develop methods that increase the confidence in the correctness of such systems [Clarke et al., 1999].

**Example 1.** *As an example for this line of argument, the formal analysis of the security of the communication infrastructure for a specific application* *of the German electronic health card (eGK) [Rieke, 2009b] underpins the following press release.*

> *"...Die begleitende Sicherheitsanalyse des Fraunhofer-Instituts für Sichere Informationstechnologie (SIT) hat gezeigt, dass dem Datenschutz und der Datensicherheit auf hohem technischem Niveau Rechnung getragen wird. ..."*
>
> — *gematik, 30.07.2009 - eGK besteht Online-Test [gematik, 2009]*

These considerations led to the following four objectives for this thesis with respect to *security of cooperating system design*.

**Objective 1** (Provide a framework for model-based security analysis)**.** With respect to *security at design time*, the first objective of this thesis is to provide means to prove that - in the context of CS - the components work together in a desired manner. This is expressed by the following research question.

**RQ 1.** *How can it be proven that components of cooperating systems securely work together?*

In this thesis *operational models* are used to model CS since they are executable and thus allow to analyse the behaviour of CS with respect to security and dependability properties. An answer to RQ1 is provided by the notion of *approximate satisfaction of properties* [Nitsche & Ochsenschläger, 1996] for which an implementation has been provided by the author of this thesis.

**Objective 2** (Enable scalable verification of properties)**.**
Traditional model checking techniques allow a verification of the required behaviour only for systems with very few components. A security analysis of the German eGK telematics infrastructure [Stroetmann & Lilischkis, 2007; gematik, 2007b] which was carried out by the author of this thesis [Rieke, 2009a,b] showed the limits of explicit finite state model checking methods (state space explosion). This leads to the following question.

**RQ 2.** *How can finite state verification techniques be extended to prove properties independently of concrete parameters?*

Usually only a few characteristic actions of the system are of interest with respect to verification of security critical behaviour. So it is evident to define *abstractions* with respect to the actions of interest. For example, in context of business processes an operational model can represent business operations at a granularity that is sufficient for validating progress toward goals and to analyse the dependencies between goals [Bhattacharya et al., 2007; Grimm & Ochsenschläger, 2001]. This thesis follows an abstraction-based approach, where the key problem is the choice of an appropriate abstraction that, (a) is property preserving, (b) results in identical abstract system behaviour for any given parameter configuration, and, (c) is sufficiently precise to express the required properties at the chosen abstraction level.

**Objective 3** (Elicit security requirements systematically)**.**
A typical application area for *mobile* CS are vehicular communication systems in which vehicles and roadside units communicate in ad hoc manner to exchange information such as safety warnings and traffic information. These mobile CS typically base decisions on information from their own components as well as on input from other systems. Safety critical decisions based on cooperative reasoning, such as automatic emergency braking of a vehicle, raise severe concerns to security issues. Thus, security requirements need to be explicit, precise, adequate, non-conflicting with other requirements and complete [van Lamsweerde, 2004]. Therefore, the following question has to be answered.

**RQ 3.** *How can security requirements for cooperating systems be elicited systematically?*

This thesis provides a partial solution which solves this question for *authenticity requirements*.

**Objective 4** (Identify principles for scalability)**.**
Systems that need to be highly scalable comprise grid computing architectures and cloud computing platforms [Bullock & Cliff, 2004; Weinman, 2011]. Usually such systems consist of few different types of components and for each such type a varying set of individual components exists. Component types can be defined in such a granularity that individual components of the same type behave in the same manner, which is characteristic for the type. For example, a client-server system that is scalable consists of the component types *client* and *server* and several sets of individual clients as well as several sets of individual servers. This motivates the objective to identify *design principles for verifiability* of security properties of *scalable systems*.

**RQ 4.** *Which design principles facilitate verifiability of security properties of scalable systems?*

This thesis focusses on properties that rely on specific component types and a specific number of individual components for these component types but not on the specific individuality of the individual components. *Well-behaved scalable systems* can be characterised by those systems which fulfil such a kind of property if already one prototype system (depending on the property) fulfils that property.

Security analysis at design time is the main subject of Chapter 2 of this thesis.

### 1.1.2 *Objectives with respect to security of system configurations*

A *security policy* - a set of security-motivated constraints - is a (partial) system specification, lack of adherence to which will be regarded as a security failure [Avizienis et al., 2004]. ICT is creating innovative systems and extending existing infrastructure to such an interconnected complexity that predicting the effects of small internal changes (e.g. firewall policies) and external changes (e.g. the discovery of new vulnerabilities and exploit mechanisms) becomes a major problem. The security of such a complex networked system essentially depends on a concise specification of security goals, their correct and consistent transformation into security policies and an appropriate deployment and enforcement of these policies. This has to be accompanied by a concept to adapt the security policies to changing context and environment, usage patterns and attack situations.

A major source of security vulnerability is any kind of misconfiguration that enables an attacker to exploit a vulnerability in order for a failure to occur [Nicol et al., 2004]. Known and unknown vulnerabilities may be inherent to each of the connected components and communication paths between them. In this thesis it is assumed that the *attack surface* of a networked system with respect to a certain kind of attackers is the part of the system which can be misused by attackers of a certain strength. Thus, the attack surface is an indicator of a networked system's security.

The main aim of this thesis with respect to *security of system configurations* is to provide an approach for model-based analysis of system configurations in terms of situational awareness of the security state. In particular, it should provide means to analyse whether the presence of vulnerabilities (i.e., internal faults that enable external faults to harm the system [Avizienis et al., 2004]) within components is protected or hidden by the network security configuration. This aim motivates the following objectives.

**Objective 5** (Configure systems so that vulnerabilities are protected or hidden)**.**

In order to understand the complex interrelations of security policies, ICT infrastructure and vulnerabilities and to validate security goals in such a setting, tool-based modelling techniques are required that can efficiently and precisely predict and analyse the behaviour of such complex interrelated systems. These methods should guide a systematic evaluation of a given network security policy and assist the persons in charge with finally determining exactly what really needs protection and which security policy to apply. Therefore, the following research questions have been identified.

**RQ 5a.** *How can exploitation possibilities of networked systems' vulnerabilities be analysed?*

**RQ 5b.** *How can attacker behaviour be incorporated into the system model and the analysis?*

**RQ 5c.** *Which attacks would not be detected?*

For this type of analytical analysis, this thesis describes a formal modelling framework that, on the one hand, represents the information system including its vulnerabilities and the security policy, and, on the other hand, a model of attacker capabilities and profile. An extension of the model by intrusion detection components can be used to identify stealth attacks. Based on such an operational model, a graph representing all possible attack paths can be automatically computed. It is called an *attack graph* in the following text. Based on this attack graph, it is now possible to find out whether a given security policy successfully blocks attack paths and is robust against changes in the given vulnerability setting.

**Objective 6** (Identify network configuration risks)**.**

To ensure that security risks are managed cost-effectively, it is necessary to analyse configuration options and likely attacker behaviour.

However, it is usually impossible to visualise an attack graph of a realistic system directly because of its huge size. Therefore, abstract representations of an attack graph are needed that enable to visualise and analyse compacted information focussed on interesting aspects of possible attacker behaviour and countermeasures.

**RQ 6a.** *What are the effects of changes to the network configuration on overall vulnerability?*

**RQ 6b.** *What is the most likely attacker behaviour and most effective countermeasure?*

**RQ 6c.** *Will countermeasures of the system under attack succeed?*

This thesis makes use of alphabetic language homomorphisms to define the abstract views, minimal automaton techniques [Hopcroft & Ullman, 1979] to compute the abstract representations, and shortest path algorithms to discover the most important paths.

Liveness (in this context often called survivability) comes into play, if part of the behaviour of the network is also modelled. With this information it is possible to analyse effects of countermeasures that the system performs under attack or the behaviour of vital services it provides.

**Objective 7** (Assess zero-day exploit vulnerability)**.**

A *zero-day vulnerability* is a vulnerability that is exploited prior to being publicly known [Turner, 2007]. Zero-day vulnerabilities represent a serious threat because at the time of exploitation no patches are available and exploits based on these vulnerabilities will probably not be detected by signature-based Intrusion Detection Systems (IDSs).

*Zero-day attacks* can be defined as attacks which use unknown vulnerabilities [Kotenko & Chechulin, 2012]. This motivates the following research question.

**RQ 7.** *To which extent is a networked system resilient against exploits of unknown vulnerabilities?*

This thesis considers resilience of an information infrastructure against attacks to unknown vulnerabilities by defining a new vulnerability for each installed product.

**Objective 8** (Validate implementation of security goals)**.**

To ensure compliance with laws and regulations, it is necessary to analyse security policies with respect to security and safety goals.

Security policies provide a well-understood and suitable means to administer security issues. However, using policies in distributed environments where applications, services and nodes dynamically join and leave the system raises additional questions.

**RQ 8.** *Does a policy correctly implement high-level security goals?*

This thesis describes a policy validation component that can be used to prove, whether a policy correctly implements given security goals. It supports a subset of Xtensible Access Control Markup Language (XACML) that comprises the most important elements of the language. Furthermore, a concept to deploy an XACML policy using the Common Open Policy Service (COPS) protocol [Durham et al., 2000] has been developed.

Security at configuration time is the main subject of Chapter 3 of this thesis.

### 1.1.3 *Objectives with respect to predictive security analysis at runtime*

Enforcing security and dependability in *process-aware* information systems at runtime requires the monitoring of system operation using process information. Analysis of this information with respect to security and compliance aspects is growing in complexity with the increase in functionality, connectivity, and dynamics of process evolution. To tackle this complexity, the application of models is becoming standard practice. Considering today's frequent changes to processes, model-based support for security and dependability analysis is not only needed in pre-operational phases but also at runtime.

The aim of this thesis with respect to *predictive security analysis at runtime* is to support model-based evaluation of the current security status of process instances as well as to allow for decision support by analysing close-future process states. In order to reach this goal the following objectives have to be addressed.

**Objective 9** (Develop a security monitoring approach)**.**

It is necessary to develop advanced techniques for the evaluation of security-related events and their interpretation with respect to the known control-flow of the processes involved and the required security properties. These techniques should enable methodologies for performing dynamic predictive process analysis at runtime, detecting any prospective potential violation of the required security properties.

**RQ 9a.** *How can operational models reflect the state of observed systems and thus capture abstractions of runtime behaviour?*

**RQ 9b.** *How can operational process models be used for early detection of and reaction to deviations of process execution from its specification?*

This thesis presents an approach to support model-based evaluation of the current status of business process instances as well as to allow for decision support by analysing close-future process states. The approach is based on operational formal models derived from process specifications.

**Objective 10** (Validate security compliance at runtime)**.**
In addition to the predicted process behaviour, the security model is needed to identify security relevant states of the business process.

**RQ 10.** *How can security analysis at runtime exploit process models to identify current and close-future violations of security requirements?*

This thesis proposes the derivation of security and compliance models from high-level security and safety goals. Monitor automata are introduced as a formal notation for the security model.

**Objective 11** (Integrate security management)**.**
It is furthermore important to overcome the contextual restrictions of existing solutions, with their predefined and closed models, and rather to provide an extensible model that spans all parts of the security monitoring and decision support process.

**RQ 11.** *How can security analysis at runtime be integrated in a security management strategy?*

This thesis proposes a Security Strategy Meta Model (SSMM) that supports an integration of functionalities of all parts of the security monitoring and decision support process, namely: (i) detecting threatening events; (ii) putting them in context of the current system state; (iii) explaining their potential impact with respect to some security- or compliance model; and (iv) taking appropriate actions.

**Objective 12** (Provide evidence for usability in large scale industrial scenarios)**.**
Finally, the developed methods should be implemented into a prototype, featuring a new generation, intelligent, multi-domain security event-processing and predictive security monitoring and simulation.

**RQ 12a.** *Can the developed methods and tools be successfully adapted to large scale industrial scenarios?*

**RQ 12b.** *What are the performance effects of the number of events, processes, security requirements, predicted steps, and of event abstraction?*

This thesis evaluates the adaptability and performance of the proposed solution in several industrial scenarios using requirements and data from the scenario providers.

Security at runtime is the main subject of Chapter 4 of this thesis. In summary, this thesis covers the research objectives shown in Figure 3.

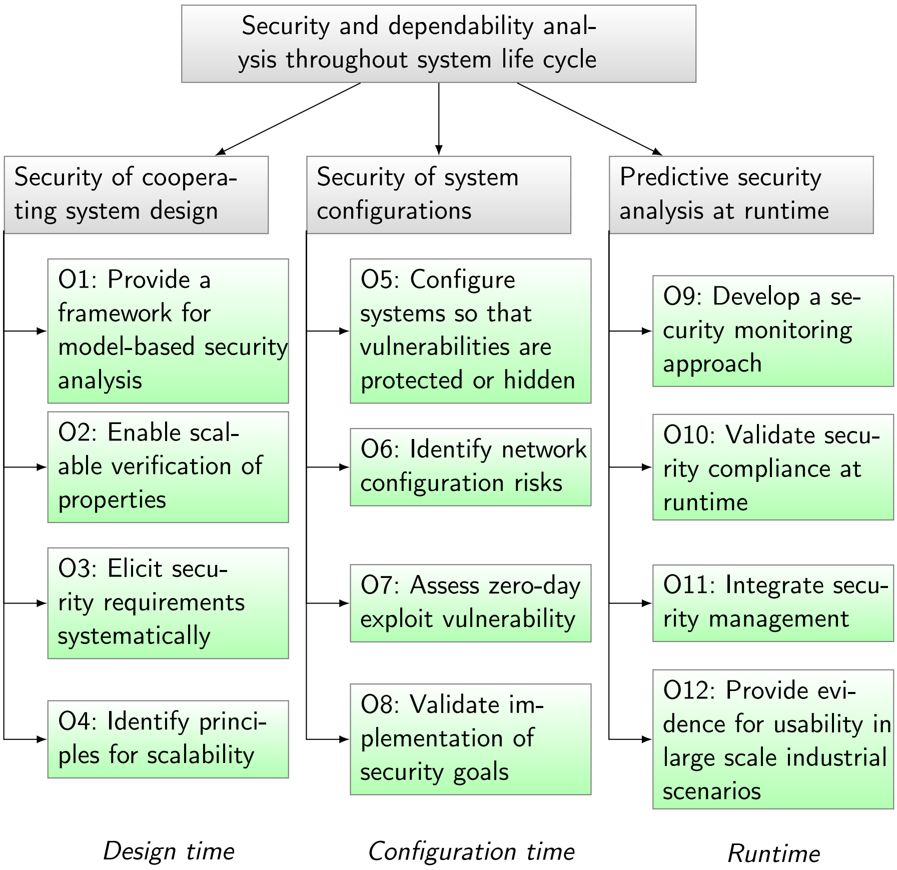

Security and dependability analysis throughout system life cycle
Security of cooperating system design
Security of system configurations
Predictive security analysis at runtime
O1: Provide a framework for model-based security analysis
O2: Enable scalable verification of properties
O3: Elicit security requirements systematically
O4: Identify principles for scalability
O5: Configure systems so that vulnerabilities are protected or hidden
O6: Identify network configuration risks
O7: Assess zero-day exploit vulnerability
O8: Validate implementation of security goals
O9: Develop a security monitoring approach
O10: Validate security compliance at runtime
O11: Integrate security management
O12: Provide evidence for usability in large scale industrial scenarios
Design time
Configuration time
Runtime

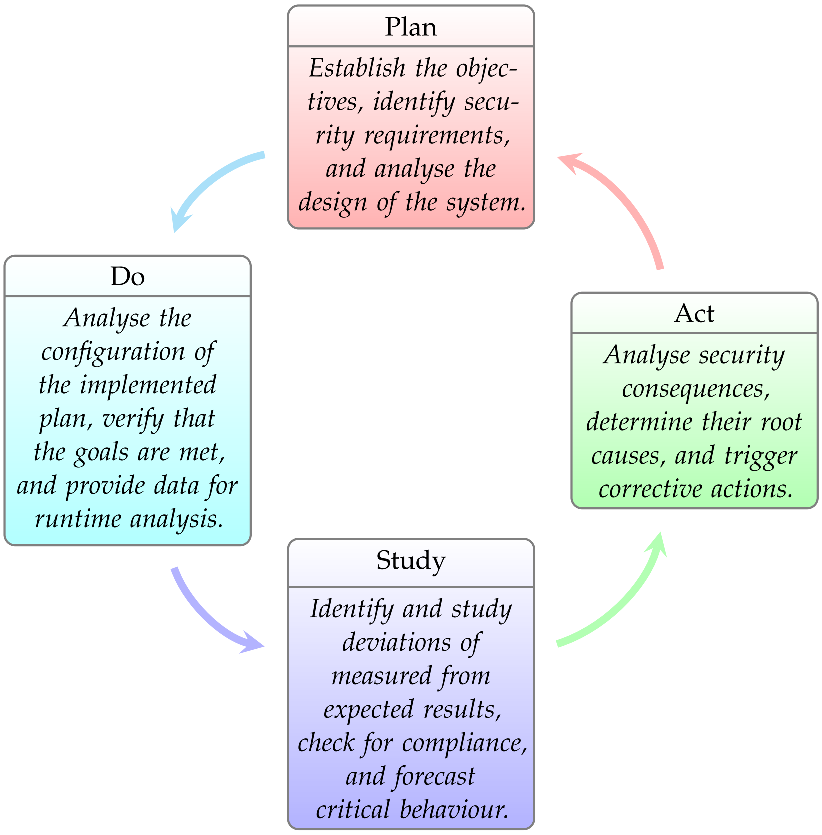
Figure 3: Research objectives

### 1.1.4 *Security topics in practice*

The selection of security topics to be addressed in this thesis has been influenced by practical questions and needs from several research projects in which the author of this thesis has been involved. For example, the projects ProOnline-VSDD [Rieke, 2009b], NoW [Rieke & Steinemann, 2007; Festag et al., 2008], and EVITA [Fraunhofer SIT, 2011] required *model-based security analysis* in order to *prevent faults* *in the design* of the respective system architectures. *Security requirements elicitation* was needed in EVITA [Ruddle et al., 2009] and MASSIF [Repp & Rieke, 2011]. *Scalable verification of security and safety properties* was required in the projects NoW, EVITA, and ProOnline-VSDD. *Attack graph analysis* was used for *fault tolerance analysis* in the projects SKe [Sarbinowski, 2002] and MASSIF [MASSIF project consortium, 2013b]. *Policy validation at configuration time* was required in the project SicAri [Rieke & Ebinger, 2008], and *policy compliance at runtime* is subject in the project MASSIF. *Model-based runtime monitoring* of security properties was required in the projects ADiWa[ADiWa Konsortium, 2012] and MASSIF. *Behaviour anomaly detection* was subject of project MASSIF and is subject of project ACCEPT [Philipps-Universität Marburg, 2013].

## 1.2 RESEARCH CONTRIBUTIONS

The results of this thesis provide a framework for security analysis of system behaviour. Operational models are utilised at design time, configuration time and at runtime.



Plan

*Establish the objectives, identify security requirements, and analyse the design of the system.*

Do

*Analyse the configuration of the implemented plan, verify that the goals are met, and provide data for runtime analysis.*

Study

*Identify and study deviations of measured from expected results, check for compliance, and forecast critical behaviour.*

Act

*Analyse security consequences, determine their root causes, and trigger corrective actions.*

Figure 4: Deming's PDSA cycle adapted to security analysis

Figure 4 illustrates the cyclical security analysis activities addressed by this thesis in relation to the PDSA cycle [Deming, 1993] for the establishment of security critical systems as well as for continuous monitoring and improvement of security during their life cycle.

The methods and tools provided by the results of this thesis contribute to the steps of the PDSA cycle as follows.

**PLAN**

The thesis provides a method to identify security requirements and express them formally; it provides methods and tools to analyse system design with respect to given requirements; and, it provides some construction principles that have to be taken into account in the design of *well-behaved* scalable systems. Thus, it addresses fault prevention and fault removal in the early stages of the security engineering process.

**DO**

The thesis provides methods and tools to analyse the exposition of vulnerabilities in the software components of a networked system to exploitation by internal or external threats. This allows the security assessment of alternative system configurations and thus to minimise the attack surface of the networked system and mitigate potential impact of successful attacks. Furthermore, methods and tools to validate and deploy security policies are provided.

**STUDY**

The thesis provides a method and tool to observe - *de-facto* - behaviour of processes, compare it with the planned - *de-jure* - behaviour, and evaluate security compliance at runtime. Where applicable, knowledge on process' expected behaviour is used for forecasting critical situations in the near future. The reported results also take into account other relevant context information such as the current attack state for review and countermeasure assessment. Furthermore, a concept to integrate the tool into a holistic security management strategy is proposed.

**ACT**

The thesis provides methods and tools to analyse possible corrective and preventive actions - based on the results of the security assessment - as well as a tool to trigger their execution in order to achieve continual improvement of the system.

In the following three subsections research results provided in this thesis will be summarised with respect to each of the research questions identified in Section 1.1. The results are provided by *19* selected peer-reviewed papers. For each paper Part III provides a separate chapter (P1–P19) with a factsheet, an abstract describing the content of the paper and its contribution to this cumulative thesis, and a copy of the paper.

### 1.2.1 *Results with respect to security of cooperating system design*

The research results of this thesis with respect to *security of cooperating system design* contribute to the security engineering process. Within this process the aim of this thesis is to support the analysis of a system design at an early stage based on an *operational model* of the system. For a full coverage of the security engineering process the verification of the refinement of the model up to the implementation and the analysis of the implementation at the source-code level is of course necessary but these topics are out of scope of this thesis.

Operational models such as Petri nets [Petri, 1962] or Asynchronous Product Automata [Ochsenschläger et al., 1998] – which are used in this thesis – are executable. They are in general qualitative, and are thus at a higher abstraction level than quantitative models and they are easier to analyse [Bonzanni et al., 2009] (because they are executable). If needed, operational models can be *enriched* by specific quantitative information when they are used in combination with monitoring information from a running system (cf. Chapter 4).

An APA consists of a family of so called *elementary automata* communicating by *common components of their state* (shared memory). The applied specification method based on APA is supported by the Simple Homomorphism Verification Tool (SHVT) [Ochsenschläger et al., 1998, 2000a; Ochsenschläger et al., 2002; Fraunhofer SIT, 2009]. The SHVT has been developed at the *Fraunhofer-Institute for Secure Information Technology* with significant participation of the author of this thesis in the design and implementation of the tool. It provides components for the complete cycle from formal specification to exhaustive validation as well as visualisation and inspection of computed reachability graphs and minimal automata. In Hartel et al. [1999] ten different formal methods and tools in this area including an old version of the SHVT have been compared with respect to strengths and weaknesses in supporting the modelling activity. It was concluded that it is useful for specifications of systems at the architectural level, to compare different designs, and to search for errors in high-level designs.

**Result 1** (APA, TL and verification tool [P1, P2, P3])**.**
Adressing research question

RQ1 *How can it be proven that components of cooperating systems securely work together?*

a number of contributions to the model-based analysis approach and the SHVT have been provided. Specific model checking algorithms for Temporal Logic (TL) properties have been designed and implemented within the SHVT, for example, the construction of a Büchi automaton [Clarke et al., 1999] representing the property given by a Propositional Linear Temporal Logic (PLTL) formula [Gerth et al., 1996; Clarke et al., 1999; Peled, 2001], the construction of the synchronous product of the automaton of property and system behaviour, construction of the complement automaton, the construction of the intersection with the automaton representing the behaviour, and a check whether the resulting automaton is empty. The method for checking *approximate satisfaction* of properties fits exactly to the already existing built-in simple homomorphism check. These results have been published in P1 [Ochsenschläger, Repp, Rieke & Nitsche, 1998] and P2 [Ochsenschläger, Repp & Rieke, 2000a]. P3 [Rieke, 2003] provides an extensive example for the use of the methods and tool described in P1 and P2. It has been shown in  Apel, Repp, Rieke & Steingruber [2007] that operational models can also be used for test case generation to validate that a system behaves as specified.

**Result 2** (Abstraction based verification [P4])**.**
Because of state space explosion problems, traditional model checking techniques allow a verification of behaviour properties only for systems with very few components. To be able to verify entire families of critical systems, independent of the exact number of replicated components, an *abstraction based approach* has been developed that extends tool supported verification techniques to such parameterised systems. Abstraction is a fundamental and widely-used verification technique. It can be used to reduce the verification of a property over a concrete system, to checking a related property over a simpler abstract system [Ochsenschläger, Repp & Rieke, 2000]. This allows the verification of parameterised systems by constructing abstract systems that can be model checked and thus contributes to research question

RQ2 *How can finite state verification techniques be extended to prove properties independently of concrete parameters?*

In P4 [Ochsenschläger & Rieke, 2007], a verification concept is proposed that makes use of an inductive proof on the construction of the behaviour of the parameterised system to show that it results in identical abstract system behaviour for any given parameter configuration.

**Result 3** (Authenticity requirements identification [P5, P6])**.**
The first step in the design of an architecture for a novel system is the *requirements engineering process*. With respect to security requirements this process typically covers the identification of the principal security goals, the actual security requirements elicitation process, and a requirements categorisation and prioritisation, followed by requirements inspection [Mellado et al., 2007; Mead & Hough, 2006; Mead, 2007].

The approach provided in P5 [Fuchs & Rieke, 2010] addresses the security requirements elicitation process. In particular, a systematic and constructive approach for the *identification of a consistent and complete set of authenticity requirements* has been developed. The method is based on functional dependency analysis. It avoids premature assumptions on the architectural structure and mechanisms to implement security measures which solves several issues compared to existing approaches [Firesmith, 2003]. In P6 [Coppolino, Jäger, Kuntze & Rieke, 2012] it is shown by means of a representative example, namely, a hydroelectric power plant in a dam, how the elicitation method can be applied in order to analyse security threats for critical infrastructures. Thus research question

RQ3 *How can security requirements for cooperating systems be elicited systematically?*

has been solved for *authenticity* requirements. *Integrity* can be expressed in terms of *authenticity within a phase* [Gürgens et al., 2005]. Furthermore, *authenticity* is closely related to *accountability*, and *non-repudiability* [Avizienis et al., 2004]. It is still an open question whether a similar approach works for other security properties such as *confidentiality* and *availability*.

**Result 4** (Parameterised verification problem reduced to finite state [P7,P8]).

Behaviour properties of systems can be divided into two classes: *safety* and *liveness* properties [Alpern & Schneider, 1985]. Intuitively a safety property stipulates that "something bad does not happen" and a liveness property stipulates that "something good eventually happens". In P7 [Ochsenschläger & Rieke, 2011] it is shown that for *well-behaved* scalable systems a wide class of safety properties can be verified by finite state methods. To extend this verification approach to reliability or general liveness properties, additional assumptions for well-behaved scalable systems have to be established. In P8 [Ochsenschläger & Rieke, 2012a] such assumptions have been developed for uniformly parametrised two-sided cooperations. With respect to research question

RQ4 *Which design principles facilitate verifiability of security properties of scalable systems?*

the most important insight from this research is that *behavioural self-similarity* is a key property which should be addressed in any design of a well-behaved scalable system. Behavioural self-similarity formalises the property that from an abstracting point of view, where only actions of some selected partners are considered, the complex system of all partners behaves like the smaller subsystem of the selected partners. However, this research path has opened a lot more interesting new questions which should be addressed in further research (cf. P8).

Figure 5 depicts the relations between the objectives, research questions, and results of this thesis with respect to *security of cooperating system design*.

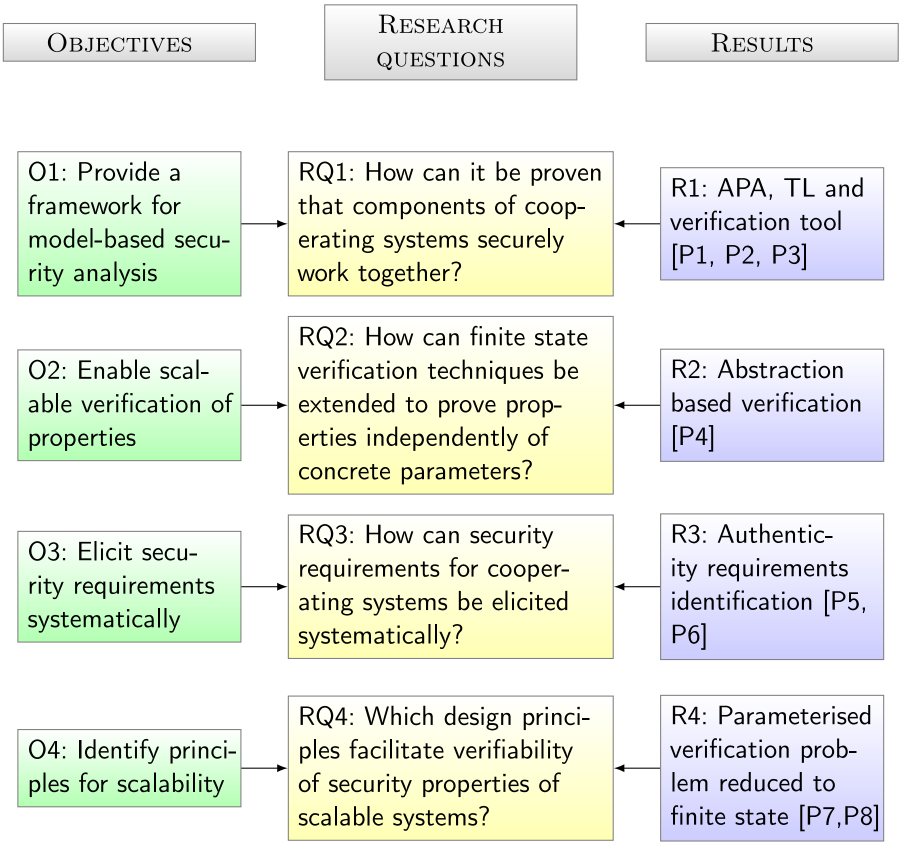

OBJECTIVES
RESEARCH QUESTIONS
RESULTS
O1: Provide a framework for model-based security analysis
RQ1: How can it be proven that components of cooperating systems securely work together?
R1: APA, TL and verification tool [P1, P2, P3]
O2: Enable scalable verification of properties
RQ2: How can finite state verification techniques be extended to prove properties independently of concrete parameters?
R2: Abstraction based verification [P4]
O3: Elicit security requirements systematically
RQ3: How can security requirements for cooperating systems be elicited systematically?
R3: Authenticity requirements identification [P5, P6]
O4: Identify principles for scalability
RQ4: Which design principles facilitate verifiability of security properties of scalable systems?
R4: Parameterised verification problem reduced to finite state [P7,P8]

Figure 5: Security of cooperating system design: Objectives, research questions, and results.

### 1.2.2 *Results with respect to security of system configurations*

Security configuration is necessary to reflect the evolution of security requirements during the life cycle of a system. It provides a proactive approach to the management of changing security policies. It furthermore mitigates problems emerging from design or implementation flaws and enables appropriate reaction to novel threats to which the system is exposed.

The research results of this thesis with respect to *security of system configurations* facilitate the identification of critical security risks related to system vulnerabilities and the evaluation of different configuration variants with regard to attack surface, detection as well as impact mitigation aspects.

**Result 5** (Attack graph model [P9])**.**

In P9 [Rieke, 2004b] an analysis approach has been developed that builds on a model-based construction of an attack graph. The proposed operational model comprises, (1) an asset inventory including critical network components, topology and vulnerability attributions, (2) a network security policy, (3) vulnerability specifications and exploit descriptions, and (4) an attacker model taking into account the attackers knowledge and behaviour. This answers research question

RQ5A *How can exploitation possibilities of networked systems' vulnerabilities be analysed?*

In order to answer research question

RQ5B *How can attacker behaviour be incorporated into the system model and the analysis?*

attacker capabilities are modelled by atomic exploits and by a strategy to select and apply them. Several different attackers can easily be included because an attacker is modelled as a role not a single instance and the tool can automatically generate multiple instances from one role definition. P9 further describes how the research question

RQ5C *Which attacks would not be detected?*

can be answered by introducing intrusion detection components into the system model.

**Result 6** (Abstraction based analysis method [P10])**.**

In consecutive work published in P10 [Rieke, 2006], generic formal templates for the specification of vulnerabilities and exploits have been developed that allow to easily extend the model. A unique feature of the approach is that abstract representations of attack graphs can be computed that allow visualisation of specific aspects and comparison of focussed views on the behaviour of the system. The impact of changes to security policies or network structure can be computed and visualised by differences in the respective attack graphs. Results of this analysis support the process of dependable configuration of critical information infrastructures. This answers research question

RQ6a *What are the effects of changes to the network configuration on overall vulnerability?*

P10 further shows, how additional assignments of cost benefit ratings to the exploits in the model and additional analysis features on the abstract representation such as shortest path computations [Aho & Ullman, 1995] solve the research question

RQ6b *What is the most likely attacker behaviour and most effective countermeasure?*

When in addition a system's countermeasures and the behaviour of vital services the system provides are included in the model, then these effects and the system's resilience can be analysed, which answers research question

RQ6c *Will countermeasures of the system under attack succeed?*

**Result 7** (Model of unknown vulnerabilities [P11]).

In order to analyse resilience of systems against exploits of unknown vulnerabilities, generic vulnerabilities for each installed product and affected service are added to the model. The reachability analysis now considers every possible choice of product, and so all alternatives are evaluated in the attack graph, which answers research question

RQ7 *To which extent is a networked system resilient against exploits of unknown vulnerabilities?*

This capability has been added to the approach of P10 and was published in P11 [Rieke, 2008a]. To the best of the knowledge of the author of this thesis, this was the first publication approaching this research question. The analysis of network security against unknown zero-day attacks is still an active research topic [Ingols et al., 2009; Wang et al., 2010; Kotenko & Chechulin, 2012].

**Result 8** (Policy validation concept and tool [P12]).

In order to address the enforcement of a security policy within a system, an approach for policy validation and deployment based on the middleware provided by the SicAri platform [Peters, 2013] has been developed and published in P12 [Peters, Rieke, Rochaeli, Steinemann & Wolf, 2007], which answers research question

RQ8 *Does a policy correctly implement high-level security goals?*

The validation of a security strategy which is given by an Organization Based Access Control (OrBAC) policy has been demonstrated in Ochsenschläger, Rieke & Velikova [2008].

Figure 6 depicts the relations between the objectives, research questions, and results of this thesis with respect to *security of system configurations*.

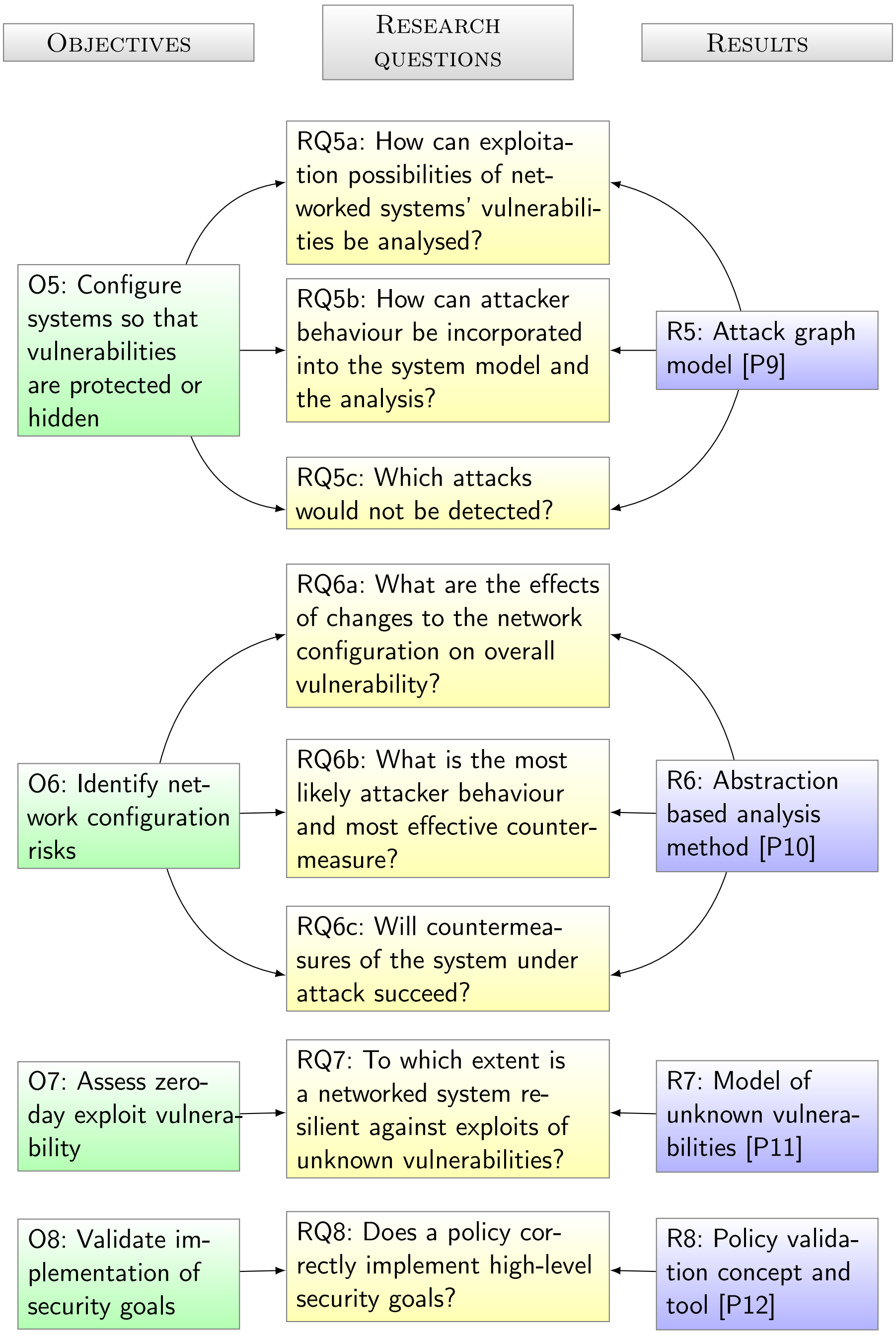

OBJECTIVES
RESEARCH QUESTIONS
RESULTS
RQ5a: How can exploitation possibilities of networked systems' vulnerabilities be analysed?
O5: Configure systems so that vulnerabilities are protected or hidden
RQ5b: How can attacker behaviour be incorporated into the system model and the analysis?
R5: Attack graph model [P9]
RQ5c: Which attacks would not be detected?
RQ6a: What are the effects of changes to the network configuration on overall vulnerability?
O6: Identify network configuration risks
RQ6b: What is the most likely attacker behaviour and most effective countermeasure?
R6: Abstraction based analysis method [P10]
RQ6c: Will countermeasures of the system under attack succeed?
O7: Assess zero-day exploit vulnerability
RQ7: To which extent is a networked system resilient against exploits of unknown vulnerabilities?
R7: Model of unknown vulnerabilities [P11]
O8: Validate implementation of security goals
RQ8: Does a policy correctly implement high-level security goals?
R8: Policy validation concept and tool [P12]

Figure 6: Security of system configurations: Objectives, research questions, and results.

### 1.2.3 *Results with respect to predictive security analysis at runtime*

The research results of this thesis with respect to *predictive security analysis at runtime* contribute to a new model-based approach for Predictive Security Analysis at Runtime (PSA@R). In PSA@R the operation of a system or SoS is observed by analysing events received from this system. PSA@R is not executed by this observed system but rather by an observing and reacting system such as a Security Information and Event Management (SIEM) system [MASSIF project consortium, 2013b].

The thesis exemplifies the approach utilising processes from several industrial scenarios and demonstrates the systematic development and application of models for PSA@R.

**Result 9** (Process monitoring and uncertainty management [P13,P18])**.** P13 [Rieke & Stoynova, 2010] provides a blueprint of the architecture of PSA@R and describes the process modelling techniques in detail. It is assumed that the purpose of the observed system is given by technical or business processes and that the intended behaviour can be specified by process models. In order to support evaluation of the security status of these processes at runtime PSA@R uses operational formal models derived from process specifications and security policies. The behaviour of the observed system is given by the behaviour of the set of involved processes. This behaviour - viewed as a sequence of actions - is the *shuffle* of the behaviour of each process taken in isolation [Sakarovitch, 2009].

To answer research question

RQ9a *How can operational models reflect the state of observed systems and thus capture abstractions of runtime behaviour?*

the process behaviour model is synchronised with the running process through events received from the execution environment. The events from the monitored system are mapped to the abstract model. The behaviour of the observed system is thus traced in the model state that reflects the states of current processes within the model.

With respect to research question

RQ9b *How can operational process models be used for early detection of and reaction to deviations of process execution from its specification?*

it is shown how a (partial) representation of the process behaviour model with respect to the current state can be computed. This representation can then be used to compare the *de-facto* behaviour given by the event stream with the expected behaviour given by the behaviour model. Furthermore, P18 [Rieke, Zhdanova, Repp, Giot & Gaber, 2013] describes how uncertainty management is used to adjust the specified (*de-jure*) process model to the measured (*de-facto*) behaviour. The incoming events are interpreted using the event model and mapped to the process behaviour model. Uncertainty management is initiated whenever an event does not match the expected states within the behaviour model. Three cases of uncertainty are handled: (1) evolution of the process specification, (2) process measurement problems (e.g., lost or delayed events), and (3) anomalies caused by malicious activity. In case the detected behaviour deviation is considered legitimate, the underlying process model is adapted to the changed conditions.

**Result 10** (Close-future security violation prediction [P14,P19]).
In order to answer research question

RQ10 *How can security analysis at runtime exploit process models to identify current and close-future violations of security requirements?*

a *security model* - specified by *monitor automata* - formally defines security requirements restricting the allowed process behaviour. PSA@R utilises prediction of expected close-future process states to find possible security violations and thus allows early decisions on countermeasures. P14 [Eichler & Rieke, 2011] and P19 [Rieke, Repp, Zhdanova & Eichler, 2014] describe the approach for security requirements monitoring and violation prediction in detail and exemplify it on an application scenario from the logistics domain (P14) and critical infrastructures (P19).

**Result 11** (Security strategy management [P15,P19]).
Addressing research question

RQ11 *How can security analysis at runtime be integrated in a security management strategy?*

P15 [Rieke, Schütte & Hutchison, 2012] describes a framework to integrate PSA@R into a holistic security strategy management system. The proposed SSMM provides a way to model security directives at an abstract level which can be compiled into specific rules for an underlying framework of monitoring, decision support, and enforcement engines. The information from other components of the security strategy management system can provide situational awareness with regard to network state and attack state to PSA@R and thus improve the analysis results.

**Result 12** (Industrial use cases [P14,P16, P17, P18, P19]).
Addressing research question

RQ12A *Can the developed methods and tools be successfully adapted to large scale industrial scenarios?*

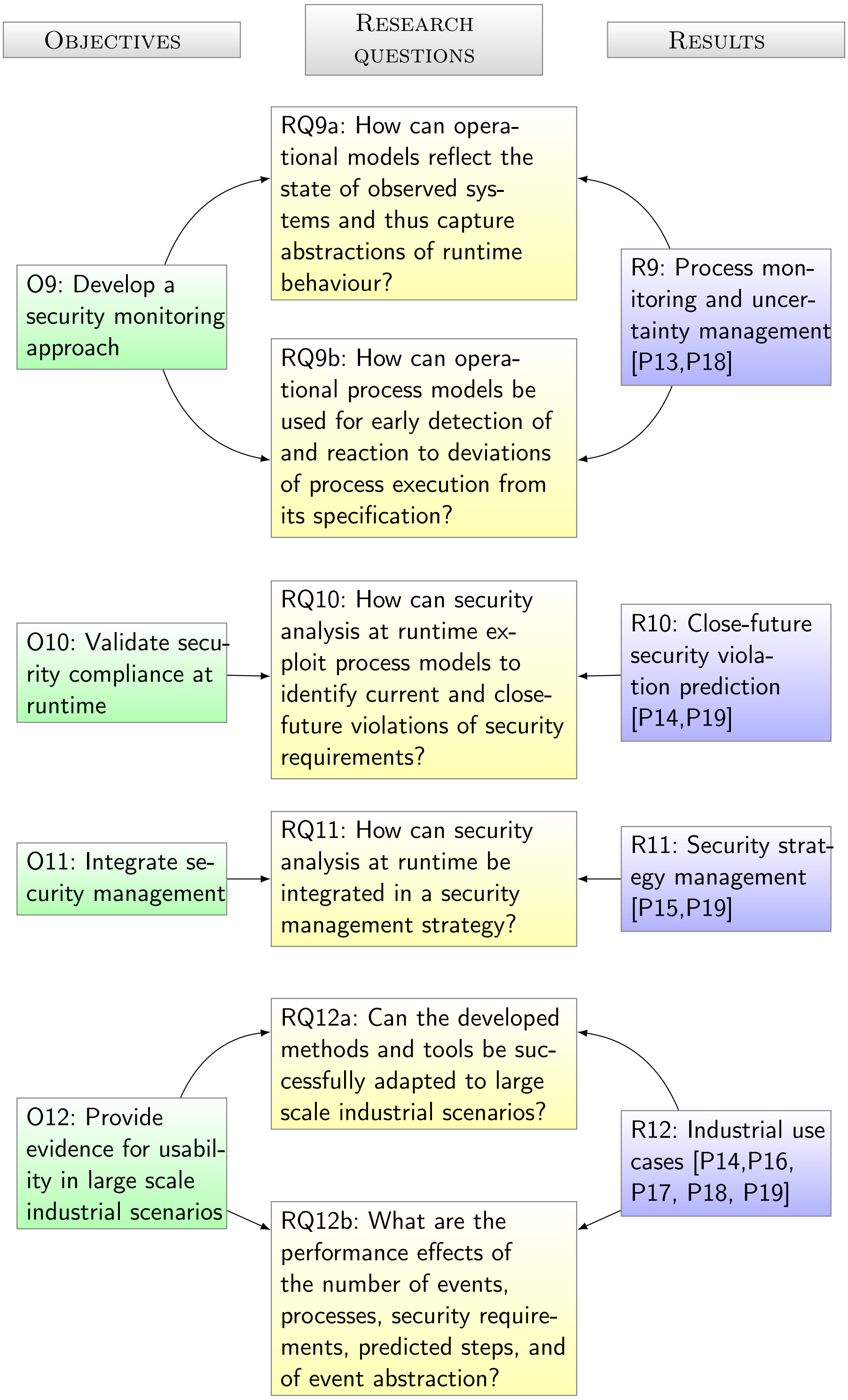
a prototype Predictive Security Analyser (PSA) has been implemented by Fraunhofer SIT headed by the author of this thesis in order to evaluate the applicability and performance of different modelling strategies in the scope of PSA@R. The PSA supports the complete span of application of PSA@R from formal process specification to exhaustive validation. The application context for the developed solution is given by the requirements from industrial scenarios documented in P16 [Prieto, Diaz, Romano, Rieke & Achemlal, 2012] and P17 [Rieke, Coppolino, Hutchison, Prieto & Gaber, 2012].

With respect to research question

RQ12B *What are the performance effects of the number of events, processes, security requirements, predicted steps, and of event abstraction?*

first promising results on performance are provided by the tool adaptation to the scenarios of the MAnagement of Security information and events in Service InFrastructures (MASSIF) project. Specifically, for an MMTS scenario this is published in P18 [Rieke, Zhdanova, Repp, Giot & Gaber, 2013] and for a critical infrastructure scenario in P19 [Rieke, Repp, Zhdanova & Eichler, 2014].

Figure 7 depicts the relations between the objectives, research questions, and results of this thesis with respect to *predictive security analysis at runtime*.



O9: Develop a security monitoring approach

RQ9a: How can operational models reflect the state of observed systems and thus capture abstractions of runtime behaviour?

RQ9b: How can operational process models be used for early detection of and reaction to deviations of process execution from its specification?

R9: Process monitoring and uncertainty management [P13,P18]

O10: Validate security compliance at runtime

RQ10: How can security analysis at runtime exploit process models to identify current and close-future violations of security requirements?

R10: Close-future security violation prediction [P14,P19]

O11: Integrate security management

RQ11: How can security analysis at runtime be integrated in a security management strategy?

R11: Security strategy management [P15,P19]

O12: Provide evidence for usability in large scale industrial scenarios

RQ12a: Can the developed methods and tools be successfully adapted to large scale industrial scenarios?

RQ12b: What are the performance effects of the number of events, processes, security requirements, predicted steps, and of event abstraction?

R12: Industrial use cases [P14,P16, P17, P18, P19]

Figure 7: Predictive security analysis at runtime: Objectives, research questions, and results.

## 1.3 thesis organisation

This cumulative thesis comprises four parts: Part I gives an introduction; Part II provides an extensive summary of the results achieved with specific consideration of the connection to the individual papers; Part III contains the peer-reviewed articles; Part IV is an appendix with personal information on the author. These parts are now described in more detail.

part i. Part I provides an overview of the research work presented in this thesis. It gives the background and motivation for the work and describes in detail the research objectives. Furthermore, it provides an overview of the approach taken as well as of the main results obtained. The links between the research questions and the corresponding publications are illustrated in the summaries of the three research topics (cf., Figure 5, Figure 6, and Figure 7). Finally, the structure of the thesis is introduced.

part ii. Part II of this thesis gives an overview about the research process and its results.

Chapter 2 provides a framework for operational modelling of systems and computation of possible system behaviour. On the basis of such behaviour models, the analysis and verification of safety and security properties is possible. It also comprises the elicitation of security requirements in the context of SoS and provides results from theory to apply the results of analysis of small-scale prototype systems to large-scale systems composed of many identical components.

Chapter 3 aims at a systematic security assessment of the configuration of information infrastructures taking into account constraints given by a network security policy. To achieve this objective, a model-based construction of an attack graph and appropriate abstractions are utilised that enable focussed views on the system with respect to possible attacks. It furthermore introduces a framework for enforcement of security requirements by configuration of security policies.

Chapter 4 complements the *security by design* approach by an *security at runtime* approach. Extensions to the modelling framework support real-time observation of a SoS and thus allow for *on-the-fly security reasoning* utilising knowledge about *de-jure* and *de-facto* behaviour. The applicability of the developed methods in complex large-scale application scenarios is demonstrated and the basic performance of the developed tools is evaluated.

Chapter 5 evaluates the research proposition and the outcomes planned in Chapter 1. Furthermore, it considers current and future application domains for the approach, open issues, and lessons learnt.

Part II ends with a bibliography of related work.

PART III. Part III of this cumulative thesis consists of previously reviewed and published research results. The papers have been selected based on their contribution to the research objectives of this thesis. Each publication is introduced with a fact sheet which describes the contribution of the author of this thesis in detail. Table 1, Table 2, and Table 3 list the papers considered for this cumulative dissertation. They are not sorted in their actual timeline, but rather based on their contribution to the objectives and research questions. For each publication the authors, the title, the publication outlet, and the publication type, namely, workshop proceedings (WS), conference proceedings (CON), book chapter (BC), or journal (JNL) is given. The contribution of the author of this thesis is marked as follows: ● – only author, ◕ – main author, ◐ – co-author ranking equally, ◔ – co-author with significant contribution.

| Pub. | Authors | Title | Outlet | Type |
|---|---|---|---|---|
| P1<br>◔ | Ochsenschläger, Repp, Rieke, Nitsche | The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems | Springer | JNL |
| P2<br>◔ | Ochsenschläger, Repp, Rieke | The SH-Verification Tool | AAAI | CON |
| P3<br>● | Rieke | Development of formal models for secure e-services | Eicar | CON |
| P4<br>◐ | Ochsenschläger, Rieke | Abstraction Based Verification of a Parameterised Policy Controlled System | Springer LNCS | CON |
| P5<br>◐ | Fuchs, Rieke | Identification of Security Requirements in Systems of Systems by Functional Security Analysis | Springer LNCS | BC |
| P6<br>◕ | Coppolino, Jäger, Kuntze, Rieke | A Trusted Information Agent for Security Information and Event Management | IARIA | CON |
| P7<br>◔ | Ochsenschläger, Rieke | Security Properties of Self-similar Uniformly Parameterised Systems of Cooperations | IEEE | CON |
| P8<br>◔ | Ochsenschläger, Rieke | Reliability Aspects of Uniformly Parameterised Cooperations | IARIA | CON |

Table 1: Publications contributing to Chapter 2

PART IV. Part IV is the appendix. It starts with a short curriculum vitae of the author with respect to research work and publications followed by a declaration of the author of this cumulative thesis.

| Pub. | Authors | Title | Outlet | Type |
|---|---|---|---|---|
| P9 ● | Rieke | Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration | Eicar | CON |
| P10 ● | Rieke | Modelling and Analysing Network Security Policies in a Given Vulnerability Setting | Springer LNCS | CON |
| P11 ● | Rieke | Abstraction-based analysis of known and unknown vulnerabilities of critical information infrastructures | Inderscience | JNL |
| P12 ◔ | Peters, Rieke, Rochaeli, Steinemann, Wolf | A Holistic Approach to Security Policies – Policy Distribution with XACML over COPS | Elsevier | JNL |

Table 2: Publications contributing to Chapter 3

| Pub. | Authors | Title | Outlet | Type |
|---|---|---|---|---|
| P13 ◐ | Rieke, Stoynova | Predictive Security Analysis for Event-Driven Processes | Springer LNCS | CON |
| P14 ◐ | Eichler, Rieke | Model-based Situational Security Analysis | CEUR | WS |
| P15 ◐ | Rieke, Schütte, Hutchison | Architecting a Security Strategy Measurement and Management System | ACM-DL | WS |
| P16 ◔ | Prieto, Diaz, Romano, Rieke, Achemlal | MASSIF: A Promising Solution to Enhance Olympic Games IT Security | Springer LNICST | CON |
| P17 ◐ | Rieke, Coppolino, Hutchison, Prieto, Gaber | Security and Reliability Requirements for Advanced Security Event Management | Springer LNCS | CON |
| P18 ◔ | Rieke, Zhdanova, Repp, Giot, Gaber | Fraud Detection in Mobile Payment Utilizing Process Behavior Analysis | IEEE | WS |
| P19 ◐ | Rieke, Repp, Zhdanova, Eichler | Monitoring Security Compliance of Critical Processes | IEEE | CON |

Table 3: Publications contributing to Chapter 4

Part II

# INTRODUCTION TO THE SUBJECT MATTER AND SUMMARY OF THE RESULTS

*Knowledge is built on theory. The theory of knowledge teaches us that a statement, if it conveys knowledge, predicts future outcome, with risk of being wrong, and that it fits without failure observations of the past.*

— William Edwards Deming [Deming, 1993]

# 2 SECURITY OF COOPERATING SYSTEM DESIGN

> *An automaton is a* structure*, but that tells us nothing - because everything is structure, in other words a set equipped with operations - except that we should remember this when defining a map from one automaton to another. Furthermore, an object can be viewed as a structure in several ways. Since they describe the* same *object, they are equivalent, so can be translated into each other. Each illuminates certain aspects and leaves others obscure. To begin, we have to pick one!*
>
> — Jacques Sakarovitch, Elements of Automata Theory [Sakarovitch, 2009]

AIM OF THIS CHAPTER. This chapter aims to provide a specification and analysis framework for model-based analysis of system behaviour with respect to systematically deduced security and reliability requirements. Extensions of this approach for the verification of scalable systems and design principles that facilitate such verifiability are given.

This chapter is based on the work published in P1, P2, P3, P4, P5, P6, P7, and P8 (cf. Table 1).

## 2.1 INTRODUCTION

For safety critical systems as well as for business critical systems or parts thereof, assuring the correctness - conformance to the intended purpose - is imperative. These systems must guarantee a variety of safety, liveness and security properties. The behaviour of a discrete system can be formally described by the set of its possible sequences of actions. This point of view is important to define security requirements as well as to verify such properties, because for these purposes sequences of actions of the system have to be considered [Schneider, 1996; Zegzhda et al., 2012].

The aim of this thesis with respect to security of cooperating system design is approached by the following four objectives (cf. Figure 3).

O1: Provide a framework for model-based security analysis.

O2: Enable scalable verification of properties.

O3: Elicit security requirements systematically.

O4: Identify principles for scalability.

Addressing objective O1 this chapter introduces the basic method and tool to analyse a system's behaviour with respect to security and reliability requirements. To be suitable for the verification of Cooperating Systems (CS), a modified type of satisfaction relation (approximate satisfaction) is considered and an implementation of a suitable Temporal Logic (TL) variant is described. An approach to investigate an abstract behaviour in order to verify the correctness of the underlying concrete behaviour is given. The problem that under such an abstraction an incorrect part of the behaviour can be hidden by a correct one is addressed.

Traditional model checking techniques allow a verification of the required behaviour only for systems with very few components. In order to be able to verify entire families of systems that are parameterised by a number of replicated identical components, an approach to extend finite state verification techniques to prove properties that are valid independently of the exact number of replicated components is introduced. This enables scalable verification of properties, the aim of objective O2.

CS typically base decisions on information from their own components as well as on input from other systems' components. Safety critical decisions based on cooperative reasoning however raise severe concerns to security issues. Therefore, the security requirements elicitation step for such Systems of Systems (SoS) is addressed. Addressing objective O3 an approach to systematically deduce comprehensive sets of formally defined authenticity requirements with respect to allowed sequences of actions is given.

Finally, addressing objective O4, design principles that facilitate verifiability of security properties of scalable systems are given.

Section 2.6 gives an overview of related work. Finally, this chapter ends with a summary of the results in Section 2.7.

## 2.2 OPERATIONAL MODELLING APPROACH

With respect to *security of cooperating system design*, the first objective of this thesis is to provide means to prove that - in the context of CS - the components work together in a desired manner.

RQ1: *How can it be proven that components of cooperating systems securely work together?*

In order to *verify* that system components work together in a desired manner, the *dynamic behaviour* of the system has to be investigated.

### 2.2.1 *Modelling the dynamic behaviour of a system*

Modelling - in this context - means to represent the system in terms of mathematical objects that reflect its observed properties [Peled, 2001]. In this thesis, the dynamic behaviour of the system will be specified by an operational state model based on Asynchronous Product Automata (APA), a flexible operational specification concept for CS, which was first introduced by Ochsenschläger, Repp, Rieke & Nitsche in P1. An APA consists of a family of so called *elementary automata* communicating by *common components of their state* (shared memory). For a graphical representation of an APA that is supported by the Simple Homomorphism Verification Tool (SHVT) (cf. Section 2.2.4) circles are used to represent state components and rectangles are used to represent elementary automata. Edges between state components and elementary automata represent the neighbourhood relation. It indicates which state components are included in the state of an elementary automaton and may be changed by a state transition of the elementary automaton.

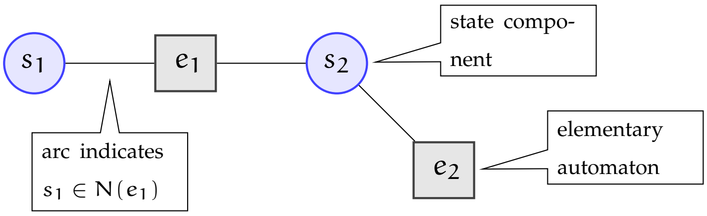


Figure 8: Graphical representation of an APA with two elementary automata $e_1$ and $e_2$ and state components $s_1$ and $s_2$

Formally, an APA is defined as follows.

**Definition 1** (Asynchronous Product Automaton (APA)).
*An APA consists of*

- *a family of* state sets $Z_s, s \in \mathbb{S}$,
- *a family of* elementary automata $(\Phi_e, \Delta_e), e \in \mathbb{E}$ *and*
- *a* neighbourhood relation $N : \mathbb{E} \to \mathcal{P}(\mathbb{S})$.

$\mathbb{S}$ *and* $\mathbb{E}$ *are index sets with the names of state components and of elementary automata and* $\mathcal{P}(\mathbb{S})$ *is the power set of* $\mathbb{S}$. *For each elementary automaton* $(\Phi_e, \Delta_e)$ *with* Alphabet $\Phi_e$, *its* state transition relation *is*

$$\Delta_e \subseteq \bigtimes_{s \in N(e)}(Z_s) \times \Phi_e \times \bigtimes_{s \in N(e)}(Z_s).$$

*For each element of* $\Phi_e$ *the state transition relation* $\Delta_e$ *defines state transitions that change only the state components in* $N(e)$. *An APA's (global)* states *are elements of* $\bigtimes_{s \in \mathbb{S}}(Z_s)$. *To avoid pathological cases it is generally assumed that* $\mathbb{S} = \bigcup_{e \in \mathbb{E}}(N(e))$ *and* $N(e) \neq \emptyset$ *for all* $e \in \mathbb{E}$. *Each APA has* *one* initial state $q_0 = (q_{0s})_{s \in \mathbb{S}} \in \times_{s \in \mathbb{S}}(Z_s)$. *In total, an APA* $\mathbb{A}$ *is defined by*

$$\mathbb{A} = ((Z_s)_{s \in \mathbb{S}}, (\Phi_e, \Delta_e)_{e \in \mathbb{E}}, N, q_0).$$

**Definition 2.** *An elementary automaton* $(\Phi_e, \Delta_e)$ *is* activated *in a state* $q = (q_s)_{s \in \mathbb{S}} \in \times_{s \in \mathbb{S}}(Z_s)$ *as to an* interpretation $i \in \Phi_e$*, if there are* $(p_s)_{s \in N(e)} \in \times_{s \in N(e)}(Z_s)$ *with* $((q_s)_{s \in N(e)}, i, (p_s)_{s \in N(e)}) \in \Delta_e$*. An activated elementary automaton* $(\Phi_e, \Delta_e)$ *can execute a state transition and produce a* successor state $p = (p_s)_{s \in \mathbb{S}} \in \times_{s \in \mathbb{S}}(Z_s)$*, if* $q_r = p_r$ *for* $r \in \mathbb{S} \setminus N(e)$ *and* $(q_s)_{s \in N(e)}, i, (p_s)_{s \in N(e)} \in \Delta_e$*. The corresponding state transition is* $(q, (e, i), p)$*.*

Formally, the *behaviour* of an operational APA model is described by a Reachability Graph (RG).

**Definition 3** (reachability graph)**.** *The behaviour of an APA is represented by all possible coherent sequences of state transitions starting with initial state* $q_0$*. The sequence*

$$(q_0, (e_1, i_1), q_1)\ (q_1, (e_2, i_2), q_2)\ \ldots\ (q_{n-1}, (e_n, i_n), q_n)$$

*with* $i_k \in \Phi_{e_k}$*, where* $\Phi_{e_k}$ *is the alphabet of the elementary automaton* $e_k$*, represents one possible sequence of actions of an APA.*

*State transitions* $(p, (e, i), q)$ *may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA:* $(p, (e, i), q)$ *is the edge leading from* $p$ *to* $q$ *and labelled by* $(e, i)$*. The subgraph reachable from the node* $q_0$ *is called* Reachability Graph *of an APA.*

Please note that by this definition it is not allowed that transitions that can execute concurrently interact in a way that can not be achieved by executing them one after the other. Fortunately, however, this kind of interaction is not required for modelling software [Peled, 2001].

States in which no transition is enabled are called dead states. The derivation of RGs is called reachability analysis.

In the literature, a RG is sometimes also referred to as Transition System (TS) or Labeled Transition System (LTS) [Baier & Katoen, 2008; Peled, 2001]. Besides from APA, an LTS can also be generated from other models such as $\pi$-calculus [Milner, 1999; Sobocinski, 2007] or B [Bert & Cave, 2000]. In this thesis the term RG will be used to emphasise that the behaviour representation was generated from an APA and the term LTS will be used when the origin of the behaviour model does not matter in the given context.

A small example from P3 is now used to illustrate how APA can be used to specify a system and how the dynamic behaviour of the operational model representing the system is computed.

**Example 2.** *The problem: Given the puzzle in Figure 9 construct an operational model of the system that can be used to compute all possible positions* *reachable by shifting the numbers on the squares of the board to the empty square at any one time starting from the initial state shown in Figure 9 on the left and verify if the state on the right is reachable.*

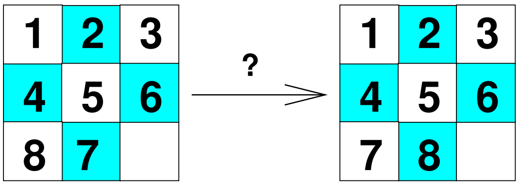


Figure 9: Is it possible to change positions of the *8* and *7*

*One solution is, (1) to represent the actual state of the puzzle by nine* state components *of an* APA *corresponding to the nine locations in the puzzle, (2) the board by twelve elementary automata - one for each dividing line between each two positions on the board -, and (3) to model the shifting of a numbered square between each two positions by a* state transition *of the respective elementary automaton.*

*For this example, the graphical* APA *representation (cf. Figure 8) is used. Only the syntactical elements used in the example are introduced here. For a complete description please refer to Fraunhofer SIT [2009].*

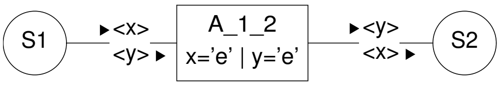


Figure 10: The elementary automaton A_1_2

*For the model of the puzzle example each elementary automaton has the form depicted in Figure 10. It shows an elementary automaton named* `A_1_2` *with two neighbour state components* `S1` *and* `S2`*. A state transition of the elementary automaton* `A_1_2` *may only change the content of directly connected state components* `S1` *and* `S2` *representing two neighbour positions on the puzzle board. In this example* `x` *is bound to the content of* `S1` *and* `y` *to the content of* `S2`*. The inscription* `x = 'e' | y = 'e'` *in the box represents a restriction for the possible transitions of* `A_1_2`*. If one of the state components contains the value* `'e'` *representing the empty square then the content of* `S1` *and* `S2` *can be interchanged (which represents a move on the board).*

*The* APA *for this example is given in Figure 11. Note that the squares with numbers are not part of the* APA *they just illustrate the initial state.*

*Formally, the model is given by the following specification.*

*APA state components:*

$\mathbb{S} = \{S1, S2, \ldots, S9\}$ *with*

$Z_{S1} = Z_{S2} = \ldots = Z_{S9} = \{1, 2, 3, 4, 5, 6, 7, 8, 'e'\}$ *and*

$q_0 = (q_{0S1}, q_{0S2}, \ldots, q_{0S9}) = (1, 2, 3, 4, 5, 6, 8, 7, 'e')$.

*Elementary automata:*

$\mathbb{E} = \{A\_1\_2, A\_2\_3, \ldots, A\_8\_9\}$.

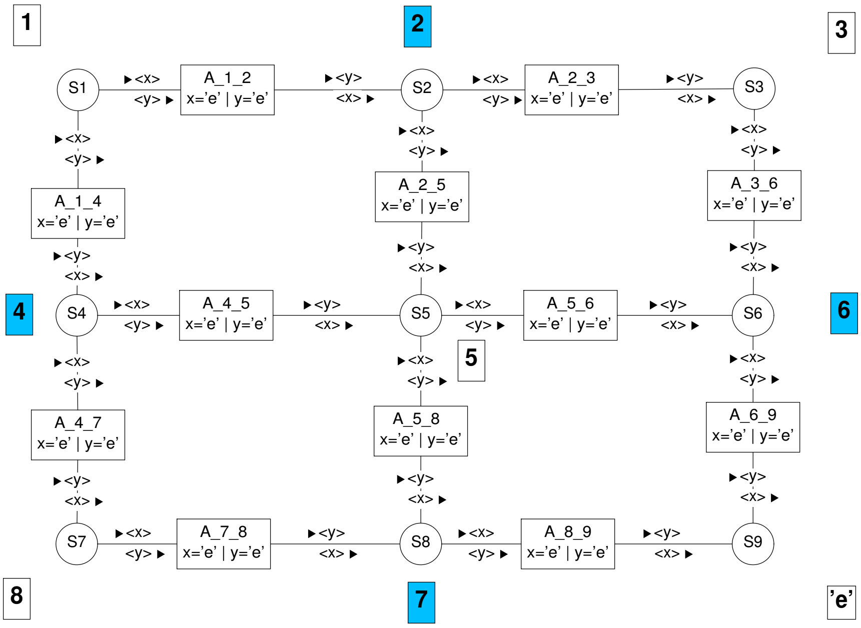


Figure 11: An APA model of the puzzle

*Neighbourhood relation:*
$N(A\_1\_2) = \{S1, S2\}, N(A\_2\_3) = \{S2, S3\}, \ldots, N(A\_8\_9) = \{S8, S9\}$.
*State transition relation:*
*Let* $Z = \{1, 2, 3, 4, 5, 6, 7, 8, 'e'\}$ *and for each* $e \in \mathbb{E}$ *let* $\Phi_e = \{\#\}$. *Therefore, the middle component of the state transition relation* $\Delta_e$ *can be omitted.*
*For each* $e \in \mathbb{E}$*:*
$\Delta_e = \{((x, y), \#, (y, x)) \in (Z \times Z) \times (Z \times Z) \mid ((x = 'e') \wedge (y \in Z)) \vee ((y = 'e') \wedge (x \in Z))\}$.

*Alternative behaviour is represented here by the* asynchronicity *of the possible transitions of the elementary automata that are neighbours to the empty square. For example, in the initial position the elementary automata A_6_9 and A_8_9 can act, that is,*

$$((1, 2, 3, 4, 5, 6, 8, 7, 'e'), (A\_6\_9, \#), (1, 2, 3, 4, 5, 'e', 8, 7, 6))$$

*and*

$$((1, 2, 3, 4, 5, 6, 8, 7, 'e'), (A\_8\_9, \#), (1, 2, 3, 4, 5, 6, 8, 'e', 7))$$

*are possible initial state transitions in the puzzle example. As mentioned above, # is omitted in the sequel. Both alternatives have to be evaluated.*

*The SHVT which can compute the possible behaviour of an APA has a built-in check for equal states. In Figure 12 the state following* M-2 *when shifting the square with content 7 back to the original position is identified with* M-1*. Please note that the tool prints the state* q0 *as* M-1*. The states following* M-3*,* M-4*, and* M-5 *have not been computed in Figure 12.*

*The brute force method to find out if the puzzle has the property asked for in Figure 9 is, to run a complete analysis of the example and then search in*

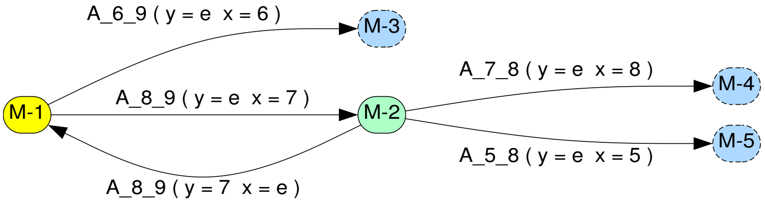


Figure 12: Initial part of the representation of the possible behaviour

*the generated states (in this case 181440 different states and 483840 state transitions) for a state matching the solution. However, in the given example the respective query will find no matching state because it is impossible to reach such a state by using the given operations.*

### 2.2.2 *Model checking*

In the context of CS, *verification* is the proof that system components work together in a desired manner. So the dynamic behaviour of the system has to be investigated. One usual approach is to start with a formal specification of the dynamic behaviour of the system which is represented by an LTS with an initial node and then to prove properties of such an LTS [Baeten & Weijland, 1990; Kurshan, 1994]. For finite state systems verification can be done by an exhaustive search of the state space given by the LTS. This search and check of properties on the states is called model checking. Model checking is preferable to deductive verification whenever applicable, because it can be performed automatically [Clarke et al., 1999].

Temporal Logic (TL) [Emerson, 1990] formulas are one way to represent the properties to be checked on some or all paths of the LTS. As a means to prove that components of CS securely work together (RQ1) a model checking algorithm for Propositional Linear Temporal Logic (PLTL) has been implemented by the author of this thesis. PLTL [Emerson, 1990] is the language used to specify *linear* properties that the system (characterised by its behaviour) is supposed to satisfy.

*A system satisfies a property* linearly *if and only if all its computations satisfy the property.*

The formulas are constructed as follows:

- 'True', 'False', and the edge-labels of the automaton representing the concrete or abstracted behaviour of the system to be checked are atomic TL formulas.
- If $f$ and $g$ are TL formulas, then $\neg f$, $f \wedge g$, $f \vee g$, $f \Rightarrow g$, $f \Leftrightarrow g$, **G**(always) $f$, **F**(eventually) $f$, $f$ **U**(until) $g$, $f$ **B**(before) $g$, and **X**(next) $f$ are TL formulas.

#### ALGORITHM TO CHECK PLTL SATISFACTION OF TL-FORMULAS

The concept of the algorithm for checking PLTL satisfaction of a TL formula is to find a counter-example, that is, to check if a model for the negation of the given TL formula exists.

STEP 1: First the given TL formula is negated and a Büchi automaton ***FA*** is constructed using the algorithm given in Gerth et al. [1996]. ***FA*** accepts infinite words corresponding to the property represented by the negated PLTL formula. The alphabet is given by the edge-labels of the automaton ***BA*** representing the concrete or abstracted behaviour of the system.

STEP 2: The synchronous product automaton ***SPA*** of the automaton ***BA*** (representing the system's behaviour) and the automaton ***FA*** (representing the negated property to be checked) is computed. Because ***BA*** has no acceptance conditions (all nodes accept) the synchronous product automaton ***SPA*** inherits the acceptance conditions of ***FA.***

STEP 3: A graph representation of the strongly connected components of ***SPA*** called the *component graph* of ***SPA*** (***CG-SPA***) is computed. Each node in this graph is a strongly connected component and there is an arc from one node to another, if there exists a transition to move from one strongly connected component to the other. ***CG-SPA*** is a graph without cycles. This approach allows to divide the original problem into subproblems, one for each strongly connected component.

STEP 4: Now for each component of the component graph ***CG-SPA*** it is checked if it satisfies the acceptance conditions (contains an accepting loop). If such a component is found this is a counter-example for the property to be verified an so the formula is ***false*** else it is ***true***. If the formula is *false* the part of the automaton ***BA*** that corresponds to the counter-example can be identified.

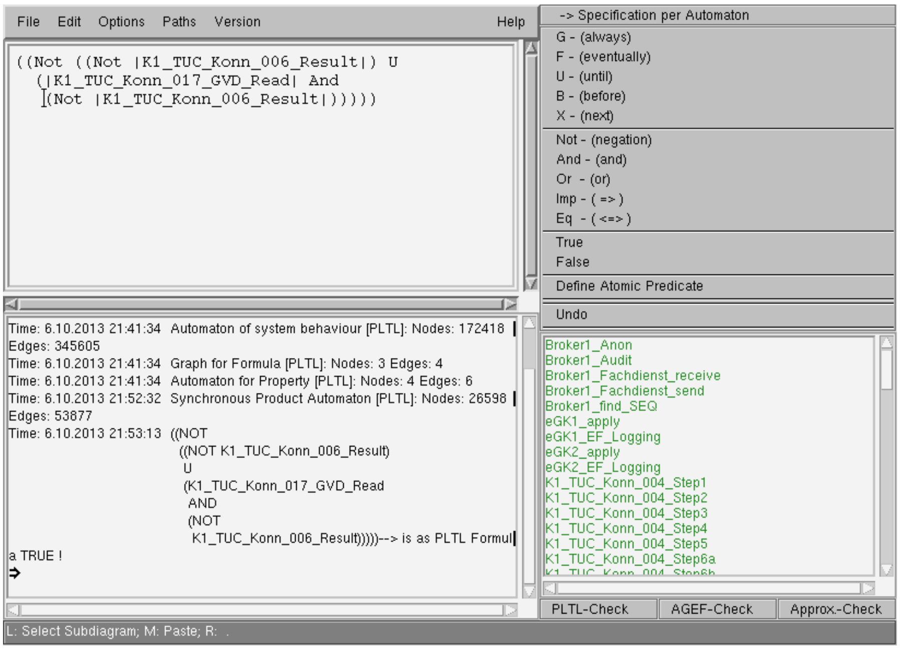
**Example 3.** *In the project ProOnline-VSDD an operational formal model of the communication infrastructure of the German eHealth Card was used to verify the following security requirement (log access before read).*

$$\begin{aligned}&\neg((\neg|K1\_TUC\_Konn\_006\_Result|)\ \mathbf{U}\\&\quad(|K1\_TUC\_Konn\_017\_GVD\_Read| \wedge\\&\quad(\neg|K1\_TUC\_Konn\_006\_Result|)))\end{aligned}$$

*Figure 13 shows the results of the verification of this requirement on an* LTS *with 172418 nodes and 345605 edges which took about 12 minutes (the computation of the reachability graph took about 22 minutes).*

### 2.2.3 *Abstraction based verification concept*

For real life applications the corresponding LTS are often too complex to apply the verification by model checking directly.



```
((Not ((Not |K1_TUC_Konn_006_Result|) U
  (|K1_TUC_Konn_017_GVD_Read| And
   (Not |K1_TUC_Konn_006_Result|)))))
```

```
Time: 6.10.2013 21:41:34  Automaton of system behaviour [PLTL]: Nodes: 172418
Edges: 345605
Time: 6.10.2013 21:41:34  Graph for Formula [PLTL]: Nodes: 3 Edges: 4
Time: 6.10.2013 21:41:34  Automaton for Property [PLTL]: Nodes: 4 Edges: 6
Time: 6.10.2013 21:52:32  Synchronous Product Automaton [PLTL]: Nodes: 26598
Edges: 53877
Time: 6.10.2013 21:53:13  ((NOT
                            ((NOT K1_TUC_Konn_006_Result)
                             U
                             (K1_TUC_Konn_017_GVD_Read
                              AND
                              (NOT
                               K1_TUC_Konn_006_Result))))--> is as PLTL Formul
a TRUE !
➔
```

Figure 13: TL-frame

In contrast to the immense number of transitions of such an LTS usually only a few *characteristic actions* of the system are of interest with respect to verification. So it is evident to define *abstractions with respect to the actions of interest* and to compute a representation of such an abstract behaviour, which usually is much smaller than the LTS of the specification.

Formally, the *behaviour* $L$ of a discrete system can be described by the set of its possible sequences of actions. Therefore, $L \subset \Sigma^*$ holds where $\Sigma$ is the set of all actions of the system, and $\Sigma^*$ (free monoid over $\Sigma$) is the set of all finite sequences of elements of $\Sigma$ (*words*), including the empty sequence denoted by $\varepsilon$. Subsets of $\Sigma^*$ are called *formal languages* [Sakarovitch, 2009]. Words can be concatenated: if $u$ and $v$ are words, then $uv$ is also a word; especially $\varepsilon u = u\varepsilon = u$. A word $u$ is called a *prefix* of a word $v$ if there is a word $x$ such that $v = ux$. The set of all prefixes of a word $u$ is denoted by $\text{pre}(u)$. Formal languages, which describe system behaviour, have the characteristic that $\text{pre}(u) \subset L$ holds for every word $u \in L$. Such languages are called *prefix closed*. System behaviour is thus described by *prefix closed formal languages*.

Different formal models of the same system are partially ordered with respect to different levels of abstraction. Abstraction is a fundamental and widely used verification technique. It can be used to reduce the verification of a property over a concrete system, to checking a related property over a simpler abstract system [Ochsenschläger, Repp & Rieke, 2000]. By these abstractions certain transitions are ignored and others are renamed, which may have the effect, that different transitions are identified with one another. Generally, under abstraction the problem occurs that an incorrect subbehaviour can be hidden by a correct one. However, in Ochsenschläger, Repp & Rieke [2000] it has been shown that when the abstraction complies to a specific restriction then by investigating an abstract behaviour, the correctness of the underlying concrete behaviour can be inferred.

**Definition 4** (abstractions)**.** Abstractions *are described by* alphabetic language homomorphisms*. These are mappings* $h^* : \Sigma^* \longrightarrow \Sigma'^*$ *with*

$$h^*(xy) = h^*(x)h^*(y), h^*(\varepsilon) = \varepsilon \textit{ and } h^*(\Sigma) \subset \Sigma' \cup \{\varepsilon\}.$$

*So they are uniquely defined by corresponding mappings* $h : \Sigma \longrightarrow \Sigma' \cup \{\varepsilon\}$*. In the following, both the mapping* $h$ *and the homomorphism* $h^*$ *is denoted by* $h$*. In general, let* $\check{L} \subset \check{\Sigma}^*$ *and* $L \subset \Sigma^*$ *be prefix closed languages.* $\check{L}$ *is called* finer than $L$ *and* $L$ *is called* coarser than $\check{L}$ *iff an alphabetic homomorphism* $\nu : \check{\Sigma}^* \to \Sigma^*$ *exists with* $\nu(\check{L}) = L$*.*

As it is well known, system properties are divided into two types: *safety* (what happens is not wrong) and *liveness* properties (eventually something desired happens, e.g. availability) [Alpern & Schneider, 1985]. On account of liveness aspects system properties are formalised by $\omega$-languages (sets of infinite long words) [Perrin & Pin, 2004]. Thus, to investigate satisfaction of properties, infinite system behaviour has to be considered. The usual concept of linear satisfaction of properties (each infinite run of the system satisfies the property) is not suitable in this context because no fairness constraints are considered. Therefore, an abstract notion of fairness in the satisfaction relation for properties is needed, which considers that independent of a finitely long computation of a system certain desired events may occur eventually. To formalise such "possibility properties", which are of interest when considering CS, the notion of approximate satisfaction of properties has been defined in Nitsche & Ochsenschläger [1996].

**Definition 5.** *A system* approximately satisfies *a property if and only if each finite behaviour can be continued to an infinite behaviour, which satisfies the property.*

For safety properties linear satisfaction and approximate satisfaction are equivalent [Nitsche & Ochsenschläger, 1996]. To deduce approximately satisfied properties of a specification from properties of its abstract behaviour an additional property of abstractions called *simplicity of homomorphisms* on an action language [Ochsenschläger, 1994] is required. Simplicity of homomorphisms is a very technical condition concerning the possible continuations of finite behaviours. For regular languages simplicity is decidable. In [Ochsenschläger, 1994] a sufficient condition based on the strongly connected components of corresponding automata is given, which easily can be checked. Especially: If the automaton or reachability graph is strongly connected, then each homomorphism is simple.

The following theorem from Nitsche & Ochsenschläger [1996] shows that approximate satisfaction of properties and simplicity of homomorphisms exactly fit together for verifying CS.

**Theorem 1.** *Simple homomorphisms define exactly the class of such abstractions, for which holds that each property is approximately satisfied by the abstract behaviour if and only if the "corresponding" property is approximately satisfied by the concrete behaviour of the system [Nitsche & Ochsenschläger, 1996].*

Formally, the "corresponding" property is expressed by the inverse image of the abstract property with respect to an appropriate modification of the abstraction for ω-languages [Nitsche & Ochsenschläger, 1996; Ochsenschläger & Rieke, 2012b].

Figure 14 shows an overview of this approach.

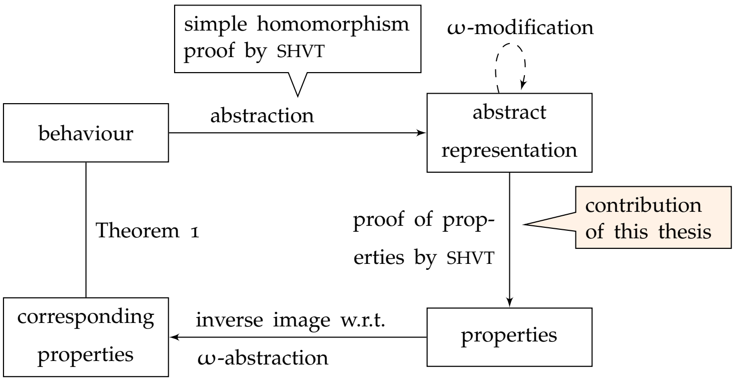


Figure 14: Abstraction based verification approach

The following algorithm from Nitsche [1998] checks whether a property is satisfied approximately by the "behaviour-automaton". On the automaton level, seven steps have to be performed (cf. P1):

Step 1: Compute a Büchi automaton representing the property given by a PLTL formula according to Gerth et al. [1996].

Step 2: Construct the synchronous product of the automaton constructed so far and the automaton representing the behaviour of the considered system. The synchronous product is the construction of the intersection of languages on the automaton level.

Step 3: Reduce the resulting Büchi automaton (remove all states which are not reachable from the initial state or from which no cycle containing an accepting state is reachable).

Step 4: Ignore acceptance conditions (make all states accepting) and do not interpret the automaton anymore as an automaton on infinite (ω-) words but as one on finitely long words (this corresponds to the prefix construction).

Step 5: Construct the complement automaton (for a finite-word automaton, not an $\omega$-automaton).

Step 6: Construct the intersection with the automaton representing the behaviour.

Step 7: Check whether the resulting automaton is empty (does not accept words).

This algorithm, a user interface, and visualisation methods for intermediate graphs and failure traceback have been implemented by the author of this thesis in the SHVT that is introduced now.

### 2.2.4 *Simple homomorphism verification tool*

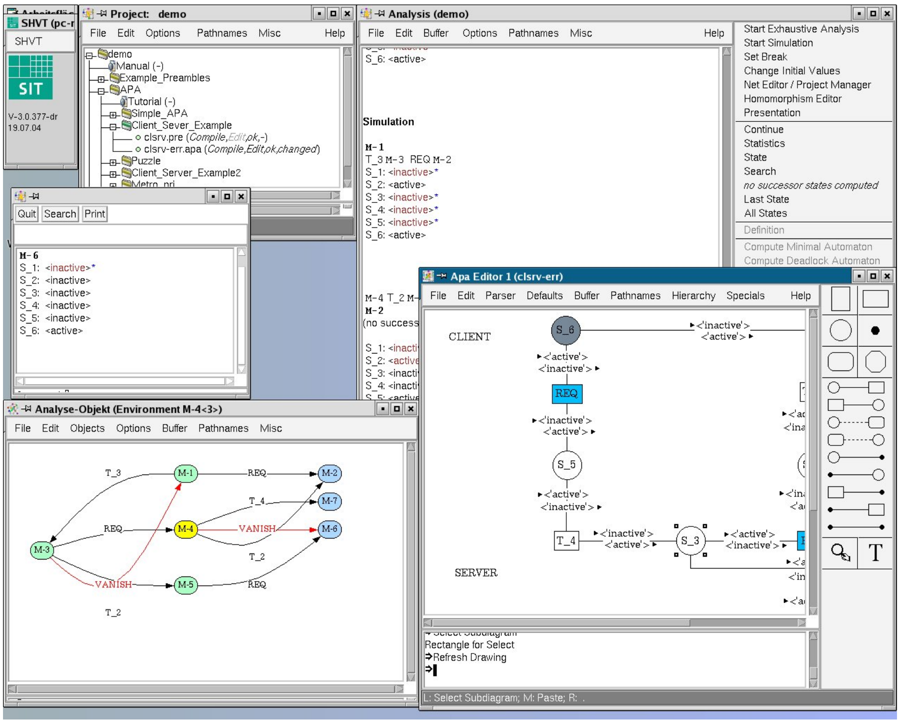
The SHVT has been developed at the *Fraunhofer-Institute for Secure Information Technology* with significant participation of the author of this thesis in the design and implementation of the tool. The SHVT provides components for the complete cycle from formal specification to exhaustive validation as well as visualisation and inspection of computed reachability graphs and minimal automata. The applied specification method based on APA is supported by this tool. The tool manages the components of the model, allows to select alternative parts of the specification and automatically *glues* together the selected components to generate a combined model of the APA specification. After an initial state is selected, the RG is automatically computed by the SHVT. The tool provides an editor to define homomorphisms on action languages, it computes corresponding minimal automata [Eilenberg, 1974] for the homomorphic images and checks simplicity of the homomorphisms.

The SHVT successfully has been applied in several security projects such as Valikrypt [Bundesamt für Sicherheit in der Informationstechnik, 2003], Computer-Aided solutions to SEcure electroNic commercE Transactions (CASENET) [NetUnion, 2003], MakoSi [Herfert et al., 2004], and SicAri [Peters, 2013; Rieke & Ebinger, 2008]. The user interface (cf. Figure 15) and general handling of the SHVT has reached a level of maturity that enabled its successful application in the industrial area [Apel et al., 2007].

P1 gives an overview about the main functions of the SHVT, P2 gives an overview about the main components of the tool, and P3 provides an extensive example for the use of the methods and tool described in P1 and P2.

## 2.3 SCALABLE VERIFICATION OF PROPERTIES

Traditional model checking techniques allow a verification of the required behaviour only for systems with very few components. For example, in Wang et al. [2000] "E-process Design and Assurance Using Model Checking" has been analysed. Their application scenario


SHVT
SIT
V-3.0.377-dr
19.07.04
Project: demo
File Edit Options Pathnames Misc Help
demo
Manual (-)
Example_Preambles
APA
Tutorial (-)
Simple_APA
Client_Sever_Example
clsrv.pre (Compile,Edit,pk,-)
clsrv-err.apa (Compile,Edit,ok,changed)
Puzzle
Client_Server_Example2
Analysis (demo)
File Edit Buffer Options Pathnames Misc Help
Start Exhaustive Analysis
Start Simulation
Set Break
Change Initial Values
Net Editor / Project Manager
Homomorphism Editor
Presentation
Continue
Statistics
State
Search
no successor states computed
Last State
All States
Definition
Compute Minimal Automaton
Compute Deadlock Automaton
Quit Search Print
Analyse-Objekt (Environment M-4<3>)
File Edit Objects Options Buffer Pathnames Misc
M-1
M-2
M-3
M-4
M-5
M-6
M-7
T_3
T_4
T_2
REQ
VANISH
Apa Editor 1 (clsrv-err)
File Edit Parser Defaults Buffer Pathnames Hierarchy Specials Help
CLIENT
SERVER
S_6
REQ
S_5
T_4
S_3
<'active'>
<'inactive'>
T

```
S_6: <active>

Simulation

M-1
T_3 M-3 REQ M-2
S_1: <inactive>*
S_2: <active>
S_3: <inactive>*
S_4: <inactive>*
S_5: <inactive>*
S_6: <active>
```

```
M-6
S_1: <inactive>*
S_2: <inactive>
S_3: <inactive>
S_4: <inactive>
S_5: <inactive>
S_6: <active>
```

```
Rectangle for Select
Refresh Drawing
```

```
L: Select Subdiagram; M: Paste; R: .
```

Figure 15: Simple Homomorphism Verification Tool

was a ticket sales example. Because they wanted to compute the complete system behaviour they had to make some simplifications to their model. With only 2 agents and 2 customers, a ticket server that has only 1 ticket for sale and other simplifications they still computed 331079 system states using automatic model checking based on the tools VeriSoft [Godefroid, 1997] and Spin [Ben-Ari, 2008].

Similarly, a security analysis of the German Health Card infrastructure [Stroetmann & Lilischkis, 2007; gematik, 2007b] and services in particular the management services for the insurance master data [gematik, 2007a] which was done by the author of this thesis [Rieke, 2009a,b] showed the limits of explicit finite state model checking methods (state space explosion).

This general state space explosion problem leads to the following question.

RQ2: *How can finite state verification techniques be extended to prove properties independently of concrete parameters?*

To address this research question, an approach has been developed that extends the techniques described in Section 2.2 to a particularly interesting class of systems called *parameterised systems*. A parameterised system describes a family of systems that are finite-state in nature but scalable. A formal specification of a parameterised system thus covers a family of systems, each member of which has a different number of replicated components. Instances of the family can be obtained by fixing the parameters. Extensions of model checking techniques are required that support verification of properties that are valid independently of given concrete parameters.

To be able to verify entire families of critical systems, independent of the exact number of replicated components, an *abstraction based approach* has been developed to extend current tool supported verification techniques to such parameterised systems.

To address the research question RQ2 using this abstraction based approach an inductive proof on the construction of the behaviour of the parameterised system is needed to show that it results in identical abstract system behaviour for any given parameter configuration. This allows the verification of parameterised systems by constructing abstract systems that can be model checked. This is demonstrated in P4.

In the case of the proposed abstraction based approach, the key problem is the choice of an appropriate abstraction that, firstly, is *property preserving*, secondly, results in identical abstract system behaviour for any given parameter configuration, and, lastly, is sufficiently precise to express the required properties at the chosen abstraction level. To solve this problem, it is suggested

- to compute the system behaviour and verify the required properties for some configurations with fixed numbers of components;
- to use the results to choose an appropriate property preserving abstraction that results in identical abstract system behaviour for any given parameter configuration;
- to provide an inductive proof (by hand) based on this abstraction that generalises the results for a family of systems with arbitrary settings of parameters.

An outline of the verification concept for parameterised models that has been published in P4 is depicted in Figure 16.

In the proposed approach for each individual verification the following proofs are needed:

**SIMPLE HOMOMORPHISM PROOF.** The computation of abstraction and proof that the homomorphism is simple for small parameters. This serves as base for the induction. The proof is done automatically - for several sufficient conditions - by the built-in algorithms of the SHVT.

**PROOF OF PROPERTIES.** The specified properties have to be proven for the abstract behaviour, for example, by the TL model check in the SHVT (cf. Section 2.2.2).

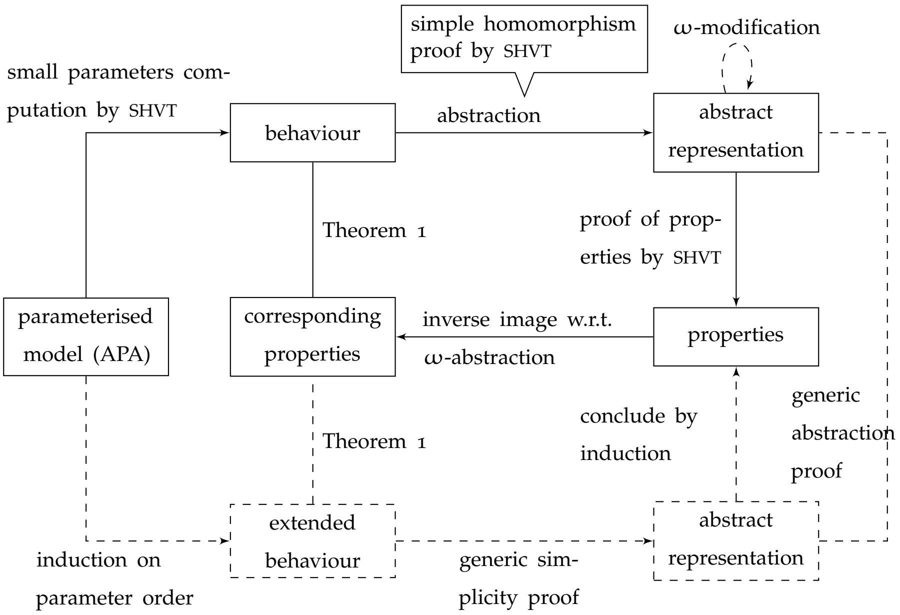

simple homomorphism proof by SHVT
ω-modification
small parameters computation by SHVT
behaviour
abstraction
abstract representation
Theorem 1
proof of properties by SHVT
parameterised model (APA)
corresponding properties
inverse image w.r.t. ω-abstraction
properties
generic abstraction proof
conclude by induction
Theorem 1
extended behaviour
abstract representation
induction on parameter order
generic simplicity proof

Figure 16: Verification concept for parameterised APA

**GENERIC SIMPLICITY PROOF.** It has to be proven that the homomorphism is simple for each parameter setting. This can be done, for example, by proving that the LTS representing the abstract behaviour is finite and strongly connected.

**GENERIC ABSTRACTION PROOF.** It has to be proven that the application of the chosen homomorphism results in identical abstract system behaviour for any given parameter configuration. This can be done, for example, by an inductive proof on the construction of the behaviour of the parameterised system.

In P4 this approach is demonstrated by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario.

## 2.4 SECURITY REQUIREMENTS ELICITATION

This section addresses the security requirements engineering process for CS. CS typically base decisions on information from their own components as well as on input from other systems. Safety critical decisions based on cooperative reasoning, such as automatic emergency braking of a vehicle, raise severe concerns to security issues. Thus, security requirements need to be explicit, precise, adequate, non-conflicting, and complete [van Lamsweerde, 2004]. Therefore, the following question has to be answered.

RQ3: *How can security requirements for cooperating systems be elicited systematically?*

In P5, an approach has been published that addresses this research question. In particular, it presents a systematic and constructive approach to the authenticity requirements elicitation step in this process. This approach makes use of the Security Modelling Framework (SeMF) that has been introduced in Gürgens et al. [2002]. In this framework, requirements are defined by specific constraints regarding sequences of actions that can or can not occur in a system's behaviour. Actions in SeMF represent an abstract view on actions of the real system. SeMF models the *interdependencies* between actions and ignores their functionality. An action is specified in a parameterised format, consisting of the action's name, the acting agent and a variable set of parameters:

$$\mathit{actionName}(\mathit{actingAgent}, \mathit{parameter1}, \mathit{parameter2}, \ldots)$$

Specifically, *authenticity* can be seen as the assurance that a particular action has occurred in the past. For a formal specification of the application-level authenticity requirements, Definition 6, which is taken from Gürgens et al. [2002], is used.

**Definition 6.** *auth*$(a, b, P)$*: Whenever an action* $b$ *happens, it must be authentic for an Agent* $P$ *that in any course of events that seem possible to him, a certain action* $a$ *has happened.*

An important aspect of a systematic security evaluation is the analysis of potential information flows. The method published in P5 derives such authenticity requirements from an information flow model of the involved CS based on functional dependency analysis. From the use case descriptions, *atomic actions* are derived and set into relation by defining the functional flow among them. The action-oriented approach considers possible sequences of actions (control flow) and information flow (input/output) between interdependent actions. Actions of interest are specifically the *boundary actions*, which represent the interaction of the system's internals with the outside world. From a functional dependency graph, the boundary actions can be identified. The analysis now spans a dependency graph with a safety critical action as root and the origins of decision relevant information as leaves. Based on this graph, a set of authenticity requirements is deduced for the input from the leaves of the derivation graph. This set is comprehensive and defines the maximal set of authenticity requirements from the given functional dependencies. Furthermore, the proposed method avoids premature assumptions on the architectural structure and mechanisms to implement security measures.

This approach is now illustrated by an example taken from P5 that is based on a security relevant use case from the project EVITA [Fraunhofer SIT, 2011] in which vehicles and roadside units communicate in an ad hoc manner to exchange information such as safety warnings and traffic information [Ruddle et al., 2009].

**Example 4** (Boundary Actions and Dependencies)**.** *The CS in this example consists of a vehicle w and a RoadSide Unit (RSU) that can send cooperative awareness messages* $\mathtt{cam}$*. The vehicle is equipped with an Electronic Stability Protection (ESP) sensor, a Global Positioning System (GPS) sensor, and a Communication Unit (CU). Furthermore, a connection to a Human Machine Interface (HMI) is required to display the warning message.*

*In the CS instance depicted in Figure 17 the authenticity requirements for the boundary action show*$(HMI_w, \mathrm{warn})$ *shall be identified. Following backwards along the functional flow it can be derived that the output action show*$(HMI_w, \mathrm{warn})$ *is depending on the input actions pos*$(GPS_w, \mathrm{pos})$ *of vehicle w and send*$(\mathrm{cam}(\mathrm{pos}))$ *of the RSU.*

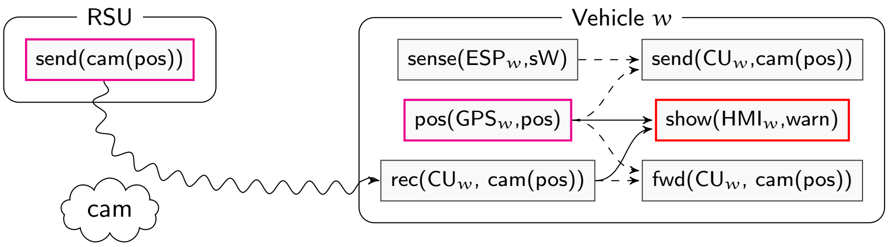


Figure 17: Vehicle $w$ receives warning from RSU

In P6, this model-based approach has been applied by the author of this thesis to systematically identify security requirements for a critical infrastructure. In critical infrastructures some information sources – like sensors that provide the event data - are typically placed in a non-protected environment at the boarder of the managed system. Thus, they are exposed to various kinds of attacks. Manipulated equipment can be used to hide critical conditions, generate false alerts, and in general cause misjudgement on subsystem's state. Wrong assumptions about a subsystem's state in turn can lead to false decisions with severe impact on the overall CS. As a consequence, the system has to assure that all safety critical actions using sensor data must only use authentic sensor data.

**Proposition 1.** *Whenever a certain control decision is made, the input information that presumably led to it must be authentic.*

The question, which measurements and system control decisions are critical to the overall system behaviour, cannot be answered independently of the concrete system and application context determined. The following example taken from P6 analyses a possible misuse case based on a scenario of the MASSIF project [Llanes et al., 2011].

**Example 5** (Water level sensor compromise)**.** *An attacker takes control of the Water Level Sensors (WLS) of a dam and uses them to send spoofed measurements of the water level (wl) to the dam control station (DCS). This hides the real status of the reservoir to the dam administrator (Admin). In* *this way, the dam can be overflown without alarms being raised by the monitoring system. From this, it is obvious that the water level measures have to be authentic for the administrator when they are displayed at the dam control station. More formally, the authenticity requirement can be specified as follows:*

$$auth(sense(WLS, wl), display(DCS, wl), Admin) \tag{1}$$

Example 5 shows that some elementary security requirements can be derived directly from misuse cases. In general, however, information flows between systems and components are highly complex, especially when organisational processes need to be considered. Hence, not all security problems are discoverable easily. In order to achieve the desired security goals, security requirements need to be derived systematically.

The following example describes authenticity requirements elicitation based on security information flows in the use case of the dam scenario [Llanes et al., 2011] that has been published in P6.

**Example 6** (On demand electric production)**.** *The dam control station feeds an hydroelectric turbine, connected to the dam by means of penstocks, for producing electric power on demand. The turbine and hydroelectric power production depends on the water discharge in the penstocks. By analysing the parameters of the command received by the dam control station, it can be inferred that the safety critical actions are the opening and closing actions of the penstock gates* (PG)*. Table 4 lists the dam scenario actions involved in the security requirements analysis.*

Table 4: Dam actions

| **Action** | **Description** |
|---|---|
| $sense(WLS, wl)$ | Measurement of the water level. |
| $sense(PP, power)$ | Measurement of voltage and current in the power grid. The power plant *PP* sends commands *ppc* to the dam control station depending on these measurements. |
| $sense(SDC, wdc)$ | Measurement of the water discharge on the penstock gates *PG*. |
| $sense(PG, open)$ | Reporting of the state of the penstock gates. |
| $display(DCS, X)$ | Display X at the dam control station, with $X \in \{wl, ppc, wdc, open\}$. |
| $activate(Admin, cmd)$ | Decision of the administrator, which command shall be triggered. |
| $exec(PG, cmd)$ | Command to be executed by penstock gates. |

*An identification of functional dependencies reveals that the dam control activity makes use of the (i) current water level, (ii) the state of the gates joined to the hydroelectric power plant, (iii) the gates openness, and, (iv)* *the discharge through the penstocks. Figure 18 shows the dependency graph of this use case. The decision of the administrator, which command shall be triggered, depends on the displayed measurements. The dashed line indicates that there is no direct functional dependency.*

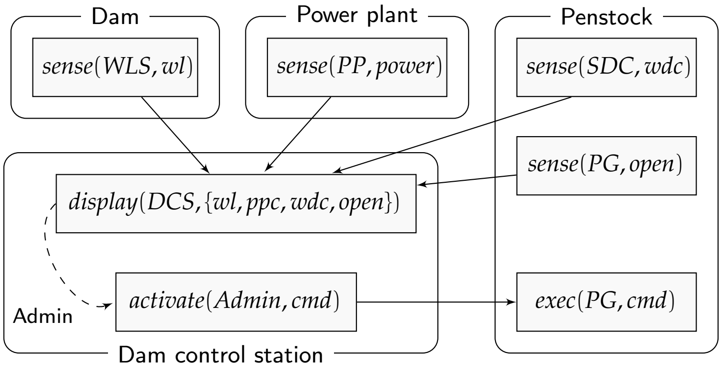


Figure 18: Functional dependencies: *On demand electric production*

*An analysis of the dependencies depicted in Figure 18 leads to the following conclusion: The control display values are derived from the measurements of* wl, power, wdc, *and* open.

*From this, it can be concluded that, in addition to the water level* wl (1), *the measurements of* power, wdc, *and* open *have to be authentic. More formally:*

$$auth(\,sense(PP, power), display(DCS, ppc), Admin) \tag{2}$$

$$auth(\,sense(SDC, wdc), display(DCS, wdc), Admin) \tag{3}$$

$$auth(\,sense(PG, open), display(DCS, open), Admin) \tag{4}$$

*Furthermore, the activation of the penstock command by the administrator has to be authentic for the penstock gate when executing it.*

$$auth(activate(Admin, cmd), exec(PG, cmd), PG) \tag{5}$$

*So the authenticity requirements for the use case described in Figure 18 are given by:* (1), (2), (3), (4), *and* (5).

It is evident that further types of security requirements are needed in order to cover important liveness properties such as *availability* of necessary information at a certain place and time. In some cases also *confidentiality* of certain information may be required. These requirements are important but not in the scope of the work developed in this thesis.

## 2.5 SCALABILITY FOR LARGE-SCALE

Scalability is a desirable property of a system. In Bondi [2000], four aspects of scalability are considered, i.e., load scalability, space scalability, space-time scalability, and structural scalability. In this thesis, the focus is on *structural scalability*, which is “the ability of a system to expand in a chosen dimension without major modifications to its architecture” [Bondi, 2000]. Examples of systems that need to be highly scalable comprise grid computing architectures and cloud computing platforms [Bullock & Cliff, 2004; Weinman, 2011] but also vehicular ad hoc networks [Gerlach, 2005] or large scale client-server architectures like the German electronic health card (eGK) telematics infrastructure [Stroetmann & Lilischkis, 2007; gematik, 2007b]. It has been shown in Section 2.3 that the complexity of such SoS causes problems with verification of security properties. This leads to the following research question.

RQ4: *Which design principles facilitate verifiability of security properties of scalable systems?*

Usually, scalable systems consist of few different types of components and for each such type a varying set of individual components exists. Component types can be defined in such a granularity that individual components of the same type behave in the same manner, which is characteristic for the type. For example, a client-server system that is scalable consists of the component types *client* and *server* and several sets of individual clients as well as several sets of individual servers. Thus, in P7 and P8 *uniform parameterisations* of cooperations are defined that comprise this important class of scalable systems.

These *uniformly parameterised cooperations* are characterised by (i) the composition of a set of identical components (copies of a two-sided cooperation); and (ii) the fact that these components interact in a uniform manner (described by the schedules of the partners). E-commerce protocols, for example, are instances of such uniformly parameterised systems of cooperations, in which the cooperation partners have to perform specific financial transactions. An E-commerce protocol should work for several partners in the same manner, and the mechanism (schedule) to determine how one partner may be involved in several cooperations is the same for each partner. So, the cooperation is parameterised by the partners and the parameterisation should be uniform with respect to the partners.

### 2.5.1 *Parameterised cooperations*

As already introduced in Section 2.2, in this thesis the focus is on the dynamic behaviour of systems, which is described by the set of all possible sequences of actions. This point of view is important to define security requirements as well as to verify such properties, because for these purposes sequences of actions of the system have to be considered [Schneider, 1996; Zegzhda et al., 2012].

To describe a two-sided cooperation, let $\Sigma = \Phi \cup \Gamma$ where $\Phi$ is the set of actions of cooperation partner F and $\Gamma$ is the set of actions of cooperation partner G and $\Phi \cap \Gamma = \emptyset$. Now a prefix closed language $L \subset (\Phi \cup \Gamma)^*$ formally defines a two-sided cooperation.

**Example 7.** *Let* $\Phi = \{f_s, f_r\}$, $\Gamma = \{g_r, g_i, g_s\}$ *and* $\Sigma = \{f_s, f_r, g_r, g_i, g_s\}$. *An example for a cooperation* $L \subset \Sigma^*$ *is now given by the automaton in Figure 19. It describes a simple handshake between* F *(client) and* G *(server), where a client may perform the actions* $f_s$ *(send a request),* $f_r$ *(receive a result) and a server may perform the corresponding actions* $g_r$ *(receive a request),* $g_i$ *(internal action to compute the result) and* $g_s$ *(send the result).*

In the following, initial states will be denoted by a short incoming arrow and final states by double circles. In the automaton depicted in Figure 19 all states are final states, since L is prefix closed.

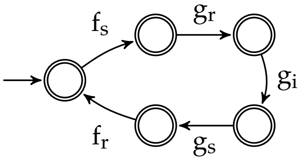


Figure 19: Automaton for 1-1-cooperation L

For parameter sets I, K and $(i,k) \in I \times K$ let $\Sigma_{ik}$ denote pairwise disjoint copies of $\Sigma$. The elements of $\Sigma_{ik}$ are denoted by $a_{ik}$ and $\Sigma_{IK} := \bigcup_{(i,k) \in I \times K} \Sigma_{ik}$. The index ik describes the bijection $a \leftrightarrow a_{ik}$ for $a \in \Sigma$ and $a_{ik} \in \Sigma_{ik}$. Now $\mathcal{L}_{IK} \subset \Sigma_{IK}^*$ (prefix-closed) describes a *parameterised system*. To avoid pathological cases, it is generally assumed that parameter and index sets to be non empty.

For a cooperation between one partner of type F with two partners of type G in Example 7 let

$$\begin{aligned}\Phi_{\{1\}\{1,2\}} &= \{f_{s11}, f_{r11}, f_{s12}, f_{r12}\},\\ \Gamma_{\{1\}\{1,2\}} &= \{g_{r11}, g_{i11}, g_{s11}, g_{r12}, g_{i12}, g_{s12}\} \text{ and}\\ \Sigma_{\{1\}\{1,2\}} &= \Phi_{\{1\}\{1,2\}} \cup \Gamma_{\{1\}\{1,2\}}.\end{aligned}$$

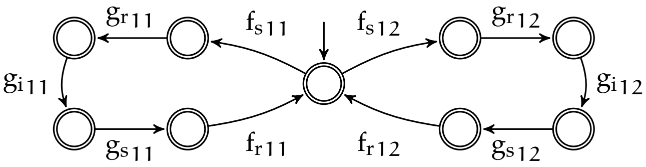


Figure 20: Automaton for 1-2-cooperation $\mathcal{L}_{\{1\}\{1,2\}}$

A 1-2-cooperation, where each pair of partners cooperates restricted by L and each partner has to finish the handshake it just is involved in before entering a new one, is now given (by reachability analysis) by the automaton in Figure 20 for $\mathcal{L}_{\{1\}\{1,2\}}$. It shows that one after another client 1 runs a handshake either with server 1 or with server 2. Figure 21 depicts an automaton for a 2-1-cooperation $\mathcal{L}_{\{1,2\}\{1\}}$ with the same overall number of partners involved but two of type F and one partner of type G. Figure 21 is more complex than Figure 20 because client 1 and client 2 may start a handshake independently of each other, but server 1 handles these handshakes one after another. For a 5-3-cooperation with the same simple behaviour of partners the SHVT already computes 194.677 states and 1.031.835 state transitions.

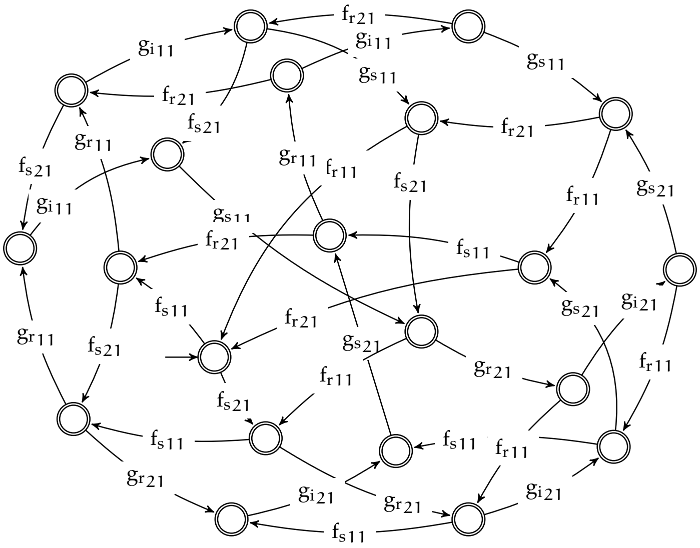


Figure 21: Automaton for the 2-1-cooperation $\mathcal{L}_{\{1,2\}\{1\}}$

For $(i,k) \in I \times K$, let $\pi^{IK}_{ik} : \Sigma^*_{IK} \to \Sigma^*$ with

$$\pi^{IK}_{ik}(a_{rs}) = \begin{cases} a \mid & a_{rs} \in \Sigma_{ik} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Sigma_{ik} \end{cases} .$$

For *uniformly parameterised systems* $\mathcal{L}_{IK}$ one generally wants to have

$$\mathcal{L}_{IK} \subset \bigcap_{(i,k) \in I \times K} ((\pi^{IK}_{ik})^{-1}(L))$$

because from an abstraction point of view, where only the actions of a specific $\Sigma_{ik}$ are considered, the complex system $\mathcal{L}_{IK}$ is restricted by L.

In addition to this inclusion, $\mathcal{L}_{IK}$ is defined by *local schedules* that determine how each "version of a partner" can participate in different cooperations. More precisely, let $SF \subset \Phi^*$, $SG \subset \Gamma^*$ be prefix closed. For $(i,k) \in I \times K$, let $\varphi_i^{IK} : \Sigma_{IK}^* \to \Phi^*$ and $\gamma_k^{IK} : \Sigma_{IK}^* \to \Gamma^*$ with

$$\varphi_i^{IK}(a_{rs}) = \begin{cases} a \mid & a_{rs} \in \Phi_{\{i\}K} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Phi_{\{i\}K} \end{cases} \quad \text{and}$$

$$\gamma_k^{IK}(a_{rs}) = \begin{cases} a \mid & a_{rs} \in \Gamma_{I\{k\}} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Gamma_{I\{k\}} \end{cases},$$

where $\Phi_{IK}$ and $\Gamma_{IK}$ are defined correspondingly to $\Sigma_{IK}$.

**Definition 7** (uniformly parameterised cooperation).
*Let* $I$, $K$ *be finite parameter sets, then*

$$\mathcal{L}_{IK} := \bigcap_{(i,k) \in I \times K} (\pi_{ik}^{IK})^{-1}(L) \cap \bigcap_{i \in I} (\varphi_i^{IK})^{-1}(SF) \cap \bigcap_{k \in K} (\gamma_k^{IK})^{-1}(SG)$$

*denotes a* uniformly parameterised cooperation.

By this definition,

$$\mathcal{L}_{\{1\}\{1\}} = (\pi_{11}^{\{1\}\{1\}})^{-1}(L) \cap (\varphi_1^{\{1\}\{1\}})^{-1}(SF) \cap (\gamma_1^{\{1\}\{1\}})^{-1}(SG).$$

In order to have $\mathcal{L}_{\{1\}\{1\}}$ be isomorphic to $L$ by the isomorphism $\pi_{11}^{\{1\}\{1\}} : \Sigma_{\{1\}\{1\}}^* \to \Sigma^*$, it is additionally required that

$$\begin{aligned} (\pi_{11}^{\{1\}\{1\}})^{-1}(L) &\subset (\varphi_1^{\{1\}\{1\}})^{-1}(SF) \text{ and} \\ (\pi_{11}^{\{1\}\{1\}})^{-1}(L) &\subset (\gamma_1^{\{1\}\{1\}})^{-1}(SG). \end{aligned}$$

This is equivalent to $\pi_\Phi(L) \subset SF$ and $\pi_\Gamma(L) \subset SG$, where $\pi_\Phi : \Sigma^* \to \Phi^*$ and $\pi_\Gamma : \Sigma^* \to \Gamma^*$ are defined by

$$\pi_\Phi(a) = \begin{cases} a \mid & a \in \Phi \\ \varepsilon \mid & a \in \Gamma \end{cases} \text{ and } \pi_\Gamma(a) = \begin{cases} a \mid & a \in \Gamma \\ \varepsilon \mid & a \in \Phi \end{cases}.$$

So, Definition 7 is completed by the additional conditions

$$\pi_\Phi(L) \subset SF \text{ and } \pi_\Gamma(L) \subset SG.$$

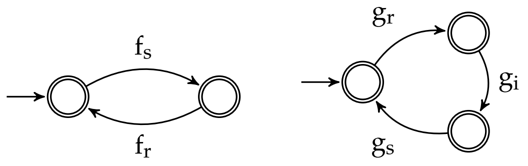
Figure 22a and Figure 22b depict schedules $SF$ and $SG$ that fit to the cooperations given in Example 7. Here, $\pi_\Phi(L) = SF$ and $\pi_\Gamma(L) = SG$.

The system $\mathcal{L}_{IK}$ of cooperations is a typical example of a *complex system*. It consists of several identical components (copies of the two-sided cooperation $L$), which interact in a uniform manner (described by the schedules $SF$ and $SG$ and by the homomorphisms $\varphi_i^{IK}$ and $\gamma_k^{IK}$).

**Remark 1.** *It is easy to see that* $\mathcal{L}_{IK}$ *is isomorphic to* $\mathcal{L}_{I'K'}$ *if* $I$ *is isomorphic to* $I'$ *and* $K$ *is isomorphic to* $K'$. *More precisely, let* $\iota_{I'}^{I} : I \to I'$ *and* $\iota_{K'}^{K} : K \to K'$ *be bijections and let* $\iota_{I'K'}^{IK} : \Sigma_{IK}^* \to \Sigma_{I'K'}^*$ *be defined by*

$$\iota_{I'K'}^{IK}(a_{ik}) := a_{\iota_{I'}^{I}(i)\iota_{K'}^{K}(k)} \textit{ for } a_{ik} \in \Sigma_{IK}.$$

*Hence,* $\iota_{I'K'}^{IK}$ *is a isomorphism and* $\iota_{I'K'}^{IK}(\mathcal{L}_{IK}) = \mathcal{L}_{I'K'}$. *The set of all these isomorphisms* $\iota_{I'K'}^{IK}$ *defined by corresponding bijections* $\iota_{I'}^{I}$ *and* $\iota_{K'}^{K}$ *is denoted by* $\mathcal{I}_{I'K'}^{IK}$.



(a) Schedule *SF*

(b) Schedule *SG*

Figure 22: Automata $\mathbb{SF}$ and $\mathbb{SG}$ for the schedules *SF* and *SG*

### 2.5.2 *Self-similarity*

The notion of *self-similarity* is introduced in order to formalise that for $I' \subset I$ and $K' \subset K$ from an abstracting point of view, where only the actions of $\Sigma_{I'K'}$ are considered, the complex system $\mathcal{L}_{IK}$ behaves like the smaller subsystem $\mathcal{L}_{I'K'}$. Therefore, now special abstractions on $\mathcal{L}_{IK}$ are considered.

**Definition 8** (Projection abstraction)**.**
*For* $I' \subset I$ *and* $K' \subset K$ *let* $\Pi^{IK}_{I'K'} : \Sigma^*_{IK} \to \Sigma^*_{I'K'}$ *with*

$$\Pi^{IK}_{I'K'}(a_{rs}) = \begin{cases} a_{rs} \mid & a_{rs} \in \Sigma_{I'K'} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Sigma_{I'K'}. \end{cases}$$

It is easy to see [Ochsenschläger & Rieke, 2010]:

**Theorem 2.** $\mathcal{L}_{IK} \supset \mathcal{L}_{I'K'}$ *for* $I' \times K' \subset I \times K$*, and therefore*

$$\Pi^{IK}_{I'K'}(\mathcal{L}_{IK}) \supset \Pi^{IK}_{I'K'}(\mathcal{L}_{I'K'}) = \mathcal{L}_{I'K'}.$$

However, the following example shows that the reverse inclusions

$$\Pi^{IK}_{I'K'}(\mathcal{L}_{IK}) \subset \mathcal{L}_{I'K'} \text{ for all } I' \times K' \subset I \times K \tag{6}$$

do not hold in general.

**Example 8.** *For a counterexample let us examine the 1-1-cooperation given by the automaton in Figure 19. Let the schedule SF again be given by the automaton* $\mathbb{SF}$ *in Figure 22a and the schedule SG be given by the automaton* $\mathbb{SG}$ *in Figure 23.*

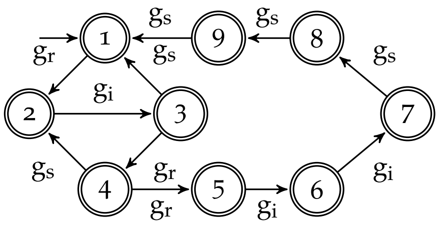


Figure 23: Schedule *SG* for the counterexample

*In the automaton* $\mathbb{SG}$ *immediately after entering a second handshake (state 4)* $G$ *may enter a third handshake but immediately after entering the first handshake (state 2)* $G$ *may not enter a second handshake. Now, for example,*

$$f_{s11}f_{s21}f_{s31}g_{r11}g_{i11}g_{r21}g_{r31} \in \mathcal{L}_{\{1,2,3\}\{1\}}.$$

*Hence*

$$f_{s21}f_{s31}g_{r21}g_{r31} \in \Pi^{\{1,2,3\}\{1\}}_{\{2,3\}\{1\}}(\mathcal{L}_{\{1,2,3\}\{1\}}),\ \textit{but}$$

$$f_{s21}f_{s31}g_{r21}g_{r31} \notin \mathcal{L}_{\{2,3\}\{1\}}.$$

The decidability status of (6) in general is not addressed in P7, but for many parameterised systems (6) holds, and therefore

$$\Pi^{IK}_{I'K'}(\mathcal{L}_{IK}) = \mathcal{L}_{I'K'},$$

which is a generalisation of $\pi^{IK}_{ik}(\mathcal{L}_{IK}) = L$.

**Definition 9** (Self-similarity)**.**
*A uniformly parameterised cooperation* $\mathcal{L}_{IK}$ *is called* self-similar *iff*

$$\Pi^{IK}_{I'K'}(\mathcal{L}_{IK}) = \mathcal{L}_{I'K'}\ \textit{for each}\ I' \times K' \subset I \times K.$$

So it is of interest to find conditions, which imply (6). Figure 23 is typical in the sense that it may serve as an idea to get a sufficient condition for self-similarity. It requires (a) two separate conditions, one for each schedule, (b) structuring schedules into phases, which may be shuffled in a restricted manner, (c) formalising "*how a cooperation partner is involved in several phases*", and (d) the more phases a cooperation partner is involved in, the less possibilities of acting in each phase he has. In Ochsenschläger & Rieke [2010] a sufficient condition for self-similarity is given, which is based on deterministic computations in shuffle automata. Under certain regularity restrictions this condition can be verified by a semi-algorithm.

In P7, an example is given that demonstrates the significance of self-similarity for verification purposes and it is shown in particular that for self-similar parameterised systems $\mathcal{L}_{IK}$ the parameterised problem of verifying a *uniformly parameterised safety property* can be reduced to finite many fixed finite state problems.

Complementary to this, in P8, *uniformly parameterised reliability properties* are defined based on this concept. The main result is a finite state verification framework for such uniformly parameterised reliability properties. In this framework the concept of structuring cooperations into phases enables completion of phases strategies. Consistent with this, corresponding success conditions can be formalised. These produce finite state semi-algorithms (independent of the concrete parameter setting) to verify the reliability properties.

*Well-behaved scalable systems* are a special class of parameterised systems [Ochsenschläger & Rieke, 2014]. The main motivation for this definition is to achieve that well-behaved scalable systems fulfil certain kind of safety properties if already one prototype system (depending on the property) fulfils that property. To this end, construction principles for well-behaved scalable systems are *design principles for verifiability* [Avizienis et al., 2004].

The research results with respect to research question RQ4 provided in P7 and P8 as well as ongoing research work in this area [Ochsenschläger & Rieke, 2014] show the significance of self-similarity for verification purposes and thus for the construction of well-behaved scalable systems.

## 2.6 RELATED WORK

The following research areas have been identified that are of major importance to the research questions regarding security of cooperating system design: *formal methods and model checking*, *characterisation of system properties*, *security requirements engineering*, and *verification approaches for parameterised systems*.

### 2.6.1 *Formal methods and model checking*

The presented approach can be compared with automata based methods as described in Alur & Henzinger [1995] or Kurshan [1994] as well as with the concurrency workbench Cleaveland et al. [1993], which uses the modal $\mu$-calculus as a specification language for properties [Stirling, 1989].

There exists a variety of verification tools which can be found in the literature. Some are based on model checking, others use proof systems. COSPAN [Kurshan, 1994] that is also the base of the commercial verification tool FormalCheck [Xie & Liu, 2007] is probably the closest to the SHVT. COSPAN is automata based and contains a homomorphism based abstraction concept. Since the transition labels of automata in COSPAN are in a Boolean algebra notation, the abstraction homomorphisms are Boolean algebra homomorphisms which correspond to non-erasing alphabetic language homomorphisms on the automata level. The SHVT, in addition, offers erasing homomorphisms as an abstraction concept. COSPAN also considers only linear satisfaction of properties. Thus fairness assumptions need to be made explicitly in this tool. In Hartel et al. [1999] ten tools in this area including an old version of SHVT are compared.

The main strength of the method presented here is the combination of an inherent fairness assumption in the satisfaction relation, a very flexible abstraction technique compatible with approximate satisfaction, and a suitable compositional and partial order method for the construction of only a partial state space. The construction of a Büchi automaton representing the property given by a PLTL formula that is used in the specific model checking algorithms implemented within the SHVT is using the algorithm given in Gerth et al. [1996]; Clarke et al. [1999]; Peled [2001].

### 2.6.2 *Characterisation of system properties*

In the information flow analysis approach presented in Guttman et al. [2003] for the *SELinux* system, a LTS is generated from the policy specifications that models the information flow policy. TL formulas are used to specify the security goals. The *NuSMV* model-checker [Cimatti et al., 2002] verifies the security goals on this LTS.

In Benenson et al. [2006] a formal framework based on three distinct classes of properties, namely safety, liveness and information flow is given, which makes it possible to reason distinguished within the formal system model. Examples of dependable systems are given in this framework in order to justify that these classes of system properties are sufficient to describe the functional requirements of dependable systems satisfying required fault-tolerance and security properties.

In Cederquist & Dashti [2011] the complexity of expressing fairness constraints for the Dolev-Yao model is analysed. The analysis is mainly based on type and size of communication buffers (bounded, unbounded).

A formal definition of *safety* and *liveness* properties is given in Alpern & Schneider [1985]. In Nitsche & Ochsenschläger [1996] a satisfaction relation, called *approximate satisfaction*, has been defined that expresses a possibilistic view on liveness and is equivalent to the satisfaction relation in Alpern & Schneider [1985] for safety properties. Besides these safety and liveness properties so called *hyperproperties* [Clarkson & Schneider, 2008] are of interest because they give formalisations for non-interference and non-inference.

The specification of the application level security requirements in this thesis is based on the security modelling framework developed by Fraunhofer SIT [Gürgens et al., 2002; Gürgens et al., 2005]. The underlying formal model describes system behaviours as (sets of) traces of actions, where these actions are associated with agents in the systems. Security properties are constraints on these sequences. In contrast to previous approaches it is not focused on a special type of security property. Formalisations of authenticity, different types of non-repudiation and confidentiality are presented within the framework. The method to derive security requirements described in this thesis results in requirements which fit to this framework.

### 2.6.3 *Security requirements engineering*

A comprehensive concept for an overall security requirements engineering process called Security Quality Engineering Methodology (SQUARE) is described in Mead & Hough [2006]; Mead [2007]. The authors propose a 9 step approach. The elicitation of the security requirements is one important step in the SQUARE process. In Mead [2007] several concrete methods to carry out this step are compared. These methods are based on misuse cases, soft systems methodology, quality function deployment, controlled requirements expression, issue-based information systems, joint application development, feature-oriented domain analysis, critical discourse analysis as well as accelerated requirements method. A comparative rating based on 9 different criteria is also given but none of these criteria measures the completeness of the security requirements elicited by the different methods. An overview of some formal and informal methods for the specification of secure systems is also given.

A similar approach based on the integration of Common Criteria (ISO/IEC 15408) called Security Requirements Engineering Process (SREP) is described in Mellado et al. [2006, 2007]. Both approaches Mead [2007] and Mellado et al. [2007] do not provide own formal security requirements specification techniques but propose to integrate other formal methods for this purpose.

In van Lamsweerde [2004] anti-goals derived from negated security goals are used to systematically construct threat trees by refinement of these anti-goals. Security requirements are then obtained as countermeasures. This method aims to produce more complete requirements than other methods based on misuse cases. The refinement steps in this method can be performed informally or formally.

In Firesmith [2003] different kinds of security requirements are identified and informal guidelines are listed that have proven useful when eliciting concrete security requirements. The author emphasises that there has to be a clear distinction between security requirements and security mechanisms.

In Haley et al. [2008] it is proposed to use Jackson's problem diagrams to determine security requirements which are given as constraints on functional requirements. Though this approach presents a methodology to derive security requirements from security goals, it does not explain the actual refinements process, which leaves open, the degree of coverage of requirements, depending only on expert knowledge. Hatebur et al. [2008] specifically addresses accountability by logging.

In Hatebur & Heisel [2009] patterns for expressing and analysing dependability requirements, such as confidentiality, integrity, availability, and reliability are given. The patterns are expressed as logical predicates. They are part of a pattern system that can be used to identify missing requirements.

There are a number of other approaches to solve specific problems in the security requirements refinement process, which are complementary to the work presented here. For example, in Bandara et al. [2003] Event Calculus and abductive reasoning is used for developing a language that supports specification and analysis of policy based systems, which can be used to detect modality conflicts and a range of application specific conflicts.

In Liu et al. [2002] actor dependency analysis is used to identify attackers and potential threats in order to identify security requirements. The so called i* approach facilitates the analysis of security requirements within the social context of relevant actors. In Giorgini et al. [2004] a formal framework for modelling and analysis of security and trust requirements at an organisational level is described. Both of these approaches target organisational relations among agents rather than functional dependence.

An overview of current security requirements engineering processes is given in Fabian et al. [2010] and Mellado et al. [2010].

Though all of the above mentioned approaches may lead to a sufficient level of security for the designed architecture, there is no obvious means by which they can be compared regarding the security requirements that they fulfil.

The application to specific application areas as for example vehicular communication networks or critical infrastructures has to consider application specific requirements. Dam monitoring applications with Automated Data Acquisition Systems (ADASs) are discussed in Parekh et al. [2010]; Myers et al. [2005]. Usually, an ADAS is organised as a Supervisory Control And Data Acquisition (SCADA) system with a hierarchical organisation. Details on SCADA systems organisation can be found in Coppolino et al. [2010]. In the majority of cases, SCADA systems have very little protection against the escalating cyber threats. Compared to traditional IT systems, securing SCADA systems poses unique challenges. In order to understand those challenges and the potential danger, Zhu et al. [2011] provides a taxonomy of possible cyber attacks including cyber-induced cyber-physical attacks on SCADA systems.

Besides identification of security requirements that is addressed in this thesis, the further security engineering process has to addresses issues such as how to mitigate risks resulting from connectivity and how to integrate security into a target architecture [Bodeau, 1994]. Specific mechanisms for enforcement of authenticity requirements which have been derived by the method proposed in this thesis comprise, for example, Trusted Computing (TC) techniques. TC technology standards provide methods for reliably verifying a system's integrity and identifying anomalous and/or unwanted characteristics

[Mitchell, 2005]. An approach for the generation of secure evidence records was presented in Richter et al. [2010].

### 2.6.4 *Verification approaches for parameterised systems*

Considering the behaviour verification aspect, which is one of the motivations to formally define well-behaved scalable systems, there are some other approaches to be mentioned.

An extension to the *Mur*$\varphi$ verifier to verify systems with replicated identical components through a new data type called RepetitiveID is presented in Ip & Dill [1999]. The verification is performed by explicit state enumeration in an abstract state space where states do not record the exact numbers of components. A typical application area of this tool are cache coherence protocols. The aim of Derepas & Gastin [2001] is an abstraction method through symmetry, which works also when using variables holding references to other processes. This is not possible in Mur$\varphi$. In Lakhnech et al. [2001], a methodology for constructing abstractions and refining them by analysing counter-examples is presented. The method combines abstraction, model-checking and deductive verification. In Basu & Ramakrishnan [2006], a technique for automatic verification of parameterised systems based on process algebra *CCS* [Milner, 1989] and the logic modal *mu-calculus* [Bradfield & Stirling, 2001] is presented. This technique views processes as property transformers and is based on computing the limit of a sequence of mu-calculus formula generated by these transformers. The above-mentioned approaches demonstrate that finite state methods combined with deductive methods can be applied to analyse parameterised systems. The approaches differ in varying amounts of user intervention and their range of application. A survey of approaches to combine model checking and theorem proving methods is given in Uribe [2000].

Far reaching results in verifying parameterised systems by model checking of corresponding abstract systems are given in Clarke et al. [2006]; Talupur [2006].

In Ochsenschläger & Rieke [2010] it is shown that the definition of uniformly parameterised cooperations is strongly related to *iterated shuffle products* [Jantzen, 1985], if the cooperations are "structured into phases". The main concept for such a condition are shuffle automata [Jedrzejowicz & Szepietowski, 2001] (multicounter automata [Björklund & Bojanczyk, 2007]) whose computations, if they are deterministic, unambiguously describe how a cooperation partner is involved in several phases.

The main contribution of P7 is to show how the parameterised problem of verifying a uniformly parameterised safety property can be solved by means of the self-similarity results of Ochsenschläger & Rieke [2010] and finite state methods.

It is well known that the general verification problem for parameterised systems is undecidable [Apt & Kozen, 1986; Suzuki, 1988]. However, this chapter introduces a formal framework to specify parameterised systems in a restricted manner, in order to achieve that these *well-behaved scalable systems* fulfil certain kind of safety properties if already one prototype system fulfils that property. Further work on construction principles for well-behaved scalable systems shows how to construct more complex systems by the composition of several synchronisation conditions based on this characterisation [Ochsenschläger & Rieke, 2014]. A similar approach is well-developed for hardware with respect to *design for testability* of physical faults [Avizienis et al., 2004].

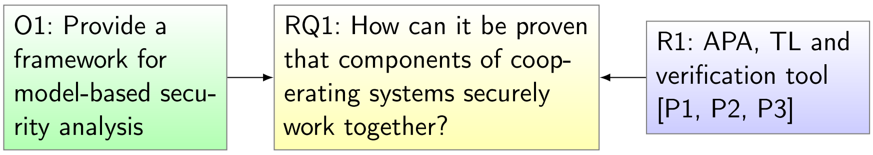
## 2.7 SUMMARY OF RESULTS

This chapter addresses security of cooperating system design. The specific objective is to support the development of secure and dependable systems by model-based analysis and verification of security and safety properties with regard to the design or redesign of a system, that is, support for *fault prevention and fault removal* in a system's design. With respect to the overall aim of this thesis – to provide a framework for security analysis of system behaviour – this chapter addresses the "Plan" activity in the Plan-Do-Study-Act (PDSA) cycle (cf. Figure 4). The general objective of this activity is to establish the objectives, identify security requirements, and analyse the design of the system.

The thesis provides a method to identify security requirements and express them formally; it provides methods and tools to analyse system design with respect to given requirements; and, it provides some construction principles that have to be taken into account in the design of *well-behaved* scalable systems. Thus, it addresses fault prevention and fault removal in the early stages of the security engineering process.

### 2.7.1 *APA, TL and verification tool*

With respect to *security of cooperating system design*, the first objective O1 of this thesis was to *provide a framework for model-based security analysis*. This motivated research question RQ1.


O1: Provide a framework for model-based security analysis

RQ1: How can it be proven that components of cooperating systems securely work together?

R1: APA, TL and verification tool [P1, P2, P3]

The results published in P1 – P3 address this research question and provide means to prove that - in the context of CS - the components work together in a desired manner. In the following, an overview of each of the papers contributing to result R1 is given.

### P1. The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems

This paper gives an overview about the main functions of the SHVT. The aim of the SHVT is to support the verification of cooperating systems. Cooperating systems are specific distributed SoS which are characterised by freedom of decision and loose coupling of their components. This causes a high degree of nondeterminism which has to be handled by the analysis methods. Typical examples of cooperating systems are telephone systems, communication protocols, smartcard systems, electronic money, and contract systems. In that context, verification is the proof that system components work together in a desired manner. At that, the main strength of the tool is the combination of an inherent fairness assumption in the satisfaction relation, an abstraction technique compatible with approximate satisfaction, and a suitable compositional and partial order method for the construction of only a partial state space.

### P2. The SH-Verification Tool

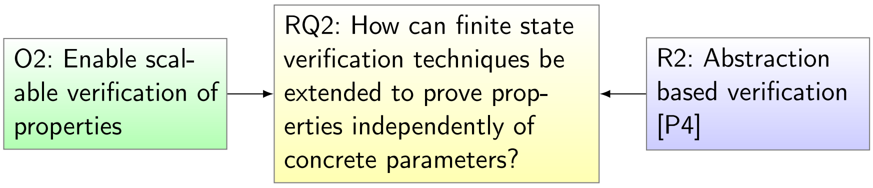
This paper gives an overview about the main components of the SHVT. With the help of an illustrative example, the usage of the methods described in P1 is shown. P2 contributes to research question RQ1 by the demonstration of the applicability of the methods developed in P1. Specifically, abstraction and temporal logic based reasoning is demonstrated. The SHVT's user interface and general handling has reached a level of maturity that enabled its successful application in the industrial area [Apel et al., 2007].

### P3. Development of formal models for secure e-services

This paper provides an extensive example for the use of the methods and tool described in P1 and P2. From e-government applications provided by project partners from the city of Cologne a typical example of an e-service implementation was selected. This e-service was modelled, augmented by an attacker model, and analysed using the SHVT. It has been shown that even if the correct behaviour of an e-service is proven under assumptions about the interfaces to the environment and about reasonable input it is necessary to inspect the system behaviour and ask 'what if' questions to check the behaviour of the model against given attack patterns or slightly changed assumptions about the environment. A vulnerability - a race condition problem - was found that leads in the end to a misrouting effect. Race conditions are just the most security-relevant type of concurrency problem [Viega & McGraw, 2002].

### 2.7.2 *Abstraction based verification*

Objective O2 of this thesis was to *enable scalable verification of properties*. Given the results from the work on RQ1, namely, the finite state verification framework, research question RQ2 addresses extensions to this framework with respect to objective O2.



O2: Enable scalable verification of properties

RQ2: How can finite state verification techniques be extended to prove properties independently of concrete parameters?

R2: Abstraction based verification [P4]

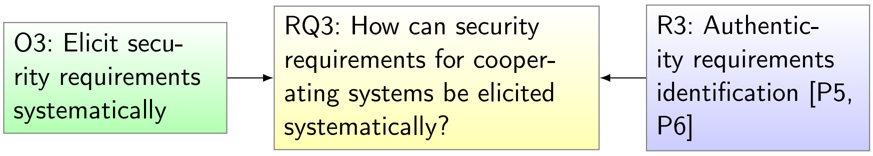
The results published in P4 address this research question and thus provide result R2.

P4. Abstraction Based Verification of a Parameterised Policy Controlled System

This paper extends the tool supported verification techniques presented in P1 and P2 by an approach to verify entire families of critical systems, independent of the exact number of replicated components. This is demonstrated by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario. Verification results for configurations with fixed numbers of components are used to choose an appropriate property preserving abstraction that provides the basis for an inductive proof that generalises the results for a family of systems with arbitrary settings of parameters. The inductive proof uses the construction of the behaviour of the parameterised system to show that it results in identical abstract system behaviour for any given parameter configuration. This allows the verification of parameterised systems by constructing abstract systems that can be model checked.

### 2.7.3 *Authenticity requirements identification*

Objective O3 of this thesis was to *elicit security requirements systematically*. This motivated research question RQ3.



O3: Elicit security requirements systematically

RQ3: How can security requirements for cooperating systems be elicited systematically?

R3: Authenticity requirements identification [P5, P6]

The results published in P5 and P6 address this research question. In the following, an overview of the papers contributing to result R3 is given.

P5. Identification of Security Requirements in Systems of Systems by Functional Security Analysis

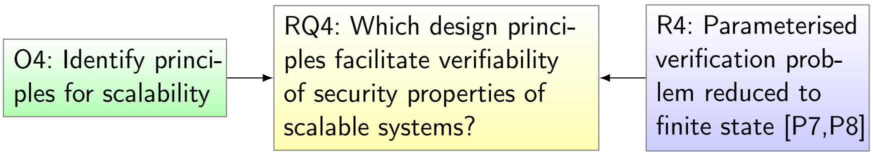
This book chapter is an extended version of [Fuchs & Rieke, 2009]. It provides a model-based approach to systematically identify security requirements for cooperating systems. The proposed method comprises the tracing down of functional dependencies over system component boundaries right onto the origin of information as a functional flow graph. Based on this graph, comprehensive sets of formally defined authenticity requirements for the given security and dependability objectives are systematically deduced. The proposed method thereby avoids premature assumptions on the security architecture's structure as well as the means by which it is realised. The most common problem with security requirements is that they tend to be replaced with security-specific architectural constraints that may unnecessarily constrain the choice of the most appropriate security mechanisms [Firesmith, 2003]. Therefore, the proposed approach avoids to break down the overall security requirements to requirements for specific components or communication channels prematurely. So the requirements identified by this approach are independent of decisions not only on concrete security enforcement mechanisms to use, but also on the structure, such as hop-by-hop versus end-to-end security measures.

P6. A Trusted Information Agent for Security Information and Event Management

This paper demonstrates on an example of a critical infrastructure - a hydroelectric power plant - how security requirements for such SoS can be derived by application of the requirements elicitation method described in P5. The elicited requirements provide implications for the design of the security architecture which - in this case - leads to the application of trusted computing technology.

### 2.7.4 *Parameterised verification problem reduced to finite state*

Objective O4 of this thesis aimed to *identify principles for scalability*. This motivated research question RQ4.


O4: Identify principles for scalability
RQ4: Which design principles facilitate verifiability of security properties of scalable systems?
R4: Parameterised verification problem reduced to finite state [P7,P8]

The results published in P7 and P8 address this research question. In the following, an overview of the papers contributing to result R4 is given.

### P7. Security Properties of Self-similar Uniformly Parameterised Systems of Cooperations

In this paper uniform parameterisations of cooperations are defined in terms of formal language theory, such that each pair of partners cooperates in the same manner, and that the mechanism (schedule) to determine how one partner may be involved in several cooperations, is the same for each partner. Generalising each pair of partners cooperating in the same manner, for such systems of cooperations a kind of self-similarity is formalised. From an abstracting point of view, where only actions of some selected partners are considered, the complex system of all partners behaves like the smaller subsystem of the selected partners. For verification purposes, so called uniformly parameterised safety properties are defined. Such properties can be used to express privacy policies as well as security and dependability requirements. It is shown, how the parameterised problem of verifying such a property is reduced by self-similarity to a finite state problem.

### P8. Reliability Aspects of Uniformly Parameterised Cooperations

In this paper reliability aspects of systems, which are characterised by the composition of a set of identical components are examined. These components interact in a uniform manner, described by the schedules of the partners. Such kind of interaction is typical for scalable complex systems with cloud or grid structure. In addition to the safety properties of such *uniformly parameterised cooperations* which have been analysed in P7 reliability of such systems in a possibilistic sense is considered. This is formalised by always-eventually properties, a special class of liveness properties using a modified satisfaction relation, which expresses possibilities. As a main result, a finite state verification framework for uniformly parameterised reliability properties is given. The keys to this framework are structuring cooperations into phases and defining closed behaviours of systems. In order to verify reliability properties of such uniformly parameterised cooperations, finite state semi-algorithms that are independent of the concrete parameter setting are used.

### 2.7.5 *Conclusion*

In summary, the results presented in this chapter aim at *prevention of faults in the design phase* and *removal of faults in redesign phases*. Specifically, the security and dependability properties expressed by the attributes *integrity* (= *authenticity within a phase*), *safety*, *reliability*, and *availability* have been analysed.

The modelling framework and tool introduced in this chapter is not only used within this chapter, for example, in P3 for security analysis of e-services provided by the city of Cologne, in P4 for a collaboration scenario with trusted and untrusted clients, in P5 for vehicle-to-vehicle communications. It is further utilised to model and analyse quite different systems throughout this thesis.

In Chapter 3 the modelling framework is utilised in P9, P10, and P11 for network models including assets, vulnerabilities, security policies and attacker behaviour, and in P12 for validation of Xtensible Access Control Markup Language (XACML) policies.

In the work presented in Chapter 4 the framework is extended for runtime usage. It is utilised in P13 for an online credit application process, in P14 for a pickup process from a logistics scenario of the project Alliance Digital Product Flow (ADiWa) [ADiWa Konsortium, 2012], in P18 for fraud detection in a Mobile Money Transfer Service (MMTS) scenario, and in P19 for a hydroelectric power plant model. The SHVT is a core component of the Predictive Security Analyser (PSA) presented in Chapter 4.

Additional peer-reviewed publications related to the methods and tool described in this chapter with participation of the author of this thesis comprise Ochsenschläger et al. [2000b]; Ochsenschläger et al. [2000]; Herfert et al. [2004]; Apel et al. [2007]; Fuchs & Rieke [2009]; Kaindl et al. [2012]; Ochsenschläger & Rieke [2012b]; Khan et al. [2013]. An invited talk related to P5 has been given at the CAST workshop 2010 on Mobile Security for Intelligent Cars [Rieke, 2010b]. In further noteworthy work of the author of this thesis APA have been used to model a framework for secure e-government [Rieke, 2002] as well as critical parts of the infrastructure of the German ehealth card [Rieke, 2009b].

# 3 SECURITY OF SYSTEM CONFIGURATIONS

> *It is easier to perceive error than to find truth, for the former lies on the surface and is easily seen, while the latter lies in the depth, where few are willing to search for it.*
>
> — Johann Wolfgang von Goethe, Maximen und Reflexionen (1823)

AIM OF THIS CHAPTER. A systematic security assessment of information infrastructures requires an analytical process to identify the critical components and their interplay, to determine the threats and vulnerabilities, to assess the risks, and to prioritise countermeasures where risk is unacceptable. To achieve this objective, this chapter presents an approach that builds on a model-based construction of an attack graph taking into account constraints given by the network security policy. The most distinctive feature of this approach is the ability to compute abstract representations of these complex graphs that enable comparison of focussed views on the behaviour of the system. In order to analyse resilience of information infrastructures against exploits of unknown vulnerabilities by zero day attacks, generic vulnerabilities for each installed product and affected service are added to the model. Furthermore, an approach for the validation and deployment of a security policy is given.

This chapter is based on the work published in P9, P10, P11, and P12 (cf. Table 2).

## 3.1 INTRODUCTION

Information and Communications Technology (ICT) is creating innovative systems and extending existing infrastructure to such an interconnected complexity that predicting the effects of small internal changes (e.g., firewall policies) and external changes (e.g., the discovery of new vulnerabilities and exploit mechanisms) becomes a major problem [Bullock & Cliff, 2004]. The security of such a complex networked system essentially depends on a concise specification of security goals, their correct and consistent transformation into security policies, and an appropriate deployment and enforcement of these policies. This has to be accompanied by a concept to adapt the security policies to changing context and environment, usage patterns and attack situations. To help to understand the complex interrelations of security policies, ICT infrastructure and vulnerabilities and to validate security goals in such a setting, tool-based modelling techniques are required that can efficiently and precisely predict and analyse the behaviour of such complex interrelated systems. Known and unknown vulnerabilities may be part of each of the connected components and communication paths between them. Analysis methods should guide a systematic evaluation of such a critical information infrastructure assist the persons in charge with finally determining exactly how to configure protection measures and which security policy to apply.

The aim of this thesis with respect to security of system configurations is expressed by the following four objectives (cf. Figure 3).

O5: Configure systems so that vulnerabilities are protected or hidden.

O6: Identify network configuration risks.

O7: Assess zero-day exploit vulnerability.

O8: Validate implementation of security goals.

In order to configure systems securely (objective O5), a formal modelling framework is presented that, on the one hand, represents the information system and the security policy, and, on the other hand, a model of attacker capabilities and profile. Building on this, a graph representing all possible attack paths called *attack graph* can be automatically computed. The attack graph allows to investigate whether a given security policy successfully blocks attack paths and is robust against changes in the given vulnerability setting.

For the identification of network configuration risks (objective O6), a method to compute abstract representations of an attack graph is proposed that helps to overcome the problems to analyse an attack graph of a realistic network configuration directly because of the huge size.

This is especially important when possible attacks based on unknown vulnerabilities are added to the model in order to identify and assess network configuration risks with respect to zero-day exploit vulnerabilities (objective O7).

In order to validate the implementation of security goals (objective O8), the analysis of a network security policy with respect to compliance with the high-level security and safety requirements is considered and an approach for the runtime management of policies and their update and synchronisation process is proposed.

This chapter is structured as follows. The modelling approach is introduced in Section 3.2, while Section 3.3 considers the use of abstraction techniques for high-level aspect visualisation. Section 3.4 presents an approach to analyse resilience of critical information infrastructures against exploits of unknown vulnerabilities. Section 3.5 addresses policy validation and deployment. Section 3.6 gives an overview of related work. Finally, this chapter ends with a summary of the results in Section 3.7.

## 3.2 CONFIGURATION ANALYSIS APPROACH

The systematic protection of critical information infrastructures requires an analytical process to identify the critical components and their interplay, to determine the threats and vulnerabilities, to assess the risks and to prioritise countermeasures where risk is unacceptable. A typical means by which an attacker (directly or using malware such as blended threats) tries to break into such a network is, to use combinations of basic exploits to get more information or more credentials and to capture more assets step by step. To find out if there is a combination that enables an attacker to reach critical network resources or block essential services, it is required to analyse all possible sequences of basic exploits, so called *attack paths*.

In order to analyse a networked Systems of Systems (SoS) with respect to such threats the following research questions have to be addressed.

RQ5a: *How can exploitation possibilities of networked systems' vulnerabilities be analysed?*

RQ5b: *How can attacker behaviour be incorporated into the system model and the analysis?*

RQ5c: *Which attacks would not be detected?*

To answer these research questions, a formal modelling framework has been developed and published in P9 that, on the one hand, represents the information system and the security policy (RQ5a), and, on the other hand, a model of attacker capabilities and profile (RQ5b). It is extensible to comprise intrusion detection components (RQ5c) and optionally a model of the system's countermeasures. Based on this model a graph representing all possible attack paths can be automatically computed. It is called an *attack graph* in the following text. Based on this attack graph, it is now possible to find out whether a given security policy successfully blocks attack paths and is robust against changes in the given vulnerability setting.

### 3.2.1 *Network and vulnerability model*

The set of all hosts of the information system consists of the union of the hosts of the ICT network and the hosts of the attacker(s). Following the M2D2 model [Morin et al., 2002], *products* are the primary entities that are vulnerable. A *host configuration* is a subset of products that is installed on that host and *affects* is a relation between vulnerabilities and sets of products that are affected by a vulnerability. A host is *vulnerable* if its configuration is a superset of a vulnerable set of products and the affected services are currently running. In order to conduct a subsequent comparative analysis of attack paths, an asset prioritisation as to criticality or worth regarding relative importance of the assets is required.

Figure 24 depicts a small example scenario that will be used to illustrate the modelling concepts and typical analysis outcome throughout this chapter. One possible attack path is sketched in the scenario.

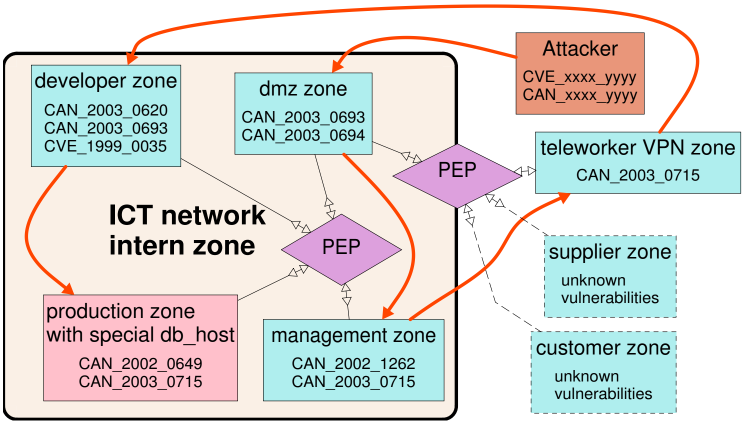


Figure 24: Attack path in vulnerable ICT network

The model of the network security policies is based on the Organization Based Access Control (OrBAC) model [Cuppens et al., 2004]. The advantage of this choice is that it is possible to link the policies in the formal model at an abstract level to the low level vendor specific policy rules for the Policy Enforcement Points (PEPs) such as firewalls in the concrete ICT network. Following the OrBAC concept, the policy is given at an abstract level in terms of *roles* (an abstraction of subjects), *activities* (an abstraction of actions) and *views* (an abstraction of objects). A *subject* in this model is any host. An *action* is a network service such as snmp, ssh or ftp. Actions are represented by a triple of protocol, source port and target port. An *object* is a message sent to a target host. Currently only the target host or rather the role of the target host is used for the view definition here. To specify the access control policy using this approach, *permissions* are given between role, activity and view. An example for policy permissions is given in Table 5.

In order to model mobile components that can transport malware from one network zone to another, it is convenient to allow a host to play different roles. For example, a laptop used for teleworking that plays the role telework_host can additionally be permitted to play the role intern_host. In this case an attack path could cross the zones from telework_host to intern_host without any restrictions by the network security policy.

Table 5: Network security policy

| Role (source) | View (target) | Activity (service) |
|---|---|---|
| internet_host | internet_host | any_activity |
| any_role | dmz_host | ssh |
| any_role | dmz_host | smtp |
| dmz_host | intern_host | ssh |
| intern_host | any_role | net |
| intern_host | internet_host | ftp |
| intern_host | internet_host | rsh |
| intern_host | dmz_host | ssh |
| db_host | production_host | rpc |
| teleworker_host | dmz_host | any_activity |

Vulnerability specifications for the formal model are derived from the Common Vulnerabilities and Exposures (CVE) [MITRE Corporation, 2013b] descriptions. CVE comprises a list of virtually all publicly known information security vulnerabilities and exposures. The CVE name is the 13 character ID used by the CVE standards group to uniquely identify a vulnerability. Additional information about the vulnerabilities also covers preconditions about the target host as well as network preconditions. Furthermore, the impact of an exploitation of a vulnerability is described. The specifications for the formal model of the vulnerabilities additionally comprise the vulnerability *range* and *impact type* assessments provided by the National Vulnerability Database (NVD) [NIST Computer Security Resource Center, 2013b]. Of course, other kinds of vulnerabilities could be added to the model in a similar manner. The Common Vulnerability Scoring System (CVSS) [FIRST.org, Inc., 2013] provides universal severity ratings for security vulnerabilities. These ratings are used in the model to assess the threat level.

The information model presented so far covers the description of a (static) configuration of an ICT network and its vulnerabilities. In the formal model such a configuration describing the *state* of the network is represented by Asynchronous Product Automaton (APA) state components (cf. Section 2.2.1).

### 3.2.2 *Attacker model*

With respect to research question RQ5b attacker behaviour has to be represented in the model. To have a vulnerable product installed on some host, does not necessarily imply, that someone can exploit that vulnerability. A target host *is configured vulnerable*, if (1) the target host has installed a product or products that are vulnerable with respect to the given vulnerability, and (2) necessary other preconditions are fulfilled (e.g. some vulnerabilities require that a trust relation is established as for example used in remote shell hosts allow/deny concepts). The second precondition to exploit a vulnerability is, that the target host *is currently running the respective products* such as a vulnerable operating system or server version. If a user interaction is required this also requires that the vulnerable product is currently used (e.g. a vulnerable Internet explorer). The third necessary precondition is, that the *network security policy permits* that the target host is reachable on the port the vulnerable product is using from the host the attacker selected as source.

The knowledge of exploits and hosts and the credentials on the known hosts constitute an attackers profile. Knowledge about hosts changes during the computation of the attack graph because the attacker might gain new knowledge when capturing hosts. On the other hand, some knowledge may become outdated because the enterprise system changes ip-numbers or other configuration of hosts and reachability. In case a vulnerability is exploited, the model has to cover the *effects for the attacker* (e.g. to obtain additional user or root credentials on the target host) and also the *direct impact on the network and host* such as, to shut down a service caused by buffer overflow.

Attacker capabilities are modelled by the atomic exploits and by the strategy to select and apply them. A state transition modelling an exploit is constructed from, (1) a predicate that states that the attacker *knows* this exploit, (2) an expression to select source and target hosts for the exploit, (3) a predicate that states that the target *host is vulnerable* by this exploit, (4) an expression for the impact of the execution of this exploit on the attacker and on the target host as for example the shut down of services. Optional add-ons are, an assignment of cost benefit ratings to this exploit and intrusion detection checks. Several different attackers can easily be included because an attacker is modelled as a role not a single instance and the tool can automatically generate multiple instances from one role definition. Modelling of Denial of Service (DoS) attacks aiming to block resources or communication channels either directly or by side effects require a much more detailed model of the resources involved. This could be accomplished using the presented framework but is out of scope of this work.

### 3.2.3 *Behaviour and properties of the model*

To describe how actions of attacker(s) and actions of the system can change the state of the model, specifications of *APA state transitions* are used. These state transitions represent atomic exploits and optionally the actions that the system executes itself (e.g., to implement vital services).

The Simple Homomorphism Verification Tool (SHVT) is used to analyse this model. It manages the components of the model, allows to select alternative parts of the specification and automatically "glues" together the selected components to generate a combined model of ICT network specification, vulnerability and exploit specification, network security policy and attacker specification. After an

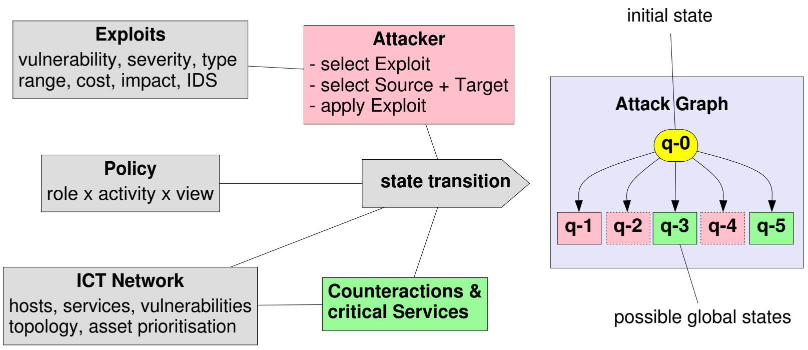


Figure 25: Computation of an attack graph

initial configuration is selected, the attack graph is automatically computed by the SHVT (see Figure 25). Formally, the attack graph is the Reachability Graph (RG) (see Definition 3) of the corresponding APA model.

A subgraph of the attack graph for the simple example scenario from Figure 24 is shown in Figure 26. This graph was computed under the assumption that the attacker knows all exploits.

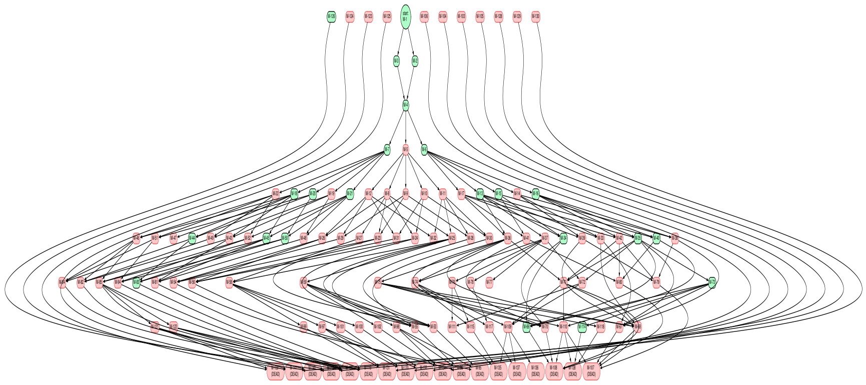


Figure 26: Subgraph of attack graph of simple example scenario

Figure 27 shows a detail of this attack graph. For better readability, the interpretations at the edge labels are omitted. For example, the edge $q_{13} \longrightarrow q_{38}$ depicts the application of an exploit where attacker A uses the ssh vulnerability CAN_2003_0693 and there is a second exploit (which is stealth, that means not detectable by intrusion detection systems) with the same state transition. The edge $q_{38} \longrightarrow q_{73}$ depicts an action of the system to restart the ssh daemon and the edge $q_{73} \longrightarrow q_{73}$ depicts an action that models the availability of a critical service.

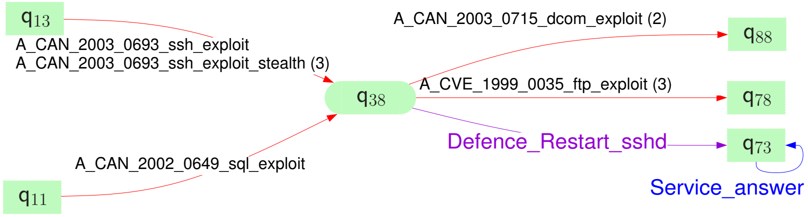

q13
A_CAN_2003_0693_ssh_exploit
A_CAN_2003_0693_ssh_exploit_stealth (3)
A_CAN_2003_0715_dcom_exploit (2)
q88
q38
A_CVE_1999_0035_ftp_exploit (3)
q78
Defence_Restart_sshd
q73
A_CAN_2002_0649_sql_exploit
Service_answer
q11

Figure 27: Attack graph detail

For very large models, on the fly analysis allows to stop computation of the reachability graph automatically when specified conditions are reached or invariants are broken.

### 3.2.4 *Cost benefit analysis*

Cost benefit analysis can be used as a means to help to assess the likely behaviour of an attacker. Cost ratings (from the view of an attacker) can be assigned to each exploit, for example, to denote the time it takes for the attacker to execute the exploit or the resources needed to develop an exploit. Cost ratings can also be based on the severity ratings given by CVSS. If not only technical vulnerabilities are modelled but also human weaknesses are considered, then cost could be, for example, the money needed to *buy* a password.

Based on these cost assignments (weights of edges), the shortest path from the root of the attack graph to a node representing a successful attack can be computed using Dijkstra's well-known algorithm. This path represents the least expensive combined attack breaking a given security goal.

A benefit for the attacker based on the negative impact he achieves can also be assigned, for example to indicate the *worth* regarding relative criticality of the captured asset.

**Example 9.** *Costs are directly assigned to the atomic exploits in this example, whereas the benefit for a transition is computed as the worth of the target host multiplied by the rank of the access right gained (cf. Figure 28).*

*Figure 29 depicts an attack path computed by cost benefit analysis (cf. P9).*

Summarised costs and benefits can be compared for selected paths or the whole graph and used for example to find the node with the greatest benefit for a potential attacker. Extensions of such analysis methods could be used for minimum-cost network hardening (cf. `http://www.patentstorm.us/patents/7555778/fulltext.html`).

Other security related properties such as the probability of being detected by intrusion detection systems can be associated with APA transitions. In Figure 29 steps of the attacker that are not detected by Intrusion Detection System (IDS) components are coloured green, detected steps are coloured orange. This information when evaluated in the analysis of an attack graph answers research question RQ5c

| Exploit | Cost |
|---|---|
| CAN_2003_0693_ssh_exploit | 3 |
| CAN_2003_0693_ssh_exploit_stealth | 4 |
| CVE_1999_0035_ftp_exploit | 2 |
| CAN_2003_0620_man_db_exploit | 3 |
| CAN_2003_0715_dcom_exploit | 4 |
| CAN_2003_0694_sendmail_exploit | 4 |
| CAN_2002_0649_sql_exploit | 4 |
| rsh_login | 1 |

(a) Exploit costs

| Host | Worth |
|---|---|
| nix_host | 2 |
| ms_host | 2 |
| db_server | 9 |
| portal | 4 |

(b) Host criticality

| Access | Rank |
|---|---|
| none | 1 |
| restricted_user | 2 |
| user | 3 |
| db_user | 4 |
| root | 5 |

(c) Access rank

Figure 28: Cost benefit values

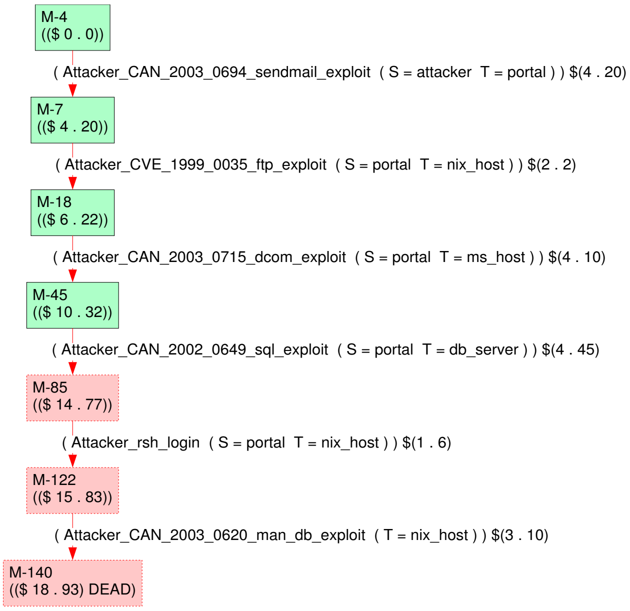

M-4
(($ 0 . 0))
( Attacker_CAN_2003_0694_sendmail_exploit ( S = attacker T = portal ) ) $(4 . 20)
M-7
(($ 4 . 20))
( Attacker_CVE_1999_0035_ftp_exploit ( S = portal T = nix_host ) ) $(2 . 2)
M-18
(($ 6 . 22))
( Attacker_CAN_2003_0715_dcom_exploit ( S = portal T = ms_host ) ) $(4 . 10)
M-45
(($ 10 . 32))
( Attacker_CAN_2002_0649_sql_exploit ( S = portal T = db_server ) ) $(4 . 45)
M-85
(($ 14 . 77))
( Attacker_rsh_login ( S = portal T = nix_host ) ) $(1 . 6)
M-122
(($ 15 . 83))
( Attacker_CAN_2003_0620_man_db_exploit ( T = nix_host ) ) $(3 . 10)
M-140
(($ 18 . 93) DEAD)

Figure 29: Attack path with cost benefit annotations

and can lead to improvements of a given configuration of a critical information infrastructure.

## 3.3 SYSTEMATIC RISK IDENTIFICATION

The SHVT usually computes attack graphs of some million edges in acceptable time and space. The problem now is that it is impossible to visualise a graph of that size. It is evident that in order to make use of the security related information that is available within this complex behaviour representation an abstraction process is needed. This abstraction should condense millions of specific transitions into a few human-understandable abstract transitions thus enabling to visualise and analyse compacted information focussed on interesting aspects of the behaviour like those expressed by the following research questions.

RQ6a: *What are the effects of changes to the network configuration on overall vulnerability?*

RQ6b: *What is the most likely attacker behaviour and most effective countermeasure?*

RQ6c: *Will countermeasures of the system under attack succeed?*

To address these research questions, it has been proposed in P10 to use an abstraction based approach using the concept depicted in Figure 14.

Abstract representations of an attack graph can be computed and used to visualise and analyse compacted information focussed on interesting aspects of the behaviour. The behaviour abstraction of an APA can be formalised by alphabetic language homomorphisms (see Section 2.2.3). The mappings used to compute the abstract representations of the behaviour have to be *property preserving*, in order to assure that properties are *transported* as desired from a lower to a higher level of abstraction and no critical behaviour is hidden by the mapping (cf. Section 2.2.3).

**Example 10** (Definition of an abstract representation)**.** *Figure 30 defines a mapping of the transitions representing an exploit of a vulnerability to the respective* range *and* impact type *assessments of the vulnerabilities (cf. Section 3.2.1). Range types of the vulnerabilities in the example scenario are* remote *(remotely exploitable) and* local *(locally exploitable). Impact types used here are* unspecific *(provides unauthorised access),* user *(provides user account access) and* root *(provides administrator access).*

*This mapping denotes, that all transitions (leaves of the tree) are to be represented by their respective father nodes, namely* system, preprocessing, unspecific, user, root *and* local *in the abstract representation. The nodes* system *and* preprocessing *are coloured in grey, symbolising that they are mapped to $\epsilon$, that means the transitions represented by these nodes will be invisible in the abstract representation. Please ignore the notation* (Pol) *at the node* remote *for the moment.*

*Figure 31 shows the result of application of the mapping in Figure 30 to the attack graph from Figure 26. This computed abstract representation (a graph with only 20 states and 37 edges) gives a visualisation focussing on the transition types* root, user, unspecific *and* local*. The simplicity of this mapping that guarantees that properties are preserved was automatically proven by the tool.*

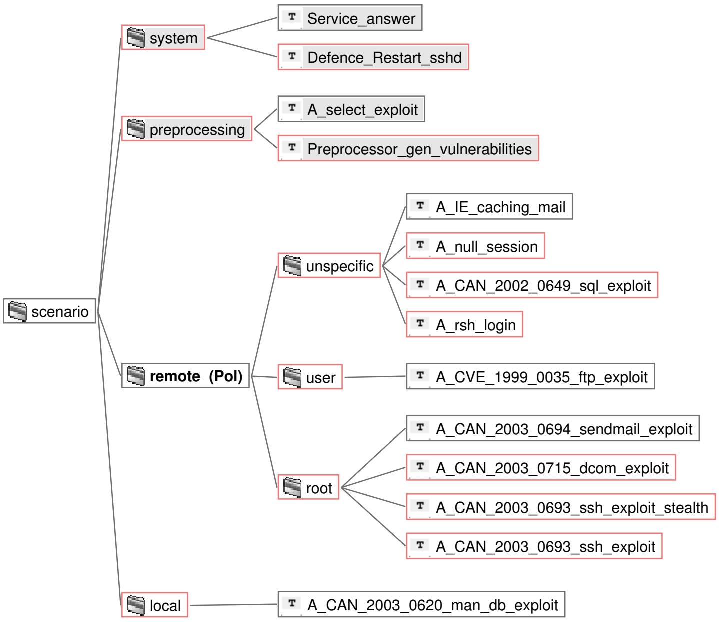

scenario
system
Service_answer
Defence_Restart_sshd
preprocessing
A_select_exploit
Preprocessor_gen_vulnerabilities
unspecific
A_IE_caching_mail
A_null_session
A_CAN_2002_0649_sql_exploit
A_rsh_login
remote (Pol)
user
A_CVE_1999_0035_ftp_exploit
root
A_CAN_2003_0694_sendmail_exploit
A_CAN_2003_0715_dcom_exploit
A_CAN_2003_0693_ssh_exploit_stealth
A_CAN_2003_0693_ssh_exploit
local
A_CAN_2003_0620_man_db_exploit

Figure 30: Definition of an abstract representation of the attack graph

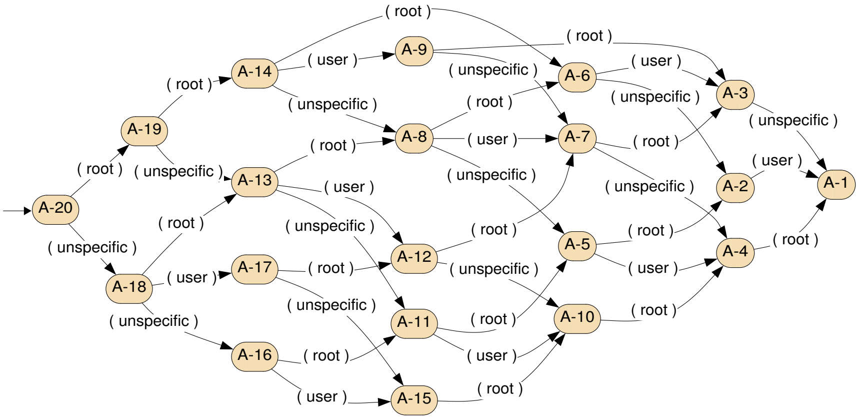
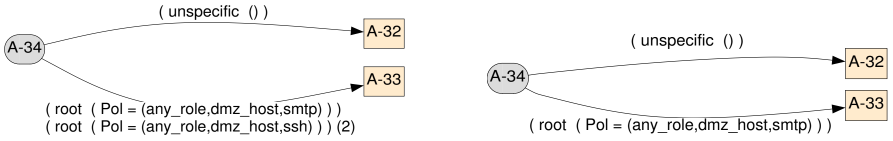


Figure 31: Abstract view on an attack graph

refined mapping. To find out which policies permit the attacks shown in Figure 31, a refinement of the abstraction defined in Figure 30 is necessary. It is possible to "fine tune" the mapping so that the interpretation variables (cf. Definition 2) stay visible in the abstract representation. In this case the binding of the interpretation variable Pol that contains the respective policy can be visualised. This is denoted by (Pol) in the node remote in Figure 30. The corresponding refined abstract representation is a graph with 34 states and 121 edges when computed on the attack graph in Figure 26. The initial nodes and edges of this graph are shown in Figure 32a. In comparison to the corresponding edges $A_{20} \longrightarrow A_{19}$ and $A_{20} \longrightarrow A_{18}$ of the graph in Figure 31 now the details on the related policies are visible.


( unspecific () )
A-34
A-32
A-33
( root ( Pol = (any_role,dmz_host,smtp) ) )
( root ( Pol = (any_role,dmz_host,ssh) ) ) (2)
( unspecific () )
A-34
A-32
A-33
( root ( Pol = (any_role,dmz_host,smtp) ) )

(a) (any_role, dmz_host, ssh/smtp)    (b) (any_role, dmz_host, smtp)

Figure 32: Details in the abstract view

ADAPT/OPTIMISE THE SYSTEM CONFIGURATION. Further analysis reveals, that, if the example policy given in Table 5 is changed to allow only smtp instead of ssh and smtp for any_role to dmz_host then the analysis yields of course a smaller graph than the original shown in Figure 26, the coarse abstract representation in Figure 31 is the same, but the finer mapping with interpretation variable Pol visible results in a different representation which is shown in Figure 32b.

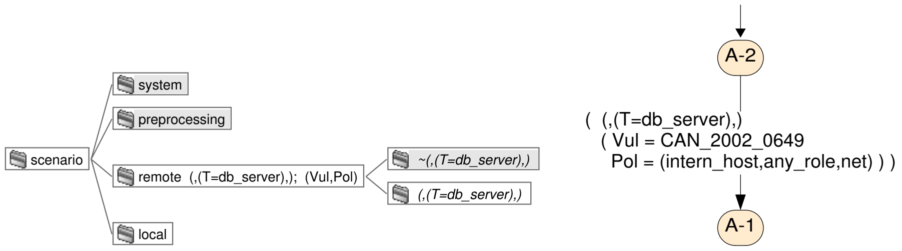
If alternatively the policy is restricted to allow only ssh instead of ssh and smtp in the above example, then the result is yet a different attack graph but the abstract view in Figure 31 is still the same.

This analysis demonstrates that there may be differences in the detailed attack graphs but no differences in the abstract representations thereof. This indicates that the different policies are equally effective (or not) concerning the enforcement of security goals on the abstract level, even if variations in the attack paths are covered by different policy rules.

USING PREDICATES TO DEFINE ABSTRACTIONS. Let us now assume that the host db_server in the scenario is the most valuable and mission critical host in the ICT network. So we want to know if in the given scenario, (1) attacks to the db_server are possible, (2) on which vulnerabilities they are based, and, (3) which policy rules are directly involved.

The abstraction given by Figure 33a exemplifies how predicates can be used to define such a mapping. In this mapping the predicate (T=db_server) matches only those transitions that model direct attacks to the target host db_server. Furthermore the bindings of the interpretation variables Vul and Pol that contain the respective vulnerability and policy are used in the mapping. The remote transitions that don't match that predicate are mapped to $\epsilon$ and so are invisible.

Evaluating this abstraction on the attack graph from Figure 26 above results in the simple graph given in Figure 33b. This proves that, (1) in the current policy configuration attacks to the db_server are possible, (2) those attacks are based on exploits of the vulnerability CAN_2002_0649, and, (3) they are utilising the policy permission (intern_hosts,any_role,net). So to prevent this attack, it has to be decided, whether it is more appropriate to uninstall the product that is hurt



(a) Abstraction used

(b) Resulting graph

Figure 33: Focus on attacks to the host db_server

by this vulnerability or to restrict the internal hosts in their possible actions by replacing the above policy with a more restrictive one.

### 3.3.1 *Countermeasure model and liveness properties*

With respect to research question RQ6c liveness properties (cf. Section 2.2.3) have to be considered. Liveness properties in the context of ICT infrastructure security analysis cover availability and business continuity aspects for example with respect to DoS attacks. When a system's countermeasures and the behaviour of vital services the system provides are included in the model, then availability properties such as the system's resilience with respect to DoS attacks can be analysed.

**Example 11.** *An example for the inclusion of countermeasures and critical services in the model is given by the transitions* Defence_restart_sshd *and* Service_answer *in Figure 27. If, for example, as a side effect of an* ssh_exploit *the attacker kills the* sshd *then afterwards the* sshd *is not active on the respective host and so some service possibly cannot answer requests anymore. Now additionally a system countermeasure is considered that restarts the* sshd*. No other details are added to keep the model small.*

A typical liveness question in this scenario is "Will a client still get answers from a server when the network is attacked?". In terms of Temporal Logic (TL) the property in question above can be written as G F Service_answer (always eventually Service_answer). This is a specialisation of the more general research question RQ6c. An appropriate type of model checking, in this case *approximate satisfaction* of TL (cf. Section 2.2.3) can now be used to answer this kind of questions.

## 3.4 zero-day exploit assessment

The analysis of network security with respect to unknown zero day attacks [Wang et al., 2010] is a relatively new research topic [Kotenko

& Chechulin, 2012]. Zero day attacks can be defined as attacks which use unknown vulnerabilities [Kotenko & Chechulin, 2012]. This motivates the following research question.

RQ7: *To which extent is a networked system resilient against exploits of unknown vulnerabilities?*

One way to consider resilience of an information infrastructure against attacks to unknown vulnerabilities is, to define a new vulnerability for each installed product. For the model of the scenario used in this chapter this has been done in P11 by definition of a new vulnerability called CAN_generic with a variable part for the affected service. In the same way a generic exploit based on this vulnerability is defined. Now in the preprocessing phase a state transition selects an arbitrary product and inserts a generic vulnerability CAN_generic for that product and the related service. Because the reachability analysis considers every possible choice of product, all alternatives are evaluated in the attack graph.

When analysing the (now much larger) attack graph, the mapping in Figure 34 exemplarily shows a possible use of resilience analysis. The state transition modelling an exploit of an unknown generic vulnerability uses the additional interpretation variables RS and RT, where RS denotes the role of the source host and RT the role of the target host. So the given predicate $(\mathsf{RS} = \mathsf{RT})$ matches only those transitions that model attacks of hosts in the same role (within the same zone). Now the attacks that fulfil this predicate are mapped to $\epsilon$ (coloured in grey in the mapping) and so are invisible, whereas the attacks with $\mathsf{RS} \neq \mathsf{RT}$ (across roles/zones) are visible. Furthermore the bindings of the interpretation variables $\mathit{VulServ}$ and $\mathit{Pol}$ that contain the respective vulnerable service and policy are used in the mapping. All other transitions are mapped to $\epsilon$.

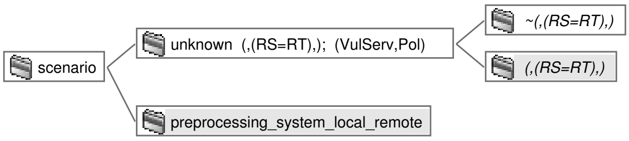

scenario
unknown (,(RS=RT),); (VulServ,Pol)
~(,(RS=RT),)
(,(RS=RT),)
preprocessing_system_local_remote

Figure 34: Mapping for attacks against unknown vulnerabilities that cross zones

The abstract representation computed from that mapping is shown in Figure 35. It gives a clear overview about what kind of zone crossing attacks would be possible in case that new unknown vulnerabilities were exploited. For each assumed vulnerable service it shows the policies that would allow the attack.

A modified definition of the mapping in Figure 34 can be used to inspect, for example, the edge $(\mathit{VulServ} = \mathit{sendmaild} \quad \mathit{Pol} = (\mathit{intern\_host}, \mathit{any\_role}, \mathit{net}))$ in Figure 35. A refined mapping with

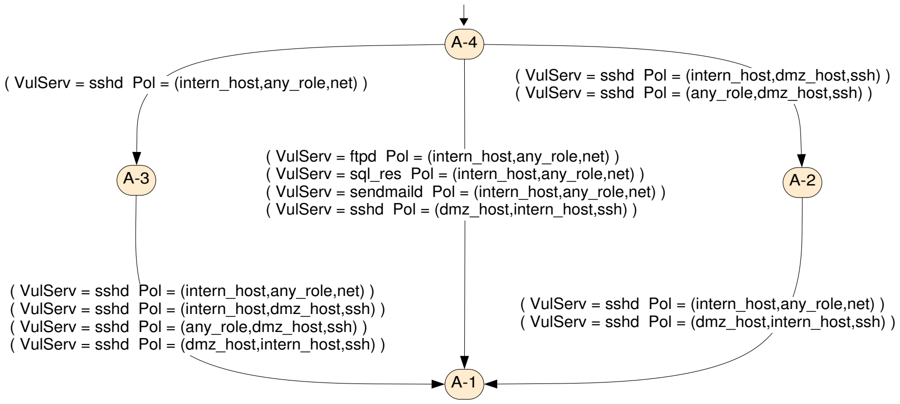

A-4
( VulServ = sshd  Pol = (intern_host,any_role,net) )
( VulServ = sshd  Pol = (intern_host,dmz_host,ssh) )
( VulServ = sshd  Pol = (any_role,dmz_host,ssh) )
A-3
( VulServ = ftpd  Pol = (intern_host,any_role,net) )
( VulServ = sql_res  Pol = (intern_host,any_role,net) )
( VulServ = sendmaild  Pol = (intern_host,any_role,net) )
( VulServ = sshd  Pol = (dmz_host,intern_host,ssh) )
A-2
( VulServ = sshd  Pol = (intern_host,any_role,net) )
( VulServ = sshd  Pol = (intern_host,dmz_host,ssh) )
( VulServ = sshd  Pol = (any_role,dmz_host,ssh) )
( VulServ = sshd  Pol = (dmz_host,intern_host,ssh) )
( VulServ = sshd  Pol = (intern_host,any_role,net) )
( VulServ = sshd  Pol = (dmz_host,intern_host,ssh) )
A-1

Figure 35: Abstract representation of attacks against unknown vulnerabilities

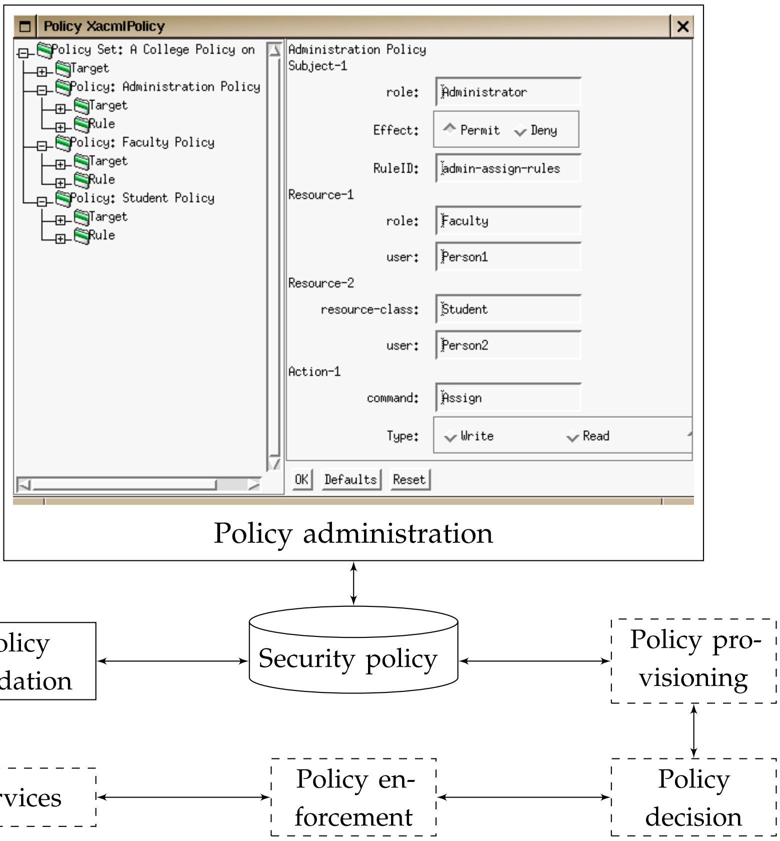
predicate $(\mathsf{RS} \neq \mathsf{RT} \ \wedge \ \mathit{VulServ} = \mathit{sendmaild})$ and visible interpretation variables RS and RT results in an abstract representation with four parallel edges labelled
$(\mathsf{RS} = \mathit{developer\_host} \ \ \mathsf{RT} = \mathit{dmz\_host})$,
$(\mathsf{RS} = \mathit{db\_host} \ \mathsf{RT} = \mathit{dmz\_host})$,
$(\mathsf{RS} = \mathit{intern\_host} \ \mathsf{RT} = \mathit{dmz\_host})$, and
$(\mathsf{RS} = \mathit{management\_host} \ \mathsf{RT} = \mathit{dmz\_host})$
respectively. This shows that if an attacker knows a new exploit for an unknown vulnerability of the product providing the $\mathit{sendmaild}$, then the current policy rule $(\mathit{intern\_host}, \mathit{any\_role}, \mathit{net})$ would allow to use the exploit to cross the four given zones.

Now if the policies are quite restrictive and no new cross role/-zone attacks are found by the reachability analysis, then it can be concluded that the network configuration is resilient with respect to attacks against *one* unknown vulnerability. In the same way resiliency with respect to two or more unknown vulnerabilities can be analysed. Please note that in many cases this will not be possible because of state space explosion problems. However, computation of a subgraph of the attack graph by giving a limitation on the number of edges to be computed is possible and helps to identify problems and to successively restrict the configuration to an acceptable risk level. Another way of mitigating state space explosion problems could be to assign common weaknesses defined by Common Weakness Enumeration (CWE) [Martin & Barnum, 2008; MITRE Corporation, 2013d] to each system instead of using vulnerabilities of products. CWE provides a unified set of software weaknesses, that is, an abstraction of CVE.

## 3.5 SECURITY POLICY VALIDATION

Security policies provide a well-understood and suitable means to administer security issues. However, using policies in distributed environments where applications, services and nodes dynamically join and leave the system raises additional questions.

Policy-based control of ICT networks has the benefit that the controlling units of the system are kept decoupled from the management components and the rule base that governs the decisions. This enables the management and change of the system's behaviour without having to modify the software or the controlled nodes. The system is controlled by policies that specify behaviour rules which are interpreted by decision components (subsequently called Policy Decision Point (PDP)) and are asserted by enforcement components (subsequently called PEP). Hence, if conditions change or new services or applications are added to the system one just adapts the policy rules. Using a central administration component the platform administrator does not have to deal with the multitude of different nodes in the system. Figure 36 shows a policy management approach that has been developed for this purpose by the SicAri project [Rieke & Ebinger, 2008; Peters, 2013].



Policy XacmlPolicy

- Policy Set: A College Policy on
  - Target
  - Policy: Administration Policy
    - Target
    - Rule
  - Policy: Faculty Policy
    - Target
    - Rule
  - Policy: Student Policy
    - Target
    - Rule

Administration Policy

Subject-1

role: Administrator

Effect: Permit Deny

RuleID: admin-assign-rules

Resource-1

role: Faculty

user: Person1

Resource-2

resource-class: Student

user: Person2

Action-1

command: Assign

Type: Write Read

Figure 36: Policy administration and validation in policy architecture

It is an important requirement for holistic policy management to bridge the gap between the informal specification of security policies – *what the security administrator wants to enforce* – and its corresponding machine-readable policy specification – *what the system actually enforces* –. This goal is expressed by the following research question.

RQ8: *Does a policy correctly implement high-level security goals?*

P12 [Peters, Rieke, Rochaeli, Steinemann & Wolf, 2007] suggests the integration of policy validation into an holistic policy architecture. The policy architecture comprises the components of the policy framework and their interactions in order to guarantee that all security relevant processes in the platform are fulfilled according to the underlying security policy.

The task of the policy validation component is to evaluate, whether a policy correctly implements given security goals. To accomplish this, the SHVT (cf. Section 2.2) was extended to accept a subset of Xtensible Access Control Markup Language (XACML) as input and to translate it into transition patterns, which specify the behaviour of APA. Each policy rule is converted into such a transition pattern which then encodes the action that is controlled by that rule. This in turn results in an operational model of the policy system that can be executed in the SHVT. It allows to analyse the policy system's behaviour, to simulate its potential information flow and to verify the wanted security goals. For that purpose the system's reachability graph is computed which spans all possible sequences of transition steps that are allowed by the given policy.

To answer RQ8 it is necessary to analyse a policy with respect to actions that can be executed by a specific subject on a specific object during any course of action. In this case the actions will likely be actions of the system components, that is, the subjects acting in a specific role. An action could be the creation or deletion of a file, the use of some resource, the activation or termination of a process or the communication with other systems or components.

For the analysis it is, firstly, necessary to compute a complete graph of the possible (critical) system behaviour (cf. Section 2.2.1). Secondly, as described in Section 3.3 an abstract representation of the behaviour can be computed and used as basis for the analysis. The abstraction could, for example, blind out all actions from subjects not involved in a critical aspect of the behaviour (see Figure 37).

Further work published in P12 provides an approach for policy provisioning based on the Common Open Policy Service (COPS) protocol. The component covers negotiation and distribution of policies and policy updates, as well as transport of policy decision requests and responses.

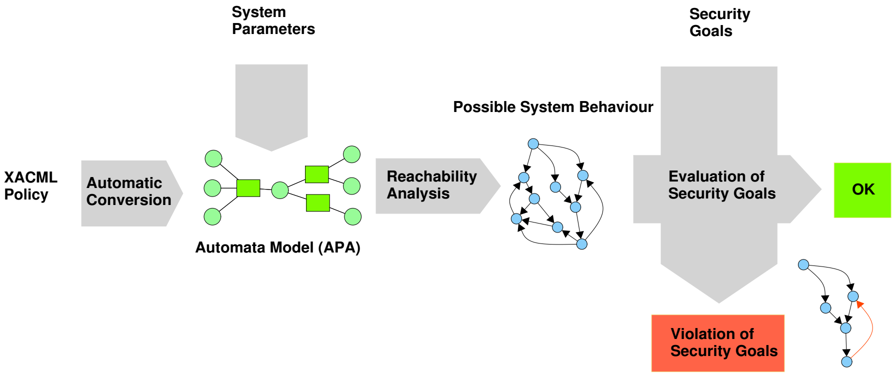


Figure 37: Policy validation

## 3.6 RELATED WORK

The rising complexity of computer network security management has led to the necessity to develop methods to detect errors in network configurations, determine critical network resources, and choose and enforce appropriate security policies with respect to current threats [Kotenko et al., 2011]. There are many different approaches to analytical attack modelling given in the literature. The key elements for security evaluation and suggested architectural solutions, however, are provided by the concepts of *attack trees*, *attack graphs*, *security configuration metrics*, and *administration and validation of security policies*.

### 3.6.1 *Attack trees*

The *attack tree* concept has been introduced by Schneier [1999, 2000]. The notion is related to the preexisting fault/failure tree model that is normally used to identify safety hazards. The method to develop an attack tree is a top down approach with a divide and conquer method to break down an attacker goal in the root of the attack tree to subgoals in the child nodes annotated by logical *and* and *or* connectives. A formal model of attack trees is given by Mauw & Oostdijk [2006].

Several variants of attack trees have been developed. *Defence trees* [Bistarelli et al., 2006, 2012] add a set of attack countermeasures on each leaf node, whereas *attack countermeasure trees* [Roy et al., 2012] support countermeasure at any tree node. The nodes of a *protection tree* [Edge et al., 2007] contains four run-time metrics, namely, probability, cost, impact and risk. *Attack response trees* introduced in Zonouz et al. [2009, 2011] consist of a root node that contains the security property, leaf nodes that denote specific vulnerability exploitations, and consequence nodes representing countermeasure actions. Formalisations of *attack defence trees* were published in Kordy et al. [2011]. They comprise attack nodes that represent attack actions to compromise the system, defence nodes that represent defender actions to protect the system, and additional child nodes representing countermeasures. An attempt to provide a generic notation, namely, *unified parametrizable attack tree* for several variants of such attack trees has been proposed in Wang et al. [2011].

An *attack pattern* is a minimal sets of nodes in an attack tree that achieves the goal at the root node [Barnum & Sethi, 2007]. Common Attack Pattern Enumeration and Classification (CAPEC) [MITRE Corporation, 2013a] is an initiative to collect core sets of attack pattern instances and make them publicly available.

Attack trees have been used, for example, for dark-side scenario analysis in the project EVITA [Fraunhofer SIT, 2011] in order to assess security risks of automotive on-board networks [Ruddle et al., 2009]. The project VIKING [Viking project consortium, 2012] modelled the security risk for networked Supervisory Control And Data Acquisition (SCADA) systems and consequences of successful attacks in terms of monetary loss for the society based on attack trees [Björkman, 2010; Dan et al., 2012].

### 3.6.2 *Attack graphs*

*Attack graphs* have been introduced by Phillips & Swiler [1998]. This concept is closely related to attack trees but attack graphs usually comprise all possible attack paths not just one goal. This analysis takes into account the vulnerabilities of the products installed on the components of a networked SoS, the connectivity of the SoS, and the assumed capabilities of an attacker or group of attackers. This can be seen as a bottom up approach that computes all goals an attacker reach based on a given set of possible exploits. The arcs in the attack graph represent attacks and the nodes represent a stage of attack. The leafs represent states where an attacker can not proceed any further, thus they can be used to find the maximum impact. Loops are possible when an attacker can act but does not gain any advantage from it or even looses attack possibilities.

The network vulnerability modelling part of the framework presented in this chapter is adopted from the approach introduced in P9 and P10. It is similar in design to the approach taken in Phillips & Swiler [1998] and Swiler et al. [2001]. A major contribution of P9 and P10 was the use of abstraction methods to visualise compact presentations of the graph and the inclusion of liveness analysis. The work presented in Ritchey & Ammann [2000], Jha et al. [2002] and Sheyner et al. [2002] uses attack graphs that are computed and analysed based on model checking. Ammann et al. [2002] presented an approach that is focussed on reduction of complexity of the analysis problem by explicit assumptions of monotonicity. The work of Kotenko & Stepashkin [2006] is focussed on security metrics computations and adaptive cooperative defence mechanisms [Kotenko

& Ulanov, 2007]. Noel & Jajodia [2004]; Noel et al. [2005] describe novel attack graph visualisation techniques. Ingols et al. [2006, 2009] developed a tool for high-performance network analysis taking into account modern client-side attacks. Kheir et al. [2010]; Debar et al. [2010] propose a new service dependency model that enables intrusion and response cost evaluation including response collateral damage effects. A formal model of an *attack surface* is provided by Manadhata & Wing [2011]. The attack surface is one indicator of a system's security; the larger the attack surface, the more insecure is the system. Attack modelling and security evaluation in Security Information and Event Management (SIEM) systems is introduced in Kotenko et al. [2012] and Kotenko & Chechulin [2012].

P11 added the capability to represent attacks based on unknown vulnerabilities to the model proposed in this thesis. To the best of the knowledge of the author, this was the first publication approaching this research question, even though, the term *zero-day* for unknown vulnerabilities was not used by the author at that time. The analysis of network security against unknown zero-day attacks is still an active research topic [Ingols et al., 2009; Wang et al., 2010; Kotenko & Chechulin, 2012]. State space explosion problems that have limited the number of unknown vulnerabilities in the model of P11 could be mitigated by using CWE [MITRE Corporation, 2013d] instead of CVE. CWE provides a unified set of software weaknesses Martin & Barnum [2008] that abstracts from specific product vulnerabilities.

An overview on the state of the art in analytical attack and defence modelling is given in Kotenko et al. [2011] and Kordy et al. [2013].

### 3.6.3 *Security configuration metrics*

To model the ICT network, that is, the platforms, installed products, known vulnerabilities of these products and the intrusion detection systems, a data model loosely resembling the formally defined M2D2 information model [Morin et al., 2002] has been used in P9, P10, and P11. Appropriate parts of this model are adopted and supplemented by concepts needed for description of exploits, attacker knowledge and strategy and information for cost benefit analysis. The information in CVE [MITRE Corporation, 2013b; Christey & Martin, 2007] has been used to represent the known vulnerabilities in the model. Information from NVD [NIST Computer Security Resource Center, 2013b] has been used for the vulnerability *range* and *impact type* that indicates the post-attack effects of exploits of CVE vulnerabilities to the respective target. The Open Sourced Vulnerability Database (OSVDB) [Martinet al. , 2013] is an independent and open sourced web-based vulnerability database that provides information similar to CVE and NVD. Recent work on a Cyber-Security Ontology Architecture [Parmelee, 2010] building on Common Platform Enumeration (CPE) [NIST Computer Security Resource Center, 2013c] that provides an open standard for a structured naming scheme for Information Technology (IT) products [Buttner & Ziring, 2009] and Common Configuration Enumeration (CCE) [NIST Computer Security Resource Center, 2013a] could be used alternatively to the M2D2 information model.

The CVSS [FIRST.org, Inc., 2013] provides universal severity ratings for CVE security vulnerabilities [Schiffmann, 2005; Mell et al., 2007; Scarfone & Mell, 2009]. In P9, P10, and P11 the CVSS is used in the model to assess the threat level. Further work on security metrics comprises Jaquith [2007], Herrmann [2007], Jansen [2009], and the Common Weakness Scoring System (CWSS) [MITRE Corporation, 2013c]. An ontology of metrics for security evaluation and decision support in SIEM systems is given in Kotenko et al. [2013].

### 3.6.4 *Administration and validation of security policies*

Approaches to specify and enforce rights for access control in ICT have been researched for many years. To name a few of the most influential results, Bell & LaPadula [1974, 1976], Harrison, Ruzzo & Ullman [1975] and Clark & Wilson [1987] have developed early models for computer system and operating system security. Many recent research papers investigate security policies on its own and abstract from the systems needed to enforce these policies. Most of these activities focus on the examination of specific properties of policies like consistency, freedom of conflicts, information flow implications and effects to system safety. This allows shifting the attention from specifics of computer system towards the analysis of properties that are inherent to the policy itself. A method to enforce rigorous automated network security management using a network access control policy is presented in Guttman & Herzog [2005]. This method is illustrated using examples based on enforcement strategy by distributed packet filtering and confidentiality/authenticity goals enforced by IPsec mechanisms.

Access control policies in SicAri are based on the Role-Based Access Control (RBAC) standard [Ferraiolo et al., 2001]. The general concepts of RBAC [Ferraiolo & Kuhn, 1992] are well-understood and extensively described, for example, in Sandhu [1998]; Ferraiolo et al. [2003]. RBAC is assumed to be policy-neutral, thus it provides a flexible means to deal with arbitrary security policies. SicAri uses a subset of XACML [Moses, 2005] that comprises the most important elements and attributes of the language as its policy specification language. The goal was to reach the expressiveness that allows to handle a well-known XACML example which has been validated in the literature before [Bryans, 2005; Fisler et al., 2005].

XACML supports the RBAC policy model RBAC [Anderson, 2005]. Extensions to this profile that support OrBAC have been proposed by

Haidar et al. [2006]. Both RBAC and OrBAC support the *role* abstraction for subjects. Roles are assigned to subjects and permissions are given to roles. OrBAC extends RBAC by supporting additional abstractions such as *activity* for actions and *views* for an object hierarchy. This allows to express the security policy on abstract entities only and separate it from its implementation [Kalam & Deswarte, 2006]. A formal approach to use an OrBAC model to specify network security policies was presented in Cuppens et al. [2004]. This approach has been adopted in this thesis to model the network security policies in the attack graph analysis framework (cf. P9, P10, P11). Ochsenschläger, Rieke & Velikova [2008] addresses OrBAC policy validation by an example of a policy controlled information flow in a hospital. Ben Mustapha & Debar [2013] presents a service dependency aware policy enforcement framework in order to explore several enforcement possibilities in an attack response decision.

The COPS protocol that has been used in the approach presented in P12 has also been used for policy based Quality of Service (QoS) management in Ponnappan et al. [2002], for policy based ad hoc network management in Phanse [2003], and for distributing and enforcing access control policies to Resource Reservation Protocol (RSVP) aware application servers in Toktar et al. [2004].

Major focus of the combined modelling and analysis framework presented in this thesis is the integration of formal network vulnerability modelling, on the one hand, and network security policy modelling, on the other hand. This aims to help adaptation of a network security policy to a given and possibly changing vulnerability setting. Recent methods for analysis of attack graphs are extended to support analysis of abstract representations of these graphs.

## 3.7 SUMMARY OF RESULTS

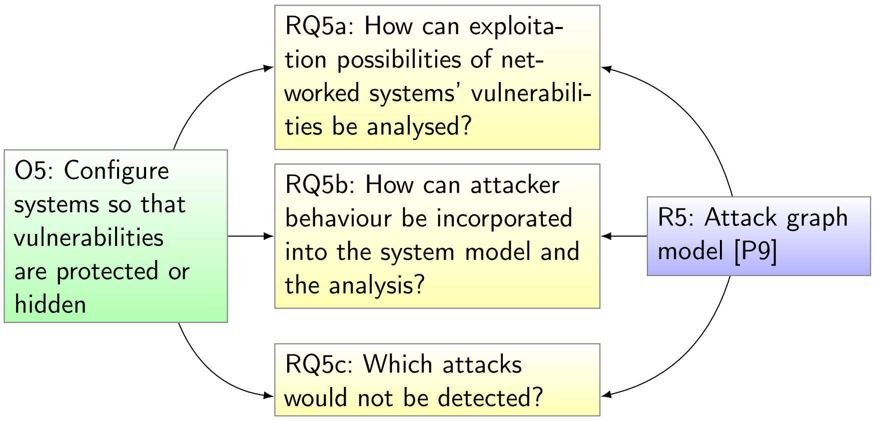
This chapter addresses security of system configurations. It considers specific aspects of *fault forecasting* and *fault tolerance* by analysis of networked system configurations with respect to external exploitability of given vulnerabilities.

With respect to the overall aim of this thesis – to provide a framework for security analysis of system behaviour – this chapter addresses the "Do" activity in the Plan-Do-Study-Act (PDSA) cycle (cf. Figure 4). The general objective of this activity is to analyse the configuration of the implemented plan, verify that the goals are met, and provide data for runtime analysis.

The thesis provides methods and tools to analyse the exposition of vulnerabilities in the software components of a networked system to exploitation by internal or external threats. This allows the security assessment of alternative system configurations and thus to minimise the attack surface of the networked system and mitigate potential impact of successful attacks. Furthermore, methods and tools to validate and deploy security policies are provided.

### 3.7.1 *Attack graph model*

Objective O5 of this thesis was to *configure systems so that vulnerabilities are protected or hidden*. This motivated research questions RQ5a, RQ5b, and RQ5c.


RQ5a: How can exploitation possibilities of networked systems' vulnerabilities be analysed?
O5: Configure systems so that vulnerabilities are protected or hidden
RQ5b: How can attacker behaviour be incorporated into the system model and the analysis?
R5: Attack graph model [P9]
RQ5c: Which attacks would not be detected?

The results published in P9 address these research questions. In the following, an overview of this paper is given.

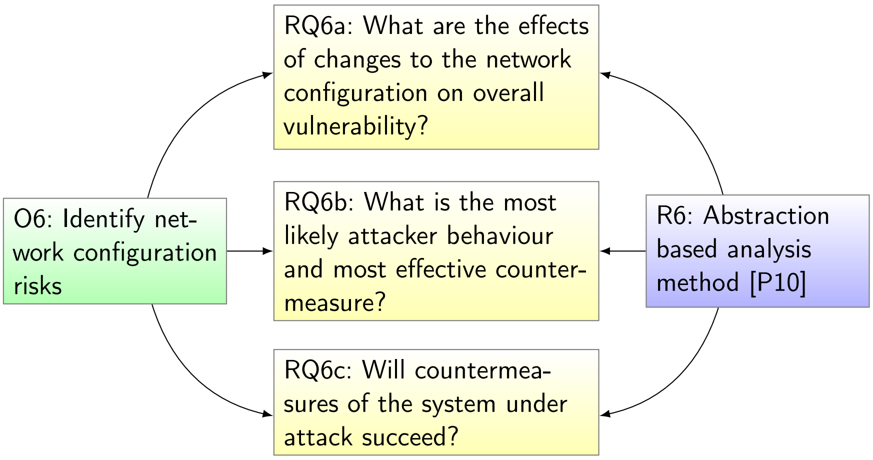
P9. Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration

A core concern of critical infrastructure protection is a careful analysis of what parts of the information infrastructure really need protection and what are the concrete threads as well as an evaluation of appropriate protection measures.

In this paper a methodology and a tool for the development and analysis of operational formal models is presented that addresses these issues in the context of network vulnerability analysis. A graph of all possible attack paths is automatically computed from the model of a government or enterprise network, of vulnerabilities, exploits and an attacker strategy. Based on this graph, security properties are specified and verified, abstractions of the graph are computed to visualise and analyse compacted information focussed on interesting aspects of the behaviour and cost-benefit analysis is performed. Survivability comes into play, when system's countermeasures and the behaviour of vital services it provides are also modelled and effects are analysed.

### 3.7.2 *Abstraction based analysis method*

Objective O6 of this thesis was to *identify network configuration risks*, which led to the research questions RQ6a, RQ6b, and RQ6c.


RQ6a: What are the effects of changes to the network configuration on overall vulnerability?
O6: Identify network configuration risks
RQ6b: What is the most likely attacker behaviour and most effective countermeasure?
R6: Abstraction based analysis method [P10]
RQ6c: Will countermeasures of the system under attack succeed?

The results published in P10 address these research questions. In the following, an overview of this paper is given.

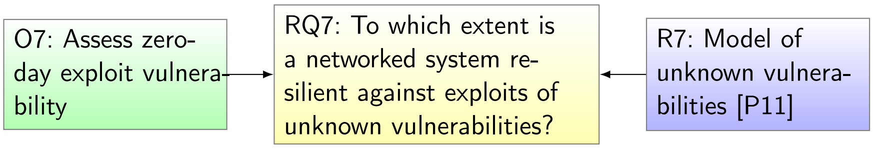
P10. Modelling and Analysing Network Security Policies in a Given Vulnerability Setting

A typical means by which an attacker or his malware try to break into a network is, to use combinations of basic exploits to get more information or more credentials and to capture more hosts step by step. To find out if there is a combination that enables an attacker to reach critical network resources or block essential services, it is required to analyse all possible sequences of basic exploits, so called *attack paths*. Based on such an analysis, it is now possible to find out whether a given security policy successfully blocks attack paths and is robust against changes in the given vulnerability setting.

For this type of security policy analysis, a formal modelling framework is presented that, on the one hand, represents the information system and the security policy, and, on the other hand, a model of attacker capabilities and profile. It is extensible to comprise intrusion detection components and optionally a model of the system's countermeasures. Based on such an operational model, a graph representing all possible attack paths can be automatically computed. Now security properties can be specified and verified on this *attack graph*. If the model is too complex to compute the behaviour, then simulation can be used to validate the effectiveness of a security policy. The impact of changes to security policies can be computed and visualised by finding differences in the attack graphs. A unique feature of the presented approach is, that abstract representations of these graphs can be computed that allow comparison of focussed views on the behaviour of the system. This guides optimal adaptation of the security policy to the given vulnerability setting.

### 3.7.3 *Model of unknown vulnerabilities*

Objective O7 of this thesis was to *assess zero-day exploit vulnerability* of a networked system. The associated research question RQ7 thus investigates network resiliency.



O7: Assess zero-day exploit vulnerability

RQ7: To which extent is a networked system resilient against exploits of unknown vulnerabilities?

R7: Model of unknown vulnerabilities [P11]

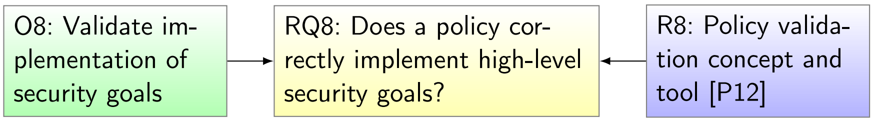
The results published in P11 address this research question. In the following, an overview of this paper is given.

P11. ABSTRACTION-BASED ANALYSIS OF KNOWN AND UNKNOWN VULNERABILITIES OF CRITICAL INFORMATION INFRASTRUCTURES

This journal paper is an extended version of P10. In addition to the results of P10 it provides an approach to analysis of unknown vulnerabilities. In order to analyse resilience of critical information infrastructures against exploits of unknown vulnerabilities, generic vulnerabilities for each installed product and affected service are added to the model. The reachability analysis now considers every possible choice of product, and so all alternatives are evaluated in the attack graph. The impact of changes to security policies or network structure can be visualised by differences in the attack graphs. Results of this analysis support the process of dependable configuration of critical information infrastructures.

### 3.7.4 *Policy validation concept and tool*

Objective O8 of this thesis was to *validate implementation of security goals*. The respective research question RQ8 aims to validate whether a policy correctly implements the primary security goals.



O8: Validate implementation of security goals

RQ8: Does a policy correctly implement high-level security goals?

R8: Policy validation concept and tool [P12]

The results published in P12 address this research question. In the following, an overview of this paper is given.

**P12. A Holistic Approach to Security Policies – Policy Distribution with XACML over COPS**

The potentials of modern information technology can only be exploited, if the underlying infrastructure and the applied applications sufficiently take into account all aspects of IT security. This paper presents the policy architecture of the SicAri project [Rieke & Ebinger, 2008; Peters, 2013] that aims to build a security platform for ubiquitous Internet usage, and gives an overview of the implicitly and explicitly used security mechanisms to enable access control for service oriented applications in distributed environments. The paper introduces the security policy integration concept with a special focus on distribution of security policies within the service infrastructure for transparent policy enforcement. Specifically, extensions to the COPS protocol to transport XACML payload for security policy distribution and policy decision requests/responses are described.

### 3.7.5 *Conclusion*

Fault tolerance denotes the ability to avoid service failures in the presence of faults [Avizienis et al., 2004]. A vulnerability is an internal fault. When a vulnerable component of a system is exposed to access by an attacker its vulnerabilities may be exploited and cause an error. This may possibly cause a subsequent failure of a service that the system provides or it may force the system into a mode providing only reduced functionality. The configuration of the network policy of a system influences the exposition of vulnerable components to external access. Thus the analysis of the network security policy and optimisation of the configuration based on the results of this analysis can improve the fault tolerance of a system as a whole.

In summary, the work presented in this chapter brings together, (1) attack graph computation technology, (2) state-of-the-art policy modelling, and, (3) formal methods for analysis and computation of abstract representations of the system behaviour. The aim is, to guide a systematic evaluation and assist the persons in charge with optimising adaptation of the network security policy to an ever-changing vulnerability setting and so to improve the configuration of the IT infrastructure. The most distinctive feature of this approach is the ability to compute abstract representations of the complex graphs that enable comparison of focussed views on the behaviour of the system. In addition, the approach enables the analysis of the resilience of IT infrastructures against exploits of unknown vulnerabilities by zero day attacks.

In the work presented in Chapter 4 attack graph analysis can be utilised in two ways. Firstly, vulnerability metrics resulting from attack graph analysis can be used to enrich the context information used by security analysis at runtime, and secondly, attack paths like the one depicted in Figure 29 can be used for the definition of the security model.

Further noteworthy work of the author of this thesis with respect to the work presented in this chapter comprises peer-reviewed publications regarding policy validation for access control of electronic health records [Ochsenschläger et al., 2008] and security policies in mobile business intelligence infrastructures [Kuntze et al., 2010]. Invited talks have been given to the IFIP working group on *Dependable Computing and Fault Tolerance* [Rieke, 2007a], at the J.W. Goethe university in Frankfurt [Rieke, 2007b], and at the first FORWARD workshop in Göteborg [Rieke, 2008b].

# 4 PREDICTIVE SECURITY ANALYSIS AT RUNTIME

> *Another challenge is the notion of* concept drift, *i.e., processes change while being observed. Existing process discovery approaches do not take such changes into account. It is interesting to detect when processes change and to visualize such changes.*
>
> — Wil M. P. van der Aalst, Process Mining [van der Aalst, 2011]

AIM OF THIS CHAPTER. Security analysis is growing in complexity with the increase in functionality, connectivity, and dynamics of current electronic business processes. Technical processes within critical infrastructures intrinsically linked to the business processes further complicate the task to provide situational security awareness. To tackle this complexity, the application of models is becoming standard practice. However, model-based support for security analysis is not only needed in pre-operational phases but also during process execution, in order to provide situational security awareness at runtime. Therefore, this chapter presents an approach to support model-based evaluation of the security status of process instances. In particular, challenges with respect to the assessment whether instances of processes violate security policies or might violate them in the near future are addressed. The approach is based on operational formal models derived from process specifications and security and compliance models derived from high-level security and safety goals. An integration concept for a holistic security strategy management is proposed and the applicability of the approach is exemplified utilising processes from several industrial scenarios.

This chapter is based on the work published in P13, P14, P15, P16, P17, P18, and P19 (cf. Table 3).

## 4.1 INTRODUCTION

The Internet today provides the environment for novel applications and processes which may evolve way beyond pre-planned scope and purpose. Frequent changes to *business process* models have to be applied to address changing business needs [Tallon, 2008]. Some workflow management systems already facilitate the necessary adaptation of business process models at runtime [Döhring et al., 2011]. *Technical processes* within critical infrastructures also have to cope with these developments. For example, many Supervisory Control And

Data Acquisition (SCADA) systems that include functions for the remote measurement and control of process devices already include computerised models of the supervised process [Björkman, 2010]. Geographically dispersed real and virtual infrastructures, services and resources are elementary components of such processes within large-scale, massively interconnected Systems of Systems (SoS). This evolving environment, however, also enables new threats and scales up the risks of financial and also physical impact. This situation specifically leads to challenges with respect to the assessment whether instances of processes violate security policies or might violate them in the near future.

The aim of this thesis with respect to predictive security analysis at runtime is, to support model-based evaluation of the current security status of process instances as well as to allow for decision support by analysing close-future process states. This chapter now introduces a novel model-based approach called Predictive Security Analysis at Runtime (PSA@R) that fits to the following four objectives of this thesis (cf. Figure 3).

O9: Develop a security monitoring approach.

O10: Validate security compliance at runtime.

O11: Integrate security management.

O12: Provide evidence for usability in large scale industrial scenarios.

Addressing objective O9, PSA@R observes the operation of a system by analysing events received from this system. Events from process instances executed by the observed system are filtered for their relevance to the analysis and then mapped to the model of the originating process instance. Deviations from the expected behaviour trigger *uncertainty management* and possibly alerts.

With respect to objective O10, PSA@R allows for specification, on-the-fly check, and visualisation of a security model. Because possible close-future process actions can be predicted, based on the operational process specification and the current process state as reflected in the model, predictive security alerts can be computed.

Addressing objective O11, an extensible meta model is presented that spans all parts of the security monitoring and decision support process, namely: (i) detecting threatening events; (ii) putting them in context of the current system state; (iii) explaining their potential impact with respect to some security- or compliance model; and (iv) taking appropriate actions.

Addressing objective O12, requirements to adapt PSA@R to specific industrial scenarios are collected and analysed. A prototype that implements the PSA@R approach, its embedding into the runtime environment, and some observations considering the runtime behaviour and performance of the prototype are provided.

This chapter is organised as follows. Section 4.2 describes the approach for security analysis of processes at runtime, the synchronisation with the running process, and the prediction of possible continuations of process instances. Section 4.3 presents the security model applied at runtime to identify security relevant states and exemplifies generated security alerts. Section 4.4 describes the integration of PSA@R into a systemic approach for security strategy measurement and management. Section 4.5 summarises the requirements to adapt PSA@R to specific industrial scenarios, describes the prototype and application results. Section 4.6 reviews related work to the approach. Finally, this chapter ends with a summary of the results in Section 4.7.

## 4.2 process monitoring and uncertainty management

With respect to Objective O9 of this thesis, namely, to develop a security monitoring approach, the following research questions have been raised.

RQ9a: *How can operational models reflect the state of observed systems and thus capture abstractions of runtime behaviour?*

RQ9b: *How can operational process models be used for early detection of and reaction to deviations of process execution from its specification?*

To answer these research questions, the PSA@R approach has been developed by the author of this thesis. PSA@R was first introduced in P13 [Rieke & Stoynova, 2010]. In PSA@R the operation of a system or a SoS is observed by analysing events received from this system. PSA@R is not executed by this observed system but rather by an observing and reacting system such as a Security Information and Event Management (SIEM) system. It is assumed that the purpose of the observed system is given by technical, organisational, and business processes and that the intended behaviour can be specified by process models. The behaviour of the system is then a composition of the behaviours of the running processes.

The PSA@R approach assumes that events from the observed system are first filtered according to their relevance to the analysis and then mapped to the operational formal model of the originating process instance. Now the possible close-future process actions can be predicted, based on the operational process specification and the current process state as reflected in the model. In the *anomaly detection phase* PSA@R identifies deviations from the normal characteristics based on the values from a transaction monitor and generates alerts. *Uncertainty management* supports semi-automatic adaptation of process models at runtime according to the context conditions. Uncertainty situations can occur during synchronisation of the state of a running process instance with the state of the model, if the process model is not accurate enough or outdated, or when unknown events are received or expected events are missing.

### 4.2.1 *Process model*

PSA@R is based on a formal process model given by an Asynchronous Product Automaton (APA) representation (cf. Section 2.2.1) that is utilised to reflect the current state of the system. This model provides the basis for the prediction of close-future actions. However, PSA@R does not depend on a specific formal method chosen for model representation. The only requirement is, that it must allow to compute the possible process behaviour from the process model. For example, Petri nets [Petri, 1962] also meet this requirement (cf. Rieke et al. [2012]). Formally, the behaviour of an operational APA model of a business process is described by a Reachability Graph (RG) (cf. Definition 3 in Section 2.2.1).

**Example 12.** *A process specification provides the control flow structure of a process as a sequence of* events *and* functions. *In an APA model that is derived from a process specification, the set of possible output events of a process function can be used as the alphabet of the elementary automaton representing the function [Eichler & Rieke, 2011]. So the interpretation* $i$ *is the output event. An example for a state transition is*

$$(p, (\mathit{transfer}, \mathit{event} = '\mathit{critical}'), q).$$

*The parameters of this state transition are the state* $p$*, the tuple composed of the elementary automaton transfer and its interpretation event* $= '\mathit{critical}'$*, and the follow-up state* $q$.

### 4.2.2 *Event model*

A stream of events characterises one specific execution trace of the observed system. This trace is a shuffle [Jantzen, 1985; Björklund & Bojanczyk, 2007] of the traces of the executed process instances. The event model determines the internal mapping for the runtime events defined by an event schema. To reduce the complexity only data required for the analysis or in generated alarms should be used in the model.

Formally, it is assumed that an event represents a letter of the alphabet that denotes the possible actions in the system. Different formal models of the same system are partially ordered with respect to different levels of abstraction (cf. Definition 4 in Section 2.2.3).

**Definition 10** (process instance projection)**.** *Let* $P$ *denote a finite set of process instances* $i$ *of some process with* $i \in P$ *and let* $\Sigma_i$ *denote pairwise* *disjoint copies of* $\Sigma$. *The elements of* $\Sigma_i$ *are denoted by* $e_i$ *and* $\Sigma_P := \dot{\bigcup}_{i \in P} \Sigma_i$. *The index* $i$ *describes the bijection* $e \leftrightarrow e_i$ *for* $e \in \Sigma$ *and* $e_i \in \Sigma_i$. *Now the projection* $\pi$ *identifies events from a specific process instance* $i$.

*For* $i \in P$, *let* $\pi_i^P : \Sigma_P^* \to \Sigma^*$ *with*

$$\pi_i^P(e_r) = \begin{cases} e \mid & e_r \in \Sigma_i \\ \varepsilon \mid & e_r \in \Sigma_P \setminus \Sigma_i \end{cases} .$$

This is similar to the notion of a *correlation condition* [Motahari-Nezhad et al., 2011] that defines which sets of events in the service log belong to the same instance of a process.

**Remark 2.** *For effective use of* PSA@R *it is assumed that a process instance projection is possible for each event. In many applications, a process instance identification is directly available as an attribute of the event. Sometimes a set of attributes identifies the process instance. However, the assumption about pairwise disjoint alphabets is not always valid.*

*This aspect contributes to the general requirement: Systems and applications need to be* designed for security assessment at runtime.

If the event data contain redundant or irrelevant attributes, a proper subset of attributes for use in model construction has to be selected. In order to avoid state space explosion problems, the *coarsest* abstraction that still contains all security relevant information should be used.

**Example 13.** *Let us assume that* $\Sigma$ *is the alphabet of events from the measured system and for a given event* $e$ *the term* $\#(e)$ *denotes the value of an attribute involved in a transaction.*

*Let* $h_2, h_3 : \Sigma^* \to \{'high', 'medium', 'low'\}^*$ *the homomorphisms given by*

$$h_2(e) = \begin{cases} 'high' \mid & 10^5 < \#(e) \\ 'low' \mid & \#(e) \leqslant 10^5 \end{cases}$$

$$h_3(e) = \begin{cases} 'high' \mid & 10^5 < \#(e) \\ 'medium' \mid & 10^3 < \#(e) \leqslant 10^5 \\ 'low' \mid & \#(e) \leqslant 10^3 \end{cases} .$$

*Then* $h_3$ *and* $h_2$ *can be used to differentiate process control flow with respect to events with different attribute values.* $h_3$ *is finer than* $h_2$ *because* $\nu : \{'high', 'medium', 'low'\}^* \to \{'high', 'low'\}^*$ *exists.*

At runtime, the current state of the process behaviour model of the process instance $i$ is synchronised with the running process using the projection of the measured events to the respective state transitions $(p, (e, i), q)$ of the RG.

Using this approach, operational models can reflect the state of observed systems and provide an abstract view of their runtime behaviour, thus answering research question RQ9a.

### 4.2.3 *Prediction of close-future process actions*

In order to answer research question RQ9b, PSA@R uses the RG to predict the close-future behaviour of the process instance. A subgraph of the RG starting with the current state of the process instance can always be computed on-the-fly based on the formal process model. The *prediction depth* is the depth of this subgraph starting from the current state.

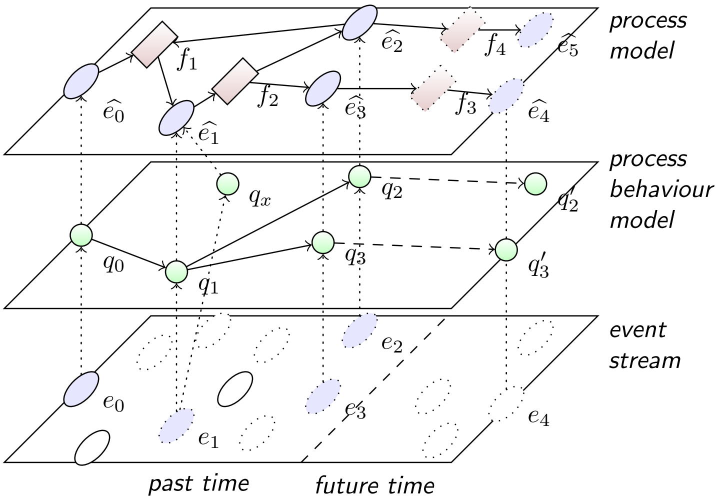


Figure 38: Predict close-future process behaviour

Figure 38 illustrates the approach taken for the prediction of close-future process behaviour. The ellipses in the event stream pane denote the observed events, whereby the filled ellipses $e_0$, $e_1$, $e_2$, and $e_3$ denote the events that belong to the specific process instance $i$, i.e., $e_0, e_1, e_2, e_3 \in \Sigma_i$. The ellipses in the process model pane denote abstract events with respect to an abstraction $h$ used in the event model, e.g., $\widehat{e_1} = h(\pi_i^P(e_1))$. The dotted arrows denote this mapping. The rectangles in the process specification pane denote the process functions and the solid lines denote the transitions. The dashed arrows in the process behaviour model (subgraph of the RG) denote the predicted process behaviour.

**Example 14.** *If in Figure 38 the function* $f_2$ *is modelled by the elementary automaton transfer and* $\widehat{e_3} = h(\pi_i^P(e_3)) =$ *'high' and the depicted process instance* $i$ *is in the state* $q_1$ *and the event* $e_3$ *is received, then the transition* $(q_1, (transfer, event = \text{'high'}), q_3)$ *will match the current situation. In order to predict the behaviour that is following* $q_3$*, the functions* $f_3$ *in the process specification can be used to compute* $q_3'$ *in the* RG*. Therefore, an event* $e_4$ *with* $\widehat{e_4} = h(\pi_i^P(e_4))$ *is predicted.*

### 4.2.4 *Observing system operation*

The control flow of model learning and uncertainty reasoning in PSA@R has been introduced in P18. Based on this, Figure 39 depicts the current algorithm used by PSA@R for semi-automatic adaptation of the *de-jure* process model based on *de-facto* measured behaviour at runtime, and detection of anomalies such as unknown, unexpected and missing events. Step 5 "Adjust process model" is done semi-automatically and requires the user's involvement during runtime; all other actions are performed automatically. Semi-automatic adjustment of the process model is particularly useful in the initial learning phase. It can be used to learn normal behaviour pattern with regard to the transaction characteristics by processing an event log without malicious content.

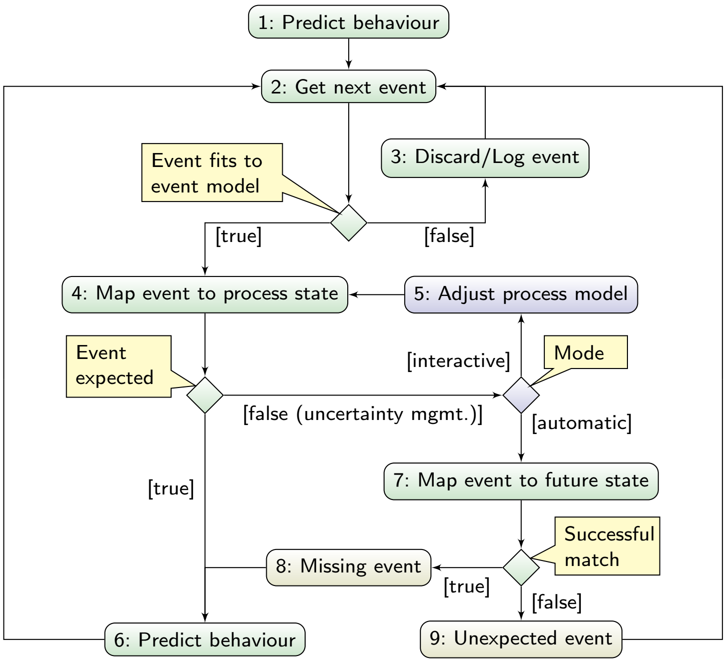


Figure 39: Uncertainty management algorithm

Let $\mathrm{RG}_P(q,d)$ denote the possible behaviour $B$ of the process $P$ given by a RG starting with initial state $q$ and prediction depth $d$, that is, the set of all possible coherent sequences of state transitions of length less or equal $d$ starting at $q$. It is further assumed that $d > 0$.

STEP 1: PREDICT BEHAVIOUR The RG is computed from the initial state $q_0$ of the process model. Let $q := q_0$ and $B := \mathrm{RG}_P(q,d)$. Continue with step 2.

STEP 2: GET NEXT EVENT Read next event $e$ from the observed system's event stream. If $e$ fits to the event model, that is, $e \in \Sigma_P$ (cf. Definition 10), then continue with step 4 else step 3.

STEP 3: DISCARD/LOG EVENT The event $e$ is *unknown*, that is, it does not fit to the event schema used for the mapping. Depending on the audit requirements, the event will be discarded or logged. Continue with step 2.

STEP 4: MAP EVENT TO PROCESS STATE The event $e$ is *expected* for process instance $i$, iff there is a transition

$$(q, (, \mathit{event} = h(\pi_i^P(e))), q')$$

in $RG_P(q, 1)$.

**Example 15.** *Figure 40 shows a situation where an event $e_x$ is received. It has already passed the check whether it fits to the event model but it is not part of the process behaviour in scope of the analysis.*

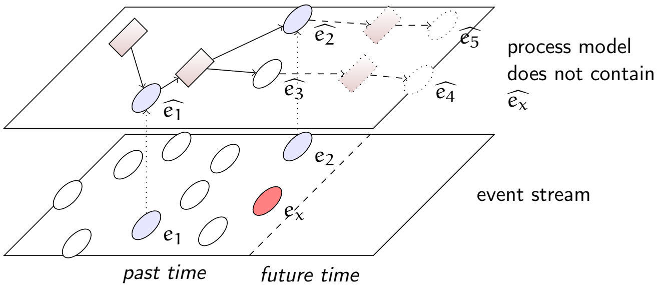


Figure 40: Event not expected in *de-jure* process model

If the event $e$ is expected, then continue with step 6. If the event $e$ is not expected, then in interactive mode continue with step 5, else continue with step 7.

STEP 5: ADJUST PROCESS MODEL The user is expected to change the process specification, so that the event $e$ fits to the possible behaviour in the current state $q$.

**Example 16.** *Figure 41 shows a possible adjustment of the process model from Figure 40. In this case, the user has decided to insert a new event $\widehat{e_x} = h(\pi_i^P(e_x))$ to the process specification along with a connecting edge from the current function. This constitutes a* belief change *with respect to the* de-jure *process model.*

Continue with step 4.

STEP 6: PREDICT BEHAVIOUR Let $q := q'$ and $B := RG_P(q', d)$, where $(q, (, \mathit{event} = h(\pi_i^P(e))), q')$ in $RG_P(q, 1)$. Continue with step 2.

STEP 7: MAP EVENT TO FUTURE STATE A mapping to a future process state means to find a transition $(p, (, \mathit{event} = h(\pi_i^P(e))), p')$ in $B$. If this is successful, then it is assumed that one or more

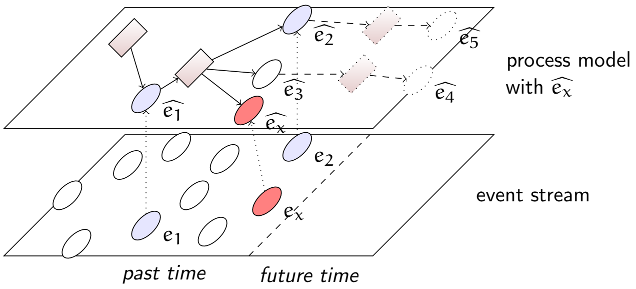


Figure 41: Adapt *de-jure* process model to *de-facto* behaviour

events have been missed and the current process state is adjusted, that is, $q := p$. If the mapping has been successful, then continue with step 8, else continue with step 9.

**Example 17.** *Figure 42 depicts a situation where an event* $e_4$ *is received but not expected in this state of the process. However, an abstract event* $\widehat{e_4} = h(\pi_i^P(e_4))$ *is part of a possible continuation of the process. Thus, it is assumed that some event* $e$ *with* $\widehat{e_3} = h(\pi_i^P(e))$ *has been missed.*

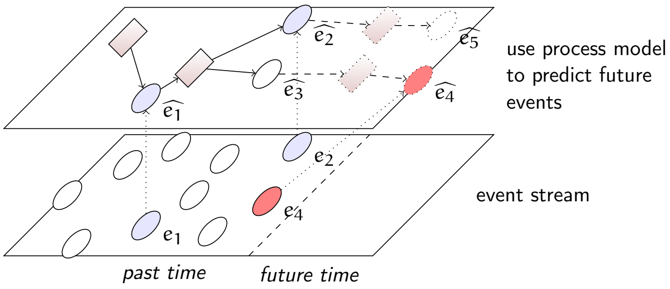


Figure 42: Map event to future state

**Remark 3.** *In order to cope with the case of major deviations of observed behaviour from the behaviour model it can be useful to add another step to the algorithm, namely, to search for the observed event* $e$ *in the process model and not only in the close-future behaviour model. This could reduce the number of alerts. This step is not added here because it is assumed that it is necessary to adjust the process model in such cases.*

STEP 8: UNEXPECTED EVENT An *unexpected event alert* is raised; continue with step 2.

STEP 9: MISSING EVENT A *missing event alert* is raised; continue with step 6.

PSA@R uses this algorithm utilising operational process models for early detection and reaction to deviations of processes' execution from specifications, which provides a solution for research question RQ9b.

## 4.3 security compliance at runtime

This section extends the idea of a model-based *observing system* presented above to a model-based *judgemental system*, thus addressing the following research question.

RQ10: *How can security analysis at runtime exploit process models to identify current and close-future violations of security requirements?*

While the observing system provides situational awareness, the judgemental system, in addition, makes a judgement whether the activity or inactivity of the observed system constitutes a failure [Randell, 2003]. In this context, a failure is a violation of a given security or dependability requirement. This judgement is based on a *security model* applied at runtime to identify security relevant states. In order to provide this novel capability, P14 [Eichler & Rieke, 2011] introduced an approach for the validation of the actual security status of business process instances. P19 [Rieke, Repp, Zhdanova & Eichler, 2014] substantially refined this concept by detailing the monitoring formalism, implementation, evaluation, and context of the approach. The security requirements to be satisfied during process execution must be derived systematically (cf. Section 2.4) and formally specified in terms of specific *monitor automata*. Monitor automata have been introduced in P19 to specify the security requirements graphically.

Figure 43 illustrates all steps of the PSA@R approach. In addition to the nine steps described in the previous section, the following steps enable the runtime assessment of security critical behaviour.

step 10: analyse security related states For runtime identification of security critical states and the prediction of possible failures in the near future PSA@R proposes to use on-the-fly checks of predicates that express the required security properties in terms of state transitions in the current or predicted behaviour. Basically, predicates annotated at the edges of a monitor automaton are applied to state transitions of the RG (cf. Section 2.2).

**Example 18.** *The predicate*

$$(,(,\mathit{event} = \text{'critical\_temperature'}),)$$

*is* true *with respect to a state transition* $(p_i,(e_j,i_k),q_l)$*, if the constant* 'critical_temperature' *is bound to the interpretation variable event of the interpretation* $i_k$*. No condition for the predecessor and successor state* $p_i$*,* $q_l$ *and the elementary automaton* $e_j$ *is given in this example.*

A *failure is detected* if the predicate matches the current state transition and the state reached in the monitor automaton is marked

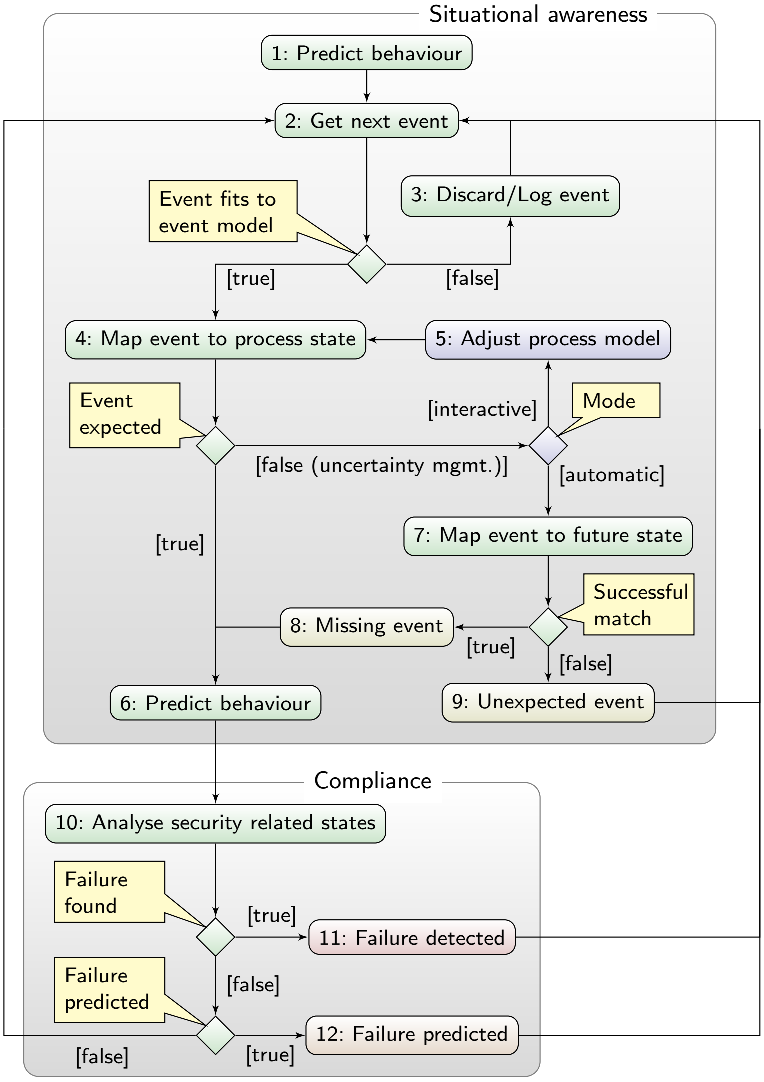


Figure 43: Predictive security analysis at runtime

as *critical state*. A *failure is predicted* if the predicate matches the current state transition and some state of the monitor automaton that is reachable by a path in the predicted behaviour (within the prediction depth) is marked as *critical state*.

STEP 11: FAILURE DETECTED A *security alert* is raised if a security critical situation has been detected. These alerts are mapped to corresponding events and fed into the runtime environment for delivery to decision support and reaction systems.

STEP 12: FAILURE PREDICTED A *predictive alert* is raised when the current situation might escalate to a security critical situation in the close future.

Examples for the application of this analysis of security related states are given in P14 and P19 (cf. Section 4.5.3).

## 4.4 TOOL ARCHITECTURE AND INTEGRATION APPROACH

Based on P19 [Rieke, Repp, Zhdanova & Eichler, 2014], this section describes the architecture of a tool called Predictive Security Analyser (PSA) that implements the PSA@R approach. Furthermore, the integration of PSA@R into a systemic approach for security strategy management that has been published in P19 [Rieke, Repp, Zhdanova & Eichler, 2014] is outlined. This work contributes to answer the following research question.

RQ11: *How can security analysis at runtime be integrated in a security management strategy?*

To gain experience with different modelling strategies in PSA@R with respect to applicability and performance in industrial scenarios, a prototype PSA has been implemented by Fraunhofer SIT headed by the author of this thesis. The PSA supports the complete life-cycle of security analysis at runtime from formal process specification to exhaustive validation, including visualisation and inspection of the computed RGs and monitor automata.

### 4.4.1 *The Predictive Security Analyser (PSA) prototype*

Figure 44 shows the architecture of the PSA consisting of two main parts: the *PSA Modeller* that provides functionality for process formalisation and the *PSA Core* that performs process security analysis.

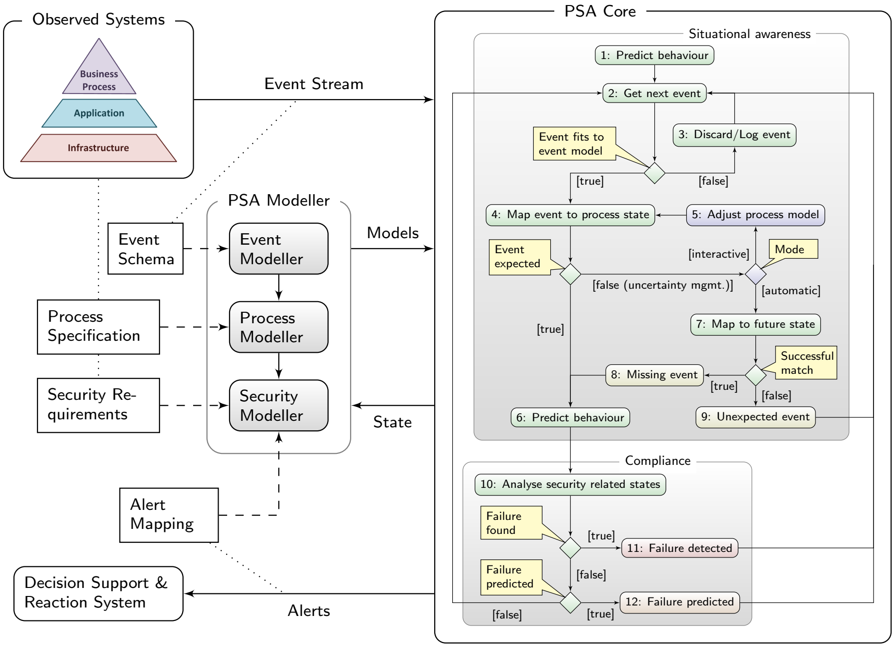


Figure 44: Architecture of the predictive security analyser

The PSA Modeller components support the configuration and adaptation of the PSA to the observed system, the runtime environment and the security requirements.

The *Event Modeller* supports the creation of an event abstraction and a mapping of events to the corresponding process instance. The event model is derived from given event schemata (cf. Figure 48) for interpretation of runtime events.

The *Process Modeller* allows to formalise process specifications based on given Event-driven Process Chain (EPC) or Business Process Execution Language (BPEL) specifications. It also supports the import of Petri Net Markup Language (PNML) process specifications [Weber & Kindler, 2003] from process discovery tools, for example, from the ProM tool [van der Aalst et al., 2009; Verbeek et al., 2011].

The *Security Modeller* provides a graphical interface for the specification of security properties that an observed process must fulfil (a security model in form of monitor automata) (cf. Figure 49a).

When the PSA service is started, the models and their initial configurations (e.g., the initial state of a process model) are compiled and loaded into the PSA Core. During the monitoring and analysis stage the PSA Core components execute the PSA@R method depicted in Figure 43. In addition to the behaviour described in Section 4.2 and Section 4.3 for the steps 8 (unexpected event), 9 (missing event), 11 (failure detected), and 12 (failure predicted), a mapping to external alerts according to the needs of the runtime environment and the decision support and reaction system is provided, for example, messages in Intrusion Detection Message Exchange Format (IDMEF) format.

If a legitimate event does not comply with the model (see step 5 in Figure 43), the PSA supports an adjustment of the model on-the-fly within the process modeller utilising backward references from the compiled process model. Backward references within the compiled security model allow to visualise the current security state within the Security Modeller at runtime.

The implementation language of the PSA is Common LISP [Steele, 1990]. Extensive technical information about the PSA (e.g., technical requirements) is provided in Repp & Rieke [2013].

### 4.4.2 *Integration into security management architecture*

Within the project MASSIF, the author of this thesis has been involved in the definition of an overall architecture of the advanced SIEM system documented in Verissimo et al. [2012]. In this system, the PSA is loosely coupled with other tools developed for the MASSIF system.

In order to improve the semantic integration of the tools used in a security management system, in P15 [Rieke, Schütte & Hutchison, 2012] an extensible model has been proposed that spans all parts of the security monitoring and decision support process, namely: (i) detecting threatening events; (ii) putting them in context of the current system state; (iii) explaining their potential impact with respect to some security- or compliance model; and (iv) taking appropriate actions.

The operational aspects of this concept are described by a Security Strategy Meta Model (SSMM) [Schütte, Rieke & Winkelvos, 2012] that describes the control flow at runtime, independent from the underlying event description language. A specific rule from a Security Strategy Model (SSM) that adheres to the SSMM is called Security Directive (SD). A distinguished Security Strategy Component (SSC) controls the execution of the SD. It can execute a SD or parts of it directly or delegate workload to a specialised Security Strategy Processing Component (SSPC). Depending on the outcome of the analysis of these components, other components that implement decision support and enforcement will be triggered. The proposed SSMM together with the framework of SSPC could be used as a core of a technology platform for an integrated concept for governance, risk and compliance [Racz et al., 2010]. Furthermore, the proposed approach is considered to be applicable within the design of a cyber attack information system [Skopik et al., 2012], which uses collaborative detection and response mechanisms for high-level situational awareness and coordination of local incident response.

A mapping of the SSMM to the components of a proposed monitoring infrastructure enables the inclusion of existing engines, which need not know about the overall security strategy but only receive specific tasks in their respective language.

Conceptually, the implementation of the processing of the SSMM is composed of SSPC. The main components and some optional components of the proposed system architecture are illustrated in Figure 45. A distinguished *Security Strategy Component* controls the execution of the SD. It can execute a SD, or parts of it, directly or delegate the workload to specialised components. The Security Strategy Component initially gets the SSM from the *Security Information Modeller*. It parses the SD of the SSM, identifies the responsible SSPC for each subtask, and distributes a respective configuration to the relevant SSPC. The Complex Event Processing (CEP) engine normally processes the : *on* part of the SD. The security monitoring probes, which are described at an abstract level in the SSM, have to be compiled to the configuration language of the actual CEP engine, if an engine specific specification is not given in the : *on* part of the SD. Optionally, the events could be processed directly. Furthermore, other event processing components such as intrusion visualisation could be triggered. The : *if* part of the SD can be processed by several different components, responsible for different aspects of the domain or several domains. One component, which will be needed in most implementations, is that responsible for the provisioning of the network state information. Other components

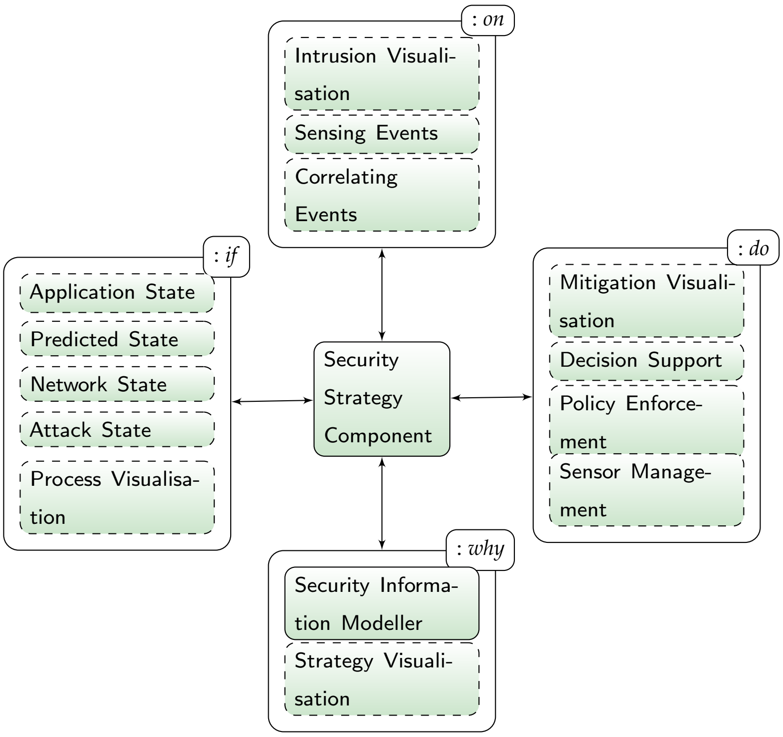


Figure 45: Conceptual components of the framework

could, e.g., provide cyber-physical models, workflow specifications, business process information or process visualisation. Thus, : *if* components such as AMSEC and PSA provide situational awareness with regard to network state, attack state, and application state.

Depending on how the : *if* condition evaluates, the respective : *do* components will be triggered. These components can implement, e.g., *simulative mitigation visualisation*, *decision support*, *policy enforcement* or *sensor management*. A sensor management component can control the configuration of sensors in a monitored system, e.g., the (de-) activation and the adaptation of the sampling rate to an optimal level [Baumgärtner et al., 2012].

A *security information modeller component* is responsible for maintaining the security strategy and a *strategy visualisation component* can help to assist in the : *why* determination.

## 4.5 APPLICABILITY AND PERFORMANCE

In order to evaluate the PSA@R approach in practice, requirements from several industrial scenarios have been collected and used as guidelines for the refinement of the PSA@R methods and the development of the PSA prototype. Results of this requirements analysis have been published in P16 [Prieto, Diaz, Romano, Rieke & Achemlal, 2012] and P17 [Rieke, Coppolino, Hutchison, Prieto & Gaber, 2012].

Besides issues like dependability, redundancy and fault tolerance, in particular, this requirements analysis revealed a lack of capability to model incidents at an abstract level. The following guideline concerning advanced security services has been identified to be particularly relevant for PSA@R.

> "**Predictive security monitoring.** Predictive security monitoring allows to counter negative future actions, proactively. There is a crucial demand for early warning capabilities. Moreover, the limitations with regards to the Managed Enterprise Service point to the fact that dealing with unknown or unpredictable behaviour patterns is not sufficient in current SIEM solutions. "
>
> — P17 [Rieke, Coppolino, Hutchison, Prieto & Gaber, 2012]

This requirement is precisely addressed by the PSA@R approach provided by this thesis.

### 4.5.1 *Adaptation and evaluation in industrial scenarios*

The PSA prototype has been used to answer the following research questions.

RQ12a: *Can the developed methods and tools be successfully adapted to large scale industrial scenarios?*

RQ12b: *What are the performance effects of the number of events, processes, security requirements, predicted steps, and of event abstraction?*

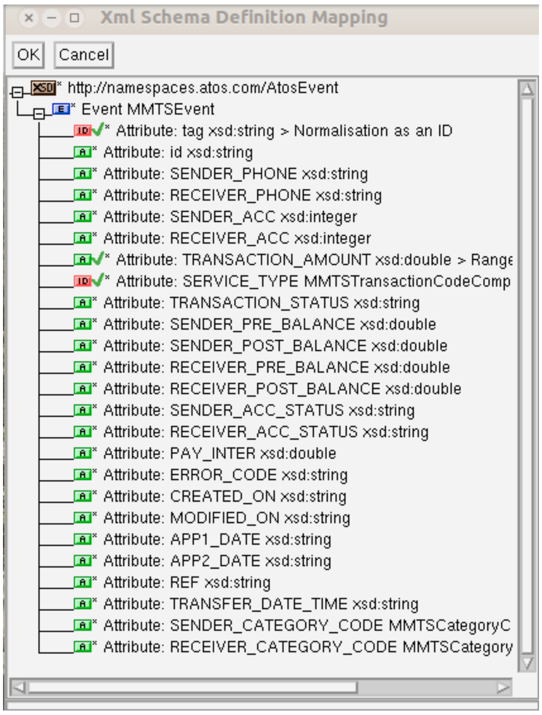
An early version of the PSA has been applied to an Internet of Things (IoT) scenario in the project Alliance Digital Product Flow (ADiWa) [ADiWa Konsortium, 2012]. The results of this first application are documented in P14 [Eichler & Rieke, 2011]. More recently, the PSA has been applied in the European research project MAnagement of Security information and events in Service InFrastructures (MASSIF) [Rieke et al., 2012] to check security requirements in four industrial domains: (i) the management of the Olympic Games IT infrastructure (OOGG) [Vianello et al., 2013], (ii) a mobile phone based Mobile Money Transfer Service (MMTS) [Gaber et al., 2013], facing high-level threats such as money laundering, (iii) Managed Enterprise Service Infrastructures (MESI), and (iv) a Critical Infrastructure Process Control (CIPC) system [Romano et al., 2012].

Results with respect to research questions RQ12a and RQ12b will now be given in more detail for two of these scenarios, namely, MMTS and CIPC.

### 4.5.2 *Adaptation to mobile money transfer scenario*

The MMTS application mainly demonstrates the event model and process model adaptation and the application of process monitoring and uncertainty management as described in Section 4.2 (cf. Figure 39).

In the initial learning phase, the normal behaviour pattern with regard to the transaction characteristics has been learnt by processing an event log without malicious content.



```
XSD* http://namespaces.atos.com/AtosEvent
  E* Event MMTSEvent
    ID ✓* Attribute: tag xsd:string > Normalisation as an ID
    A* Attribute: id xsd:string
    A* Attribute: SENDER_PHONE xsd:string
    A* Attribute: RECEIVER_PHONE xsd:string
    A* Attribute: SENDER_ACC xsd:integer
    A* Attribute: RECEIVER_ACC xsd:integer
    A ✓* Attribute: TRANSACTION_AMOUNT xsd:double > Range
    ID ✓* Attribute: SERVICE_TYPE MMTSTransactionCodeComp
    A* Attribute: TRANSACTION_STATUS xsd:string
    A* Attribute: SENDER_PRE_BALANCE xsd:double
    A* Attribute: SENDER_POST_BALANCE xsd:double
    A* Attribute: RECEIVER_PRE_BALANCE xsd:double
    A* Attribute: RECEIVER_POST_BALANCE xsd:double
    A* Attribute: SENDER_ACC_STATUS xsd:string
    A* Attribute: RECEIVER_ACC_STATUS xsd:string
    A* Attribute: PAY_INTER xsd:double
    A* Attribute: ERROR_CODE xsd:string
    A* Attribute: CREATED_ON xsd:string
    A* Attribute: MODIFIED_ON xsd:string
    A* Attribute: APP1_DATE xsd:string
    A* Attribute: APP2_DATE xsd:string
    A* Attribute: REF xsd:string
    A* Attribute: TRANSFER_DATE_TIME xsd:string
    A* Attribute: SENDER_CATEGORY_CODE MMTSCategoryC
    A* Attribute: RECEIVER_CATEGORY_CODE MMTSCategory
```

Figure 46: Event model for fraud detection application

The definition of the event model shown in Figure 46 reduces the event attributes to *TRANSACTION_AMOUNT*, *SERVICE_TYPE*, and *tag*. In addition, to classify the transactions with regard to the amount of money transferred a mapping like the one in Example 13 has been used for the attribute *TRANSACTION_AMOUNT*. This mapping has been created empirically using real operational logs of the MMTS. In the learning phase step 5 "Adjust process model" (cf. Figure 39) is done semi-automatically and requires the user's involvement during runtime. All other actions are performed automatically. Figure 47 shows a subgraph of an EPC which was learned for the MMTS model. The graph defines the control flow structure of a process as a chain of events and functions. Rectangles with rounded corners denote EPC functions and hexagons denote EPC events. Functions represent active components, i.e., activities, tasks or process steps, which are triggered by events. Events are passive, they represent the occurrence of a state which describes the situation before, or after, a function is executed. Logical operators (is this case an exclusive or) are used to connect

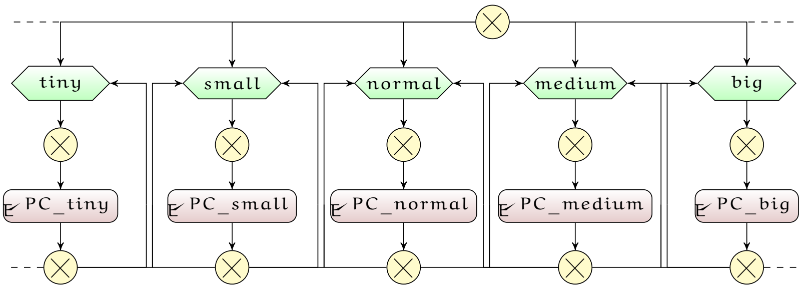


Figure 47: Subgraph of EPC for MMTS

the basic constructs. For example, after an event *medium* the function *EPC_medium* is triggered and the expected events after execution of *EPC_medium* are $\{normal, medium, big\}$.

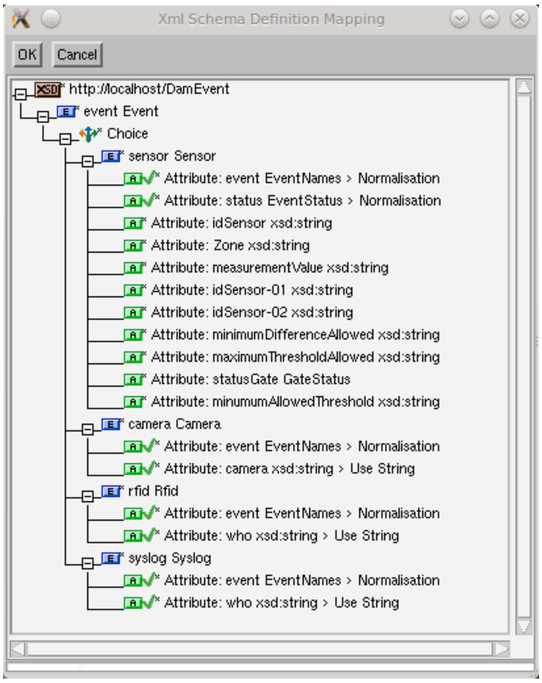
The tool adaptation was successful. The PSA is able to detect irregular events regarding the behaviour of the user of the MMTS system. It is necessary to cope with False Alarms and make decisions regarding the alerts. With the real log, the PSA was able to manage 640.000 process instances (one process per pair of users) without any problem. 40 minutes were enough to process 4.5 millions of events, with the process behaviour presented and produced 0.5 millions of alerts. Consequently, the PSA is able to manage all the logs of this operational system in real time. P18 [Rieke, Zhdanova, Repp, Giot & Gaber, 2013] describes the results of the experiments with the MMTS in detail.

### 4.5.3 *Adaptation to critical infrastructure scenario*

In the experiments with the MMTS scenario described above, only the situational awareness components of the PSA have been utilised in order to identify anomalies with respect to the process model. In the CIPC scenario it was therefore of interest, to evaluate the judgemental reasoning with respect to security compliance (cf. Section 4.3) by the PSA. For this evaluation, a combined technical and organisational process in a hydroelectric power plant in a dam has been selected from the CIPC scenario. It models a misuse case related to an insider threat that is still prevalent and posing a serious risk to critical infrastructures [Luallen, 2011].

Since dams are complex infrastructures, a huge number of parameters must be monitored to guarantee safety and security. Which parameters are actually monitored, depends on the dam's structure, design, purpose and function (cf. Section 2.4 Example 6). Figure 48 shows a mapping (an event model) with regard to the events from dam sensors, cameras, RFID scanners, and syslog.

**Example 19.** *Let the security goal be given as:* All safety critical actions in the control room are carried out by a dam operator with administrative rights.



```
⊟ XSD* http://localhost/DamEvent
  └⊟ E* event Event
     └⊟ * Choice
        ├⊟ E* sensor Sensor
        │  ├── A ✓* Attribute: event EventNames > Normalisation
        │  ├── A ✓* Attribute: status EventStatus > Normalisation
        │  ├── A* Attribute: idSensor xsd:string
        │  ├── A* Attribute: Zone xsd:string
        │  ├── A* Attribute: measurementValue xsd:string
        │  ├── A* Attribute: idSensor-01 xsd:string
        │  ├── A* Attribute: idSensor-02 xsd:string
        │  ├── A* Attribute: minimumDifferenceAllowed xsd:string
        │  ├── A* Attribute: maximumThresholdAllowed xsd:string
        │  ├── A* Attribute: statusGate GateStatus
        │  └── A* Attribute: minumumAllowedThreshold xsd:string
        ├⊟ E* camera Camera
        │  ├── A ✓* Attribute: event EventNames > Normalisation
        │  └── A ✓* Attribute: camera xsd:string > Use String
        ├⊟ E* rfid Rfid
        │  ├── A ✓* Attribute: event EventNames > Normalisation
        │  └── A ✓* Attribute: who xsd:string > Use String
        └⊟ E* syslog Syslog
           ├── A ✓* Attribute: event EventNames > Normalisation
           └── A ✓* Attribute: who xsd:string > Use String
```

Figure 48: Event model for critical infrastructure scenario

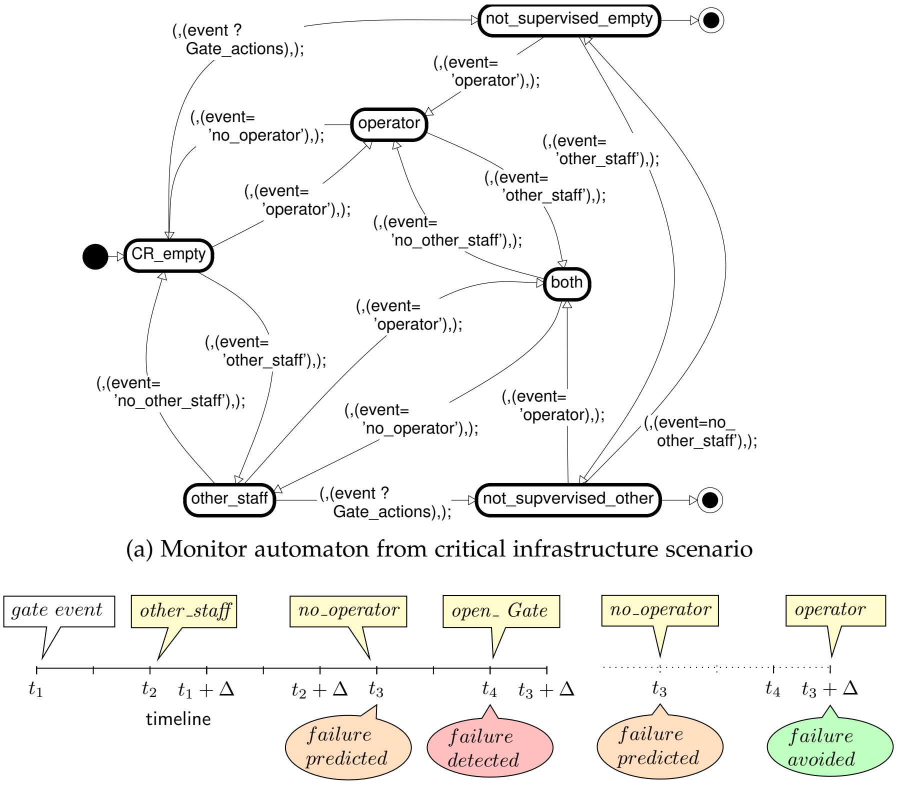
*Let only one monitor automaton as shown in Figure 49a be defined to model this security goal. The initial state* CR_empty *(control room is empty) is marked by a filled circle. The critical states* not_supervised_empty *and* not_supervised_other *are marked by a circle with a small filled circle inside. These states reflect the situation that an action from the set* Gate_actions *has been executed while the control room is either empty or only manned with non-administrative staff.*

*In order to exemplify the reasoning process at runtime, it is now assumed that the system is in a state where an operator is present in the control room, that is, the current state of the monitor automaton is* operator*. A disgruntled employee with a non-administrative role (e.g., cleaning staff) who is enabled to access the control room uses stolen administrator credentials to open dam gates. Figure 49b shows a sequence of related events that constitute this misuse case and the reaction by PSA@R.*

*If at time $t_1$ an event from a gate function is received, then the state component of the process model representing the status of the gate will be changed but the state of the monitor automaton will not change. The reachability analysis does not "see" an upcoming security violation within the scope $\Delta$.*

*If at time $t_2 > t_1$ the event* 'other_staff' *produced by the Radio Frequency IDentification (RFID) scanners of the control room is received, then the state component of the process model representing the manning of the control room will be changed and the monitor automaton changes the state to* both*. No security violation is "seen" within the scope $\Delta$.*

*If at time $t_3 > t_2$ the event* 'no_operator' *is received which indicates that the last operator has left the control room, then the state component*



(a) Monitor automaton from critical infrastructure scenario

(b) Failure predicted and detected

(c) Failure avoided

Figure 49: Security reasoning example

*of the process model representing the manning of the control room will be changed and the monitor automaton also changes the state to* other_staff. *Now in one possible process execution sequence, an event from a gate function such as* 'open_Gate' *is reachable within* $\Delta$. *In this situation the reachability analysis shows that this function would violate an associated security requirement. Therefore, a* failure is predicted *because a broken security requirement might lead to a security critical situation in the near future.*

*If at time* $t_4 > t_3$ *an event from a gate function such as* 'open_Gate' *is received, the monitor automaton changes state to* not_supervised_other *that is marked as* critical state. *Thus, a* failure is detected.

*Now let at time* $t_3 + \Delta$ *an event being received which indicates that an operator is back in the control room and the critical state was not reached as predicted. In this case, the predicted failure did not lead to a detected failure (cf. Figure 49c).*

PSA@R can also be applied to supervise other types of security requirements such as authenticity and integrity requirements (cf. Example 6). P14 exemplifies the approach using security requirements from processes in the logistics domain. As an example for a different type of security model for PSA@R, an attack path like the one depicted in Figure 29 (cf. Section 3.2) has been used in the OOGG scenario for the definition of a PSA security model. The results of application of the PSA in further domains is reported in Section 5.2 of this thesis.

## 4.6 RELATED WORK

The work presented in this chapter combines specific aspects of security analysis with generic aspects of process monitoring, simulation, and analysis. The background of those aspects is given by the utilisation of models for *process security analysis at runtime*, *information security management*, and *security information and event management*. PSA@R is a novel supplement and a link between these methods.

### 4.6.1 *Process security analysis at runtime*

A formalised approach for *security risk modelling* in the context of electronic business processes is given in Tjoa et al. [2011]. It affects also the aspect of simulation, but does not incorporate the utilisation of runtime models. A refinement methodology and modelling language for the purpose of developing secure electronic business processes based on early requirements analysis is proposed by Seguran et al. [2008]. The proposal concerns runtime enforcement of security mechanisms but does not allow for an evaluation of the security status of business processes at runtime. A model-driven approach focusing on access control for business process models is provided in Wolter et al. [2009]. It allows for the annotation of security goals to business process models, the validation of annotated process models using model checking and the generation of configuration artifacts for runtime components. Runtime analysis of security properties is not addressed by this approach. Further current approaches for the analysis and specification of security properties of business process models at development time are given by Armando et al. [2012]; Arsac et al. [2011]; Weldemariam & Villafiorita [2011]; Frankova et al. [2011].

Two approaches that focus on *security models at runtime* are given in Morin et al. [2010] and Melik-Merkumians et al. [2010]. The first approach proposes a novel methodology to synchronise an architectural model reflecting access control policies with the running system. Therefore, the methodology emphasises policy enforcement rather than security analysis. The second approach discusses the integration of runtime and development time information on the basis of an ontology to engineer industrial automation systems.

*Process monitoring* has gained some popularity recently in the industrial context prominently accompanied with the term Business Activity Monitoring (BAM). BAM applications process events, which are generated from multiple application systems, enterprise service buses, or other inter-enterprise sources in real-time in order to identify critical business key performance indicators, get a better insight into the business activities, and thereby improve the effectiveness of business operations [McCoy, 2002]. Formal methods, such as Linear Temporal Logic (LTL), state-charts, and related formalisms have been used for runtime monitoring of concurrent distributed systems in Kazhamiakin et al. [2006]; Massart & Meuter [2006]. However, this work is mainly aiming at error detection, for example, concurrency related bugs. A classification for runtime monitoring of software faults is given in Delgado et al. [2004]. Schneider [2000] analysed a class of safety properties and related enforcement mechanisms that work by monitoring execution steps of some target system, and terminating the target's execution, whenever an operation would violate the security policy. Extensions of this approach are discussed in Bauer et al. [2002] and Martinelli et al. [2005]. For example, in Martinelli et al. [2005] it is proposed to integrate a local monitor component into a grid computational service architecture. This component enforces a fine grain security policy based on a description of the correct behaviour of the applications. Only the applications whose behaviour is consistent with the security policy are executed on the computational resource. It is assumed that the application behaviour can be monitored at the system calls interface, and the policy can be enforced by preventing that the application can invoke some system calls. Patterns and methods to allow for monitoring security properties are developed in Serban & McMillin [1996]; Spanoudakis et al. [2007]; Tsigritis & Spanoudakis [2008]; Evesti et al. [2009].

In the context of process and application monitoring, in addition to the features provided by the work mentioned above, this thesis proposes a *close-future security analysis* that provides information about possible security policy violations. This functionality is based on process control-flow knowledge provided by the formal operational process model. Policy enforcement is not directly addressed by the proposed PSA@R approach, however, operational aspects of the integration of decision support components for enforcement and countermeasures are described by the SSMM proposed in P15.

Different categories of tools applicable for simulation of business processes including process modelling tools are based on different semi-formal or formal methods such as Petri Nets [Petri, 1962; Dijkman et al., 2008] or EPC [Dijkman, 2008; Mendling, 2008]. Some process management tools such as FileNet [Netjes et al., 2006] offer a simulation tool to support the design phase. Also, some general-purpose simulation tools such as CPNTools [Rozinat et al., 2009] were proven to be suitable for simulating business processes. The process mining framework ProM [van der Aalst et al., 2009; Verbeek et al., 2011] supports plug-ins for different types of models and process mining techniques [van der Aalst, 2011]. Process mining techniques for analysis of large data sets and data streams aim to extract valuable process information building on techniques from data mining and machine learning, where knowledge discovery from data is mostly based on various statistical methods [Fayyad et al., 1996]. Process mining can be seen as a specialisation of data mining for the discovery of implicit process knowledge in process flows. Many research efforts in this area are concerned with the identification of unknown control-flow of processes, referred to as *process discovery*. In addition, however, the conformity of operating records to existing process models and possible improvements in the processes are examined. However, independently from the tools and methods used, the work mentioned above concentrates on statistical aspects, redesign, and commercial optimisation of the business process. Security topics in Business Process Management (BPM), for example, seem orthogonal to the use cases and key concerns [van der Aalst, 2013].

Conversely, the approach proposed in this thesis builds on *on-the-fly* dynamic simulation and analysis on the basis of operational APA models introduced in Section 2.2. This includes consideration of the current process state and the event information combined with the corresponding steps in the process model. In P13, the first publication on PSA@R, it was proposed to use APA to specify meta-events which match security critical situations in order to generate alerts. However, since this turned out to be slow and not easily usable by end-users, it was decided to build the matching algorithm directly into the PSA. Monitor automata have been introduced in P19 to specify the operational security requirements graphically. These automata monitor the behaviour during runtime and check for (possible) security violations. For further work, it is considered to integrate methods using metrics to quantify deviations from process specifications, such as the one described in Banescu & Zannone [2011], which is, like the work presented in this chapter, based on the concept to represent system behaviour by the set of traces generated by a process.

With respect to the exhaustive survey of approaches in the field of BPM given in van der Aalst [2013], PSA@R supports the "check conformance using event data" approach. In this approach, information is used both from the process model and the event data in order to identify deviations of runtime behaviour from expected behaviour. The trend for this specific aspect of BPM, as presented in van der Aalst [2013], shows a growing interest in the last three years. A similar approach is described in Rozinat & van der Aalst [2008] but the focus is on quantification of inconsistencies by the formation of metrics. The framework presented in Maggi et al. [2011] on runtime compliance verification for business processes is considered as complementary to the work presented here.

### 4.6.2 *Information security management*

Information security management has been addressed in P15, the closely related work in Schütte, Rieke & Winkelvos [2012] and a related talk Hutchison & Rieke [2012] at the workshop on Cyber Security and Global Affairs and Global Security Forum 2012. Related work in this area is concerned with modelling security-relevant information in a way that creates the possibility to reason about it and link it to the Information Security Measurement Model (ISMM) described by the ISO27004 standard [Iso Iec, 2009]. In Fenz [2010], an approach to create ISO27001-based metrics based on a security ontology is proposed. While it lacks the automatic gathering of measurements, it could serve as a later extension to the Security Information Meta Model (SIMM) of P15, which is more focused on measurable technical events. Further, linking semantic attacker models to the : *why* part of the SSMM could be promising (cf. the Attack Modelling and Security Evaluation Component (AMSEC) model [Kotenko et al., 2012]). Another example for a potential information source is the Engineering Knowledge Base (EKB) [Melik-Merkumians et al., 2010], which is an ontology relating to sensor values and combining run-time with development time models. It is focused on the analysis of industrial automation systems, and is used to define SPARQL Protocol and RDF Query Language (SPARQL) or Semantic Web Rule Language (SWRL) queries over sensor definitions. An approach like the EKB could help defining which inconsistencies to look for in event streams, and thus which measurement points might indicate violations of the security requirements. Other approaches of interest to this end are the modelling concepts in Innerhofer-Oberperfler & Breu [2006], where business, application, physical, and technical information is merged and related, as well as concepts to use event-triggered rules for sensing and responding to business situations in Schiefer et al. [2007].

### 4.6.3 *Security information and event management*

SIEM technology provides log management and compliance reporting as well as real-time monitoring and incident management for security events from networks, systems, and applications. One objective of the work presented in this chapter was to bridge the gap between high-level security measurements and data gathered by SIEM engines, like OSSIM [AlienVault, 2012], Prelude [Prelude - CS Group, 2014], or Akab [Araknos, 2012]. OSSIM detects events at the network layer and stores respective attributes like *IP address* or *port number* in a relational database. Thus, while it is possible to link these attributes to the security information model, OSSIM itself does not support reasoning over gathered data, nor extending its model. Similarly, Akab [Araknos, 2012] is a SIEM appliance for monitoring network events. It uses a proprietary event format and also stores collected events persistently in a database. Prelude [Prelude - CS Group, 2014] is an open source SIEM framework which relies on the open IDMEF [Debar et al., 2007] event format. Also related to the envisaged model-based security information measurement are commercial tools RSA Archer, ArcSight ESM, or IBM Tivoli Security Information and Event Manager [Buecker et al., 2010]. Although they also aim at relating incidents to compliance catalogues and corporate policies, they rely on predefined event structures, comprising specific technical attributes [Software, 2010]. The RSA Archer Threat Monitor manages an assets catalogue and links it to security-relevant information, such as known vulnerabilities and patch levels. It does not, however, feature an extensible and semantic model to enable automatic reasoning regarding the implications of a detected incident with respect to the affected security requirements. It could also make amendments based on information from external sources like the PSA prototype.

A concise overview of current SIEM systems functionalities is presented in Nicolett & Kavanagh [2010]. In Securosis [2010], current threats are identified and advanced monitoring techniques such as file integrity monitoring, database activity monitoring, application monitoring, identity monitoring, and user activity monitoring are discussed. In Securosis [2011], some challenges with respect to collecting and analysing additional data sources and forensic analysis are outlined.

A new approach for the recognition, analysis and treatment of security anomalies in virtualised computing environments is developed in the project ACCEPT [Philipps-Universität Marburg, 2013; Baumgärtner et al., 2012]. The security incident measurement is supported by adequate sensors that are installed in the hypervisor, the VMs and in the runtime environments of the applications. CEP technology is used for the abstraction, correlation and aggregation of the events from all virtualisation levels. Based on the results from this analysis, the system can apply countermeasures and close potential security gaps. Offline analysis of archived events by machine learning methods is proposed in order to improve the quality of the results. Many of the concepts described in this chapter will be adopted to the specific needs of the ACCEPT environment in future work.

SIEM systems manage security events but are not primarily concerned with the trustworthiness of the event sources. Compared to traditional IT systems, securing SCADA systems poses unique challenges. In order to understand these challenges and potential dangers, Zhu et al. [2011] provides a taxonomy of possible cyber attacks – including cyber-induced cyber-physical attacks on SCADA systems. The author of this thesis has discussed specific SCADA related security problems Björkman [2010]; Dan et al. [2012] with Gunnar Björkman the former leader of the VIKING project [Viking project consortium, 2012] several times in order to better understand the problems and adapt the PSA@R concept (cf. Section 4.5.3).

In general, SIEM systems usually do not have knowledge of the application processes, thus lacking the connection between the reported security problems and the affected critical processes.

## 4.7 SUMMARY OF RESULTS

This chapter addresses predictive security analysis at runtime. The specific objective is security analysis by observing system behaviour at runtime. A given system may fail in the sense that a *judgemental system* [Randell, 2003] makes a judgement that the activity or inactivity of the given system constitutes failure. The term *fault detection* has been used for the assessment of the behaviour of the observed system by the judgemental system. The term *fault prediction* has been used to express that the judgemental system predicts a possible failure of the observed system in the near future.

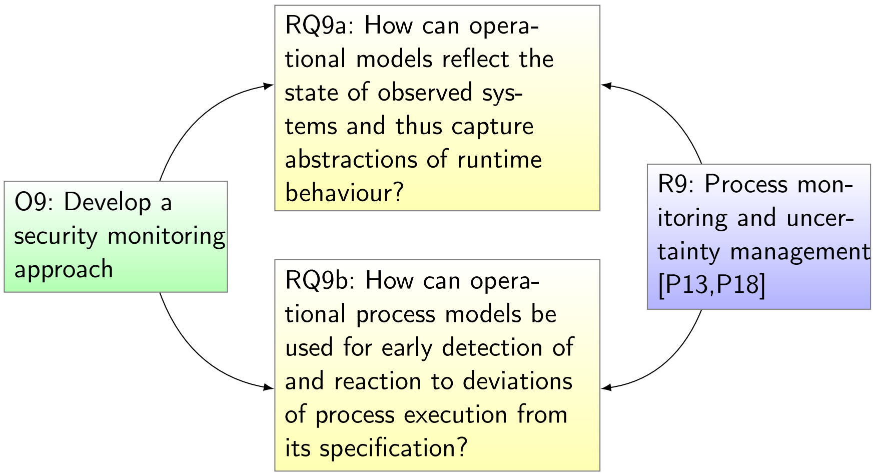
With respect to the overall aim of this thesis – to provide a framework for security analysis of system behaviour – this chapter addresses the “Study“ and “Act” activities in the Plan-Do-Study-Act (PDSA) cycle (cf. Figure 4). The Study activity aims at *fault detection* by assessment of the behaviour of the observed system by the judgemental system as well as *fault prediction* by forecasting a possible failure of the observed system in the near future. The Act activity analyses security consequences, determines their root causes, and triggers corrective actions.

The thesis provides a method and tool to observe - *de-facto* - behaviour of processes, compare it with the planned - *de-jure* - behaviour, and evaluate security compliance at runtime. Where applicable, knowledge on process’ expected behaviour is used for forecasting critical situations in the near future. The reported results also take into account other relevant context information such as the current attack state for review and countermeasure assessment. Furthermore, a concept to integrate the tool into a holistic security management strategy is proposed.

The thesis provides methods and tools to analyse possible corrective and preventive actions - based on the results of the security assessment - as well as a tool to trigger their execution in order to achieve continual improvement of the system.

### 4.7.1 *Process monitoring and uncertainty management*

Objective O9 of this thesis was to *develop a security monitoring approach*. The intention to use the results of the research work on operational models presented in the previous chapters led to the idea to leverage this knowledge at runtime and thus initiated the research questions RQ9a and RQ9b.


RQ9a: How can operational models reflect the state of observed systems and thus capture abstractions of runtime behaviour?
O9: Develop a security monitoring approach
R9: Process monitoring and uncertainty management [P13,P18]
RQ9b: How can operational process models be used for early detection of and reaction to deviations of process execution from its specification?

An architectural blueprint of the PSA@R approach addressing research question RQ9a has first been published in P13. The control flow of model learning and uncertainty reasoning addressing research question RQ9b has been published in P18. In the following, the contribution of these papers with respect to the above research questions is described.

P13. Predictive Security Analysis for Event-Driven Processes

The main constraint of current systems is the restriction of Security Information and Event Management (SIEM) [Nicolett & Kavanagh, 2009] to network infrastructure, and the inability to interpret events and incidents from other layers such as the service view, or the business impact view, or on a viewpoint of the service itself. This paper presents an approach for predictive security analysis in a business process execution environment. It is based on operational process models and leverages process and threat analysis and simulation techniques in order to be able to dynamically relate events from different processes and architectural layers and evaluate them with respect to security requirements. Based on this, a blueprint of an architecture is presented which can provide decision support by performing dynamic simulation and analysis while considering real-time process changes. It allows for the identification of close-future security-threatening process states and will output a predictive alert for the corresponding violation.

P18. Fraud Detection in Mobile Payment Utilizing Process Behavior Analysis

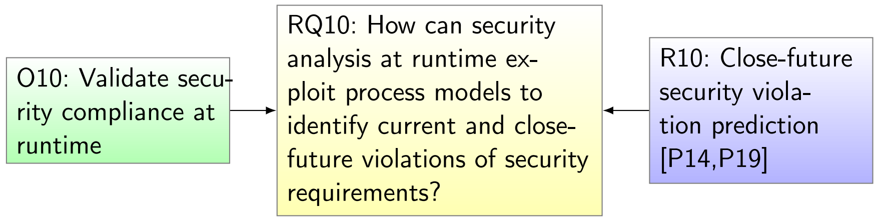
This paper describes the control flow of *model learning* and *uncertainty reasoning* for *anomaly detection* in PSA@R. This is exemplified by detection of money laundering patterns in synthetic process behaviour composed of simulated logs based on properties captured from real world money transaction events.

In the initial learning phase, the normal behaviour pattern with regard to the transaction characteristics is learned by processing an event log without malicious content. A mapping to classify the transactions with regard to the amount of money transferred and the order of such abstract events in the event log is used for this purpose. This mapping is created empirically using real operational logs of the Mobile Money Transfer (MMT) system and can change if different training sets are used. An overview of the event processing steps of PSA@R in the learning phase is given.

In the anomaly detection phase PSA@R is used to identify deviations from the normal characteristics based on the values from a transaction monitor. In an experimental setup, the transaction monitor has been replaced by synthetic process behaviour composed of simulated logs based on properties captured from real logs.

### 4.7.2 *Close-future security violation prediction*

Objective O10 of this thesis was to *validate security compliance at runtime*. This motivated research question RQ10.


O10: Validate security compliance at runtime
RQ10: How can security analysis at runtime exploit process models to identify current and close-future violations of security requirements?
R10: Close-future security violation prediction [P14,P19]

The results published in P14 and P19 address this research question. In the following, an overview of these papers with respect to the above research questions is given.

P14. Model-based Situational Security Analysis

Security analysis is growing in complexity with the increase in functionality, connectivity, and dynamics of current electronic business processes. To tackle this complexity, the application of models in pre-operational phases is becoming standard practice. Runtime models are also increasingly applied to analyse and validate the actual security status of business process instances. This paper presents an approach to support not only model-based evaluation of the current security status of business process instances, but also to allow for decision support by analysing close-future process states. The approach is based on operational formal models derived from development-time process and security models. This paper exemplifies the approach utilising real world processes from the logistics domain and demonstrates the systematic development and application of runtime models for situational security analysis.

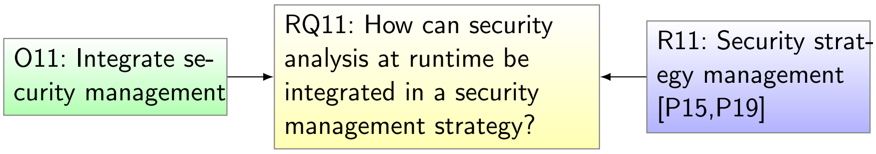
P19. Monitoring Security Compliance of Critical Processes

With respect to RQ10, this paper presents an approach to support evaluation of the security status of processes at runtime. The approach is based on operational formal models derived from process specifications and security policies comprising technical, organisational, regulatory and cross-layer aspects. A process behaviour model is synchronised by events from the running process and utilises prediction of expected close-future states to find possible security violations and allow early decisions on countermeasures.

In particular, the algorithm for the evaluation of security requirements at runtime is described.

### 4.7.3 *Security strategy management*

Objective O11 of this thesis aimed to *integrate security management*. Research question RQ11 and the respective results contribute to this objective by providing an architecture to integrate the PSA prototype into a security management framework and a meta model that consolidates the necessary security strategy information.


O11: Integrate security management
RQ11: How can security analysis at runtime be integrated in a security management strategy?
R11: Security strategy management [P15,P19]

The results published in P15 and P19 address this research question. In the following, an overview of these papers with respect to the above research questions is given.

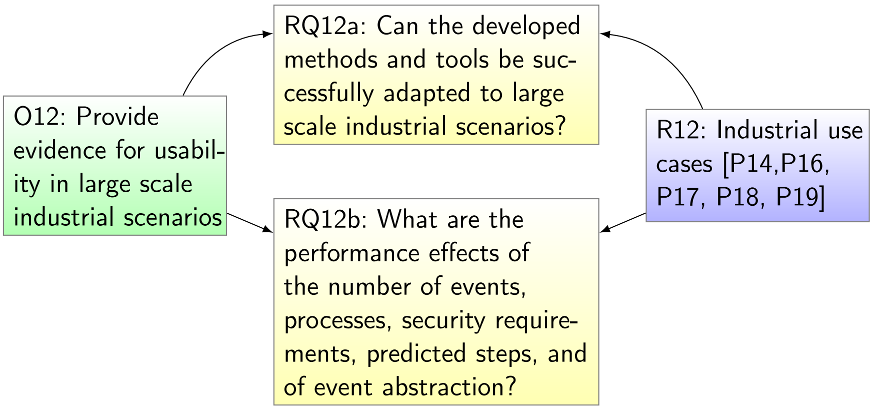
P15. Architecting a Security Strategy Measurement and Management System

This paper presents a model driven approach for architecting a *security strategy measurement and management system*. Concretely, it describes the definition of security objectives for a particular system, and a mechanism for collecting information from operational systems in a manner which enables assessment and measurement of how well the system is fulfilling the security objectives. Existing SIEM solutions are limited, while this approach overcomes these contextual restrictions (typically predefined, closed models) offering an extensible and open model, encompassing all parts of the security monitoring and decision support process, namely: (i) detecting threatening events; (ii) putting them in context of the current system state; (iii) explaining their potential impact with respect to some security- or compliance model; and (iv) taking appropriate actions. The proposed deployment model brings together all parts of security runtime management, namely, detection, reporting, handling, and explanation of security incidents, which are to date covered by different systems, such as intrusion detection systems, CEP engines [Esper contributors and EsperTech Inc., 2012], SIEM systems [AlienVault, 2012; Prelude - CS Group, 2014; Araknos, 2012], intrusion response systems [Shameli-Sendi et al., 2012], cyber attack information systems [Skopik et al., 2012], and governance, risk management, and compliance systems [Racz et al., 2010]. So, the model supports an integration of functionalities of these existing systems into one coherent security strategy management framework.

P19. Monitoring Security Compliance of Critical Processes
In this paper, the implementation of the PSA@R approach by the prototype, the PSA, is described and results of evaluation of specific aspects, such as effects of the number of security requirements, different abstraction levels and the variation of prediction depths are provided.

### 4.7.4 *Industrial use cases*

Objective O12 of this thesis was to *provide evidence for usability in large scale industrial scenarios*. Research question RQ12a thus investigated the applicability of the developed methods in industrial scenarios and RQ12b analysed the usability and performance of the PSA prototype.



O12: Provide evidence for usability in large scale industrial scenarios

RQ12a: Can the developed methods and tools be successfully adapted to large scale industrial scenarios?

RQ12b: What are the performance effects of the number of events, processes, security requirements, predicted steps, and of event abstraction?

R12: Industrial use cases [P14,P16, P17, P18, P19]

The results published in P16, P17, P18, and P19 address this research questions. In the following, an overview of these papers with respect to the above research questions is given.

P16. MASSIF: A Promising Solution to Enhance Olympic Games IT Security
This paper addresses the security management challenges that arise in the cyber-security of Olympic Games and how advanced SIEM can help to improve it. Nowadays, Olympic Games have become one of the most profitable global media events, becoming at the same way more and more attractive target from the terrorist perspective due to their media diffusion and international dimension. Critical for the

success of such a highly visible event is protecting and securing the business and the supporting cyber-infrastructure enabling it. In this context, the MASSIF project aims to provide a new generation SIEM framework for service infrastructures supporting intelligent, scalable, and multilevel/multi-domain security event processing and predictive security monitoring.

### P17. Security and Reliability Requirements for Advanced Security Event Management

This paper addresses security information management in complex application scenarios. SIEM systems collect and examine security related events, with the goal of providing a unified view of the monitored systems' security status. While various SIEMs are in production, there is scope to extend the capability and resilience of these systems. The use of SIEM technology in four disparate scenario areas is used in this paper as a catalyst for the development and articulation of Security and Reliability requirements for advanced security event management. The scenarios relate to infrastructure management for a large real-time sporting event, a mobile money payment system, a managed services environment and a cyber-physical dam control system. The diversity of the scenarios enables elaboration of a comprehensive set of security and reliability requirements which can be used in the development of future SIEM systems.

### P18. Fraud Detection in Mobile Payment Utilizing Process Behavior Analysis

In this paper the applicability of the PSA@R approach is exemplified by a MMT scenario. MMT systems are systems where electronic money is issued to different roles in order to perform various types of transactions. As with any payment system, this service can be an attractive target for attackers and fraudsters. For legal and service security issues it is mandatory to observe the transactions for potential abnormal activities. The work presented in this paper utilises alerts generated by the uncertainty reasoning component of the PSA prototype to detect money laundering patterns in synthetic process behaviour composed of simulated logs based on properties captured from real world transaction events. In particular, it is shown that the PSA is able to raise alerts in a simulated scenario of fraud with mules. For this simulated scenario, the detection is efficient, but show that such system could be sensitive to noise in a real world system. It would be necessary to improve the resistance to noise through a correlation of the generated alerts or by an application of specific evaluation of the process states when an alert is generated. The applicability of the proposed approach is evaluated and provides measurements on computational and recognition performance of the tool.

P19. Monitoring Security Compliance of Critical Processes

In this paper, the applicability of the PSA@R approach is exemplified by a misuse case scenario from a hydroelectric power plant that was analysed in the European research project MASSIF. Security requirements are taken from a combined technical and organisational process from a hydroelectric power plant in a dam [Romano et al., 2012]. Since dams are complex infrastructures, a huge number of parameters must be monitored to guarantee safety and security. Which parameters are actually monitored, depends on the dam's structure, design, purpose and function [Coppolino et al., 2012].

In particular, the algorithm for the evaluation of security requirements at runtime is described and an extensive example concerning safety critical actions in the control room is given.

In the project MASSIF [Rieke et al., 2012] PSA@R is currently applied to check security requirements in four industrial domains: (i) the management of the Olympic Games IT infrastructure [Vianello et al., 2013]; (ii) a mobile phone based MMTS [Gaber et al., 2013], facing high-level threats such as money laundering; (iii) managed IT outsource services for large distributed enterprises and (iv) an IT system supporting a critical infrastructure (dam) [Romano et al., 2012]. The hydroelectric power plant scenario (iv) has been used to demonstrate the capability of the PSA prototype to process and correlate events from heterogeneous sources. To evaluate the PSA prototype with respect to performance issues, however, event logs from scenario (ii) have been used as a resource intensive application which requires high throughput.

The measurements presented evaluate the execution time and the number of events received by the PSA. Four aspects important from the application perspective have been examined: (i) effects of the number of security requirements to the execution time; (ii) effects of the abstraction level to analysis; (iii) effects of cycle reduction in a RG; (iv) effects of changing prediction depths.

During the experiment the security requirements were successfully checked in all combinations.

### 4.7.5 *Conclusion*

In summary, the work presented in this chapter provides an integrated approach called PSA@R to analyse the security status of a process and to identify possible violations of the security policy in close future. The approach also provides early awareness about deviations of a running process from expected behaviour as specified by the model. When such anomalies refer to process misbehaviour or disruption, alarms will be raised for decision support and reaction. Moreover, it is described how to extend process behaviour computation with algorithms for on-the-fly security compliance checks and prediction of close-future security violations. Thus, the proposed integrated security analysis approach identifies current and close-future violations of the security policy. As security relies on the compliance of actual behaviour with the given specifications this early detection of changes and reaction elevates security of the process in question. In combination with other novel applications PSA@R enables anticipatory impact analysis, decision support and impact mitigation by adaptive configuration of countermeasures. A prototype PSA has been implemented by Fraunhofer SIT headed by the author of this thesis in order to evaluate the applicability and performance of different modelling strategies in the scope of PSA@R. The approach has been validated specifically with respect to security concerns but is also applicable to on-the-fly analysis of generic compliance and dependability requirements.

Further noteworthy work of the author of this thesis with respect to the work presented in this chapter comprises peer-reviewed publications regarding model-based security event management [Schütte et al., 2012] and challenges for advanced security monitoring [Rieke et al., 2012].

Invited talks related to the results of this chapter have been given at the Cyber Security & Privacy EU Forum 2013 in Brussels [CSP EU FORUM, 2013; Rieke & Giot, 2013], at the Second International Workshop on Scientific Analysis and Policy Support for Cyber Security in ST. Petersburg 2012 [Rieke, 2012b], at the Cyber Security & Privacy EU Forum 2012 in Berlin [Rieke, 2012a,a], at the 2012 Workshop on Cyber Security and Global Affairs and Global Security Forum in Barcelona [Hutchison & Rieke, 2012], at the 2011 Workshop on Cyber Security and Global Affairs in Budapest [Hutchison & Rieke, 2011], at the Effectsplus Trustworthy ICT Research Roadmap Session Cluster Meeting 2011 in Brussels [Rieke, 2011; Cleary, 2011a], at the "ICT 2010: Digitally Driven" in Brusels [Rieke, 2010c], and at the 2010 Workshop on Cyber Security and Global Affairs in Zurich [Rieke, 2010a].

The author of this thesis further organised a workshop on models [Cleary, 2011b] and co-organised the 2nd International Workshop on Recent Advances in Security Information and Event Management (RaSIEM 2013) in conjunction with the 8th International Conference on Availability, Reliability and Security (ARES 2013). In the European project MASSIF he was responsible for roadmapping [Rieke et al., 2011, 2012, 2013], and the lead of the activity on "Event-driven Process Models and Attack Simulation".

# 5 CONCLUSION

> *Although I do not know how to solve the problems I have raised, I believe that progress will be made soon on all of them, not only on the theoretical side, but on building systems that use sophisticated methods of reasoning about uncertainty to tackle large, complex real-world problems. It is an exciting time to be working on reasoning about uncertainty.*
>
> — Joseph Y. Halpern [Halpern, 2003]

AIM OF THIS CHAPTER. The results of this thesis provide a framework for security analysis of system behaviour. Operational models are utilised for security and - to some extent - dependability analysis at design time, configuration time and at runtime. This chapter evaluates the research proposition and the outcomes planned in Chapter 1. Furthermore, it considers current and future application domains for the approach, open issues, and lessons learnt.

## 5.1 SUMMARY

The starting point of this cumulative thesis has been the overall aim to provide a modelling framework that is suitable for security and dependability analysis throughout the life cycle of a system: for design, configuration, and monitoring during operation as well as in the context of system adaptations to changing requirements and application context. The work in this thesis has been partitioned into three topics, namely, security of cooperating system design, security of system configurations, and predictive security analysis at runtime.

In the first chapter of this thesis, the research topic has been motivated, research questions have been derived, and the research contributions have been described.

The second chapter has introduced a specification and analysis framework for model-based analysis of system behaviour with respect to systematically deduced security and reliability requirements. Theoretical foundations that extend this approach for the verification of scalable systems and design principles that facilitate such verifiability have been given.

The third chapter has been concerned with a systematic security assessment of system configurations aiming to identify the critical components and their interplay, to determine the threats and vulnerabilities, to assess the risks, and to prioritise countermeasures where risk is unacceptable. The presented approach builds on a model-based construction of an attack graph taking into account constraints given by the network security policy. The most distinctive feature of this approach is the ability to compute abstract representations of these complex graphs that enable comparison of focussed views on the behaviour of the system. In order to analyse resilience of information infrastructures against exploits of unknown vulnerabilities by zero day attacks, generic vulnerabilities for each installed product and affected service are added to the model. Furthermore, an approach for the validation and deployment of a security policy has been given.

In the fourth chapter, the utilisation of operational models for security analysis at runtime has been considered. In particular, an approach to support model-based evaluation of the current and possible close-future security status of process instances has been described. An integration concept for a holistic security strategy management has been proposed and the applicability of the approach has been exemplified utilising processes from several industrial scenarios.

In summary, this thesis has provided the research results shown in Figure 50.

In particular, the results R1 – R4 are concerned with model-based verification of correct operation of Cooperating Systems (CS) with respect to security and dependability and thus contribute to *prevent the introduction of faults* in the early system design phase as well as *fault removal* in system redesign phases.

*Fault tolerance* means to avoid service failures in the presence of faults, and *fault forecasting* means to estimate the present number, the future incidence, and the likely consequences of faults [Avizienis et al., 2004]. The results R5 – R8 contribute to specific aspects of fault forecasting and fault tolerance by analysis of networked system configurations with respect to external exploitability of inherent vulnerabilities.

The results R9 – R12 are concerned with the utilisation of operational models for security analysis at runtime. *Fault detection* with respect to the specification given by an operational model allows the assessment of the correct behaviour of the observed system by a judgemental system. *Fault prediction*, in addition, provides the ability to anticipate possible failures of the observed system with respect to a given security model in the near future.

## 5.2 APPLICATION DOMAINS

Due to the capability to model and analyse the critical behaviour of discrete Systems of Systems (SoS), potential applications of the framework presented in Chapter 2 are manifold. Particularly, for safety critical systems, assuring the correctness - conformance to the intended purpose - is imperative. Some examples of successful application are

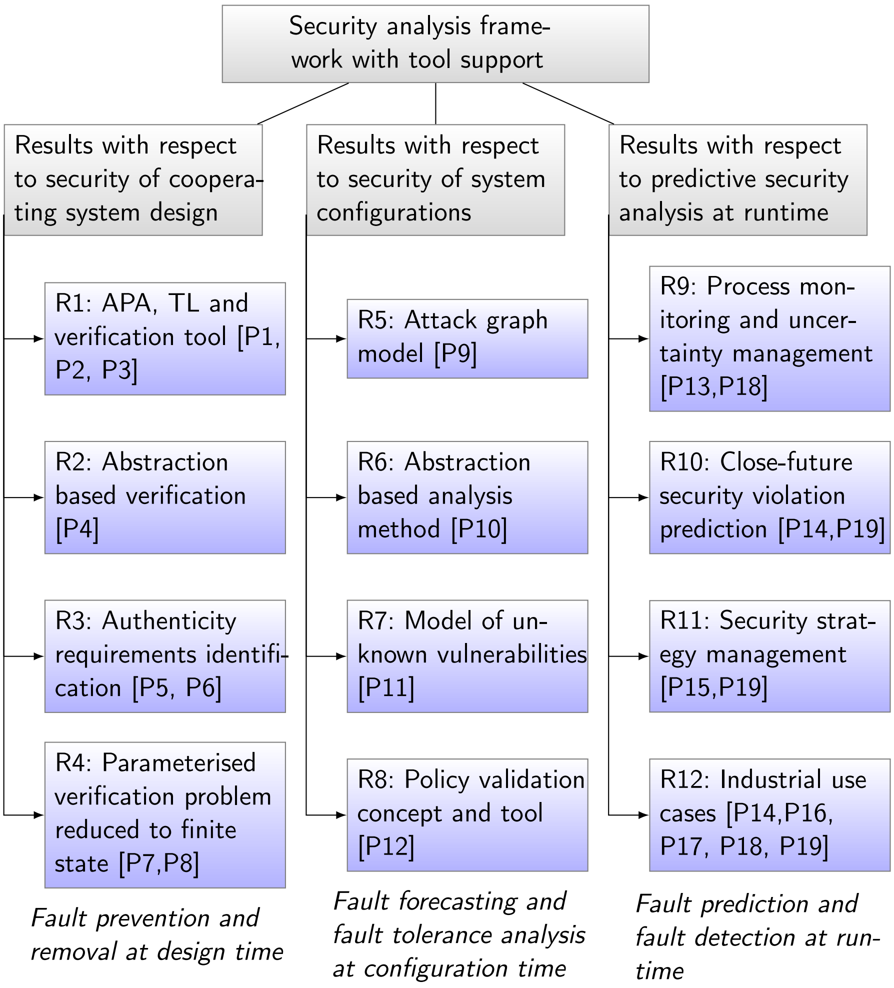

Security analysis framework with tool support
Results with respect to security of cooperating system design
Results with respect to security of system configurations
Results with respect to predictive security analysis at runtime
R1: APA, TL and verification tool [P1, P2, P3]
R2: Abstraction based verification [P4]
R3: Authenticity requirements identification [P5, P6]
R4: Parameterised verification problem reduced to finite state [P7,P8]
R5: Attack graph model [P9]
R6: Abstraction based analysis method [P10]
R7: Model of unknown vulnerabilities [P11]
R8: Policy validation concept and tool [P12]
R9: Process monitoring and uncertainty management [P13,P18]
R10: Close-future security violation prediction [P14,P19]
R11: Security strategy management [P15,P19]
R12: Industrial use cases [P14,P16, P17, P18, P19]
Fault prevention and removal at design time
Fault forecasting and fault tolerance analysis at configuration time
Fault prediction and fault detection at runtime

Figure 50: Research contributions

now given. The Simple Homomorphism Verification Tool (SHVT) has been applied in an industrial setting for model-based test case generation in order to validate security, interoperability, and robustness of the electronic health card (eGK) and the identity card for health care professionals [Apel, Repp, Rieke & Steingruber, 2007]. In the project ProOnline-VSDD an operational Asynchronous Product Automaton (APA) model of the communication infrastructure of the German eHealth Card [Stroetmann & Lilischkis, 2007; Deutsche Krankenhaus Gesellschaft, 2008] was developed by the author of this thesis to verify several security requirements for specific applications [Rieke, 2009b]. In the project NoW [Rieke & Steinemann, 2007; Festag et al., 2008] an operational APA model of the movement of vehicles as well as a communication model for the WAVE service [Intelligent Transportation Systems Committee of the IEEE Vehicular Technology Society, 2006] was used. The security requirements elicitation method proposed in Chapter 2 has been used in several different application domains. In the project EVITA [Fraunhofer SIT, 2011] it has been used to derive security requirements for vehicle-to-vehicle communication [Ruddle et al., 2009], in the project ADiWa [ADiWa Konsortium, 2012] to derive authenticity requirements in a logistics application (cf. P14), and in one of the MASSIF scenarios this method has been applied in the critical infrastructure domain (cf. P6).

An *attack simulator* based on the work presented in Chapter 3 has been developed by the author of this thesis in context of a security concept in the domain of e-service processes that was developed in the project SKe [Sarbinowski, 2002]. The policy validation approach presented in Chapter 3 has been applied in the project SicAri [Rieke & Ebinger, 2008; Peters, 2013]. The attack graph approach has furthermore been the base for a successful cooperation with MASSIF partner SPIIRAS. SPIIRAS developed the Attack Modelling and Security Evaluation Component (AMSEC) component which communicates with the Predictive Security Analyser (PSA) in the MAnagement of Security information and events in Service InFrastructures (MASSIF) framework.

In the scope of the project MASSIF, the PSA prototype has been adapted to four industrial scenarios by the participants of the project team. The PSA has then been evaluated as a component in different tool chains of a Security Information and Event Management (SIEM) framework by these project partners. The results of the adaptation of the PSA prototype are documented in Hutchison et al. [2013]. The PSA evaluation as integrated component in the MASSIF framework is described in MASSIF project consortium [2013a]. The results are promising as the following quotations from the respective MASSIF deliverables suggest.

> “Other visual components like the AMSEC and PSA require some initial knowledge and configuration to make them fully operative, but once this part is completed, the regular use is fairly simple. ”
>
> — Comment from PSA evaluation in the Olympic Games IT infrastructure (OOGG) scenario MASSIF project consortium [2013a]

> “The PSA requires building a model which corresponds to the business process. In the case of the MMT scenario, it is the user’s behaviour which is monitored. To use the PSA with the MMT therefore required creating a way of modelling the users’ behaviour. We decided to model an expected behaviour for each user and transaction type. From the moment the model was defined, the configuration and use of the PSA is easy.”
>
> — Comment from PSA evaluation in the Mobile Money Transfer (MMT) scenario in MASSIF project consortium [2013a]

"PSA has high applicability of adaptation to this particular scenario as it can facilitate the validation of extensive processes that are created through the collaboration of many devices and networks managed within a central system. In this particular case login authentication and verification is sent from more than one device and can be used to investigate the process for unusual activity."

— Comment from PSA adaptation to the Managed
Enterprise Service Infrastructures (MESI) scenario in
Hutchison et al. [2013]

"The application of the PSA to the MESI scenario has shown itself to be possible through the identification of technical processes such as the associated procedures of logon logoff authorisation. Through the description of a required sequence of events process violations can be flagged and raise alerts. The potential of this solution mechanism has been confirmed through the adaptation input events available. The extension of this technique to similiar technical processes looks promising and is an opportunity for further testing and deployment. "

— Comment from PSA adaptation to the MESI scenario in
Hutchison et al. [2013]

"In the Misuse Case 1 the PSA generates a warning when WaterLevel values decrease and before the *ALARM* events are generated by *WaterLevelCoherence*, *SeepageFlowCheck*, *TurbidityLevelCheck*. In the Misuse Case 4, the PSA generate warning when the event 5 happens and before the water level decreases significantly. For the misuse case 1, the warning can be used to send reprogramming commands toward the reaction systems. For the misuse case 4, the warnings can be used to denying the access of the suspicious role or identity to the control machine.

. . .

The PSA tool managed to generate warning messages that can be used to reconfigure the system in a timely manner and before the attacks are completed. The adaptation gave constructive results that are useful for the scenario in detecting the Misuse Cases 1 and 4."

— Comment from PSA adaptation to the Critical
Infrastructure Process Control (CIPC) scenario in
Hutchison et al. [2013]

## 5.3 LESSONS LEARNT

1. APA not only scale as framework for security analysis of system design but also fault tolerance and security at runtime.

2. Inductive proofs on the construction of a parameterised system which show that it results in identical abstract system behaviour for any given parameter configuration allow the verification of parameterised systems by constructing abstract systems that can be model checked.

3. Functional dependencies and information flow analysis can be used to identify a comprehensive set of authenticity requirements.

4. Behavioural self-similarity is an important property that avoids unwanted emergent behaviour when extending uniformly parameterised systems to large scale and thus is considered as an important construction aim for well-behaved scalable systems.

5. The configuration of the network policy of an Information and Communications Technology (ICT) system influences the exposition of vulnerable components to external access. Therefore, the analysis of attack graphs and the optimisation of the network security policy based on the results of this analysis can improve the fault tolerance and thus the security of a system as a whole.

6. Abstract representations of an attack graph can be computed and used to visualise and analyse compacted information focussed on interesting aspects of the behaviour. Abstractions have to be *property preserving*, to assure that properties are *transported* as desired from a lower to a higher level of abstraction and no critical behaviour is hidden.

7. Abstraction-based analysis allows to assess the resilience of networked information systems, and, in particular, the identification of weak points with respect to zero day attacks.

8. Holistic policy management and validation is required to bridge the gap between the informal specification of security policies – *what the security administrator wants to enforce* – and its corresponding machine-readable policy specification – *what the system actually enforces* –.

9. Systems and applications need to be *designed for security assessment at runtime*, for example, it must be possible to identify the originating source of events up to the process instance level.

10. Model-based *observing systems* can be extended by security models to *judgemental systems*. Anticipated behaviour helps to predict possible failures.

11. Goals, policies, measurement information, and decision rules used in security management need a meta model that consolidates the necessary security strategy information.

12. Model-based analysis is applicable and fast enough for security analysis of important real-world applications at runtime.

# BIBLIOGRAPHY

ADiWa Konsortium (2012). Project ADiWa (Alliance Digital Product Flow). `http://www.adiwa.net/`. [Online; accessed 13-Oct-2013]. (Cited on pages 14, 68, 112, and 134.)

Aho, A. & Ullman, J. (1995). *Foundations of Computer Science: C Edition*. Principles of Computer Science Series. W. H. Freeman. (Cited on page 21.)

AlienVault (2012). AlienVault Unified SIEM. http://alienvault.com/. [Online; accessed 16-Sep-2012]. (Cited on pages 120, 126, and 404.)

Alpern, B. & Schneider, F. B. (1985). Defining Liveness. *Information Processing Letters*, *21*(4), 181–185. (Cited on pages 18, 42, and 59.)

Alur, R. & Henzinger, T. A. (1995). Local liveness for compositional modeling of fair reactive systems. In Wolper, P. (Ed.), *Computer Aided Verification (CAV) '95*, volume 939 of *Lecture Notes in Computer Science*, (pp. 166–179). Springer. (Cited on page 58.)

Ammann, P., Wijesekera, D., & Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM conference on Computer and communications security*, (pp. 217–224). ACM Press New York, NY, USA. (Cited on page 87.)

Anderson, A. (2005). *Core and hierarchical role based access control (RBAC) profile of XACML v2.0 Oasis Standard, 1 February 2005*. OASIS. (Cited on page 89.)

Apel, C., Repp, J., Rieke, R., & Steingruber, J. (2007). Modellbasiertes Testen der deutschen Gesundheitskarten. In Horster, P. (Ed.), *DACH Security 2007 - Bestandsaufnahme, Konzepte, Anwendungen, Perspektiven.*, (pp. 338–346). (Cited on pages 17, 44, 64, 68, 133, and 196.)

Apt, K. R. & Kozen, D. C. (1986). Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, *22*(6), 307–309. (Cited on page 63.)

Araknos (2012). Araknos website. `http://www.araknos.it/en.html`. [Online; accessed 16-Sep-2012]. (Cited on pages 120, 126, and 404.)

Armando, A., Giunchiglia, E., Maratea, M., & Ponta, S. E. (2012). An action-based approach to the formal specification and automatic analysis of business processes under authorization constraints. *Journal of Computer and System Sciences*, *78*(1), 119–141. (Cited on page 117.)

Arsac, W., Compagna, L., Pellegrino, G., & Ponta, S. (2011). Security Validation of Business Processes via Model-Checking. In *Engineering Secure Software and Systems (ESSoS 2011)*, volume 6542 of *LNCS* (pp. 29–42). Springer. (Cited on page 117.)

Auyang, S. Y. (1998). *Foundations of complex-system theories: In economics, evolutionary biology, and statistical physics*. Cambridge, UK: Cambridge University Press. (Cited on page 1.)

Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. E. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, *1*(1), 11–33. (Cited on pages 3, 4, 6, 9, 18, 58, 63, 94, and 132.)

Baeten, J. & Weijland, W. (1990). *Process Algebra*. Cambridge University Press. (Cited on page 39.)

Baier, C. & Katoen, J. (2008). *Principles of Model Checking*. Mit Press. (Cited on page 36.)

Bandara, A., Lupu, E. C., & Russo, A. (2003). Using event calculus to formalise policy specification and analysis. In *Workshop on Policies for Distributed Systems and Networks*. IEEE. (Cited on page 61.)

Banescu, S. & Zannone, N. (2011). Measuring privacy compliance with process specifications. In *Workshop on Security Measurements and Metrics (MetriSec 2011)*. IEEE. (Cited on page 119.)

Barnum, S. & Sethi, A. (2007). Attack Patterns as a Knowledge Resource for Building Secure Software. In *OMG Software Assurance Workshop: Cigital*. (Cited on page 87.)

Basu, S. & Ramakrishnan, C. R. (2006). Compositional analysis for verification of parameterized systems. *Theor. Comput. Sci.*, *354*(2), 211–229. (Cited on page 62.)

Bauer, L., Ligatti, J., & Walker, D. (2002). More enforceable security policies. In *Workshop on Foundations of Computer Security (FCS 2002)*. (Cited on page 118.)

Baumgärtner, L., Graubner, P., Leinweber, M., Schwarzkopf, R., Schmidt, M., Seeger, B., & Freisleben, B. (2012). Mastering Security Anomalies in Virtualized Computing Environments via Complex Event Processing. In *Proceedings of the The Fourth International Conference on Information, Process, and Knowledge Management (eKNOW 2011)*, (pp. 76–81). XPS. (Cited on pages 111 and 121.)

Bell, D. E. & LaPadula, L. J. (1974). Security computer systems: Mathematical foundations and model. Technical report, MITRE Corp., Bedford, Mass. (Cited on page 89.)

Bell, D. E. & LaPadula, L. J. (1976). Secure computer systems: Unified exposition and multics interpretation. MTR-2997, (ESD-TR-75-306), available as NTIS AD-A023 588, MITRE Corporation. (Cited on page 89.)

Ben-Ari, M. (2008). *Principles of the Spin Model Checker*. Springer. (Cited on page 45.)

Ben Mustapha, Y. & Debar, H. (2013). Service dependencies-aware policy enforcement framework based on hierarchical colored petri net. In S. Thampi, P. Atrey, C.-I. Fan, & G. Perez (Eds.), *Security in Computing and Communications*, volume 377 of *Communications* *in Computer and Information Science* (pp. 313–321). Springer Berlin Heidelberg. (Cited on page 90.)

Benenson, Z., Freiling, F. C., Holz, T., Kesdogan, D., & Penso, L. D. (2006). Safety, liveness, and information flow: Dependability revisited. In *ARCS Workshops*, (pp. 56–65). (Cited on page 59.)

Bert, D. & Cave, F. (2000). Construction of finite labelled transition systems from b abstract systems. In W. Grieskamp, T. Santen, & B. Stoddart (Eds.), *Integrated Formal Methods*, volume 1945 of *Lecture Notes in Computer Science* (pp. 235–254). Springer Berlin Heidelberg. (Cited on page 36.)

Bhattacharya, K., Caswell, N. S., Kumaran, S., Nigam, A., & Wu, F. Y. (2007). Artifact-centered operational modeling: lessons from customer engagements. *IBM Syst. J.*, *46*(4), 703–721. (Cited on page 7.)

Bistarelli, S., Fioravanti, F., & Peretti, P. (2006). Defense trees for economic evaluation of security investments. In *ARES*, (pp. 416–423). IEEE Computer Society. (Cited on page 86.)

Bistarelli, S., Fioravanti, F., Peretti, P., & Santini, F. (2012). Evaluation of complex security scenarios using defense trees and economic indexes. *J. Exp. Theor. Artif. Intell.*, *24*(2), 161–192. (Cited on page 86.)

Björklund, H. & Bojanczyk, M. (2007). Shuffle Expressions and Words with Nested Data. In *Mathematical Foundations of Computer Science 2007*, (pp. 750–761). (Cited on pages 62 and 100.)

Björkman, G. (2010). The viking project - towards more secure scada systems. In *First Workshop on Secure Control Systems (SCS)*. (Cited on pages 87, 98, and 121.)

Bodeau, D. J. (1994). System-of-Systems Security Engineering. In *In Proc. of the 10th Annual Computer Security Applications Conference, Orlando, Florida*, (pp. 228–235). IEEE Computer Society. (Cited on pages 6 and 61.)

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. In *Workshop on Software and Performance*, (pp. 195–203). (Cited on pages 51 and 52.)

Bonzanni, N., Feenstra, K. A., Fokkink, W., & Krepska, E. (2009). What can formal methods bring to systems biology? In Cavalcanti, A. & Dams, D. (Eds.), *FM*, volume 5850 of *Lecture Notes in Computer Science*, (pp. 16–22). Springer. (Cited on page 16.)

Bradfield, J. & Stirling, C. (2001). Modal logics and mu-calculi: an introduction. (Cited on page 62.)

Bryans, J. (2005). Reasoning about XACML policies using CSP. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, (pp. 28–35)., New York, NY, USA. ACM Press. (Cited on page 89.)

Buecker, A., Amado, J., Druker, D., Lorenz, C., Muehlenbrock, F., & Tan, R. (2010). *IT Security Compliance Management Design Guide with IBM Tivoli Security Information and Event Manager*. IBM Redbooks. ISBN 0-7384-3446-9. (Cited on page 121.)

Bullock, S. & Cliff, D. (2004). Complexity and emergent behaviour in ICT systems. Technical Report HP-2004-187, Hewlett-Packard Labs. (Cited on pages 3, 8, 52, and 69.)

Bundesamt für Sicherheit in der Informationstechnik (2003). Project Valikrypt (Validation und Verifikation von kryptographischen Sicherheitsprotokollen unter Verwendung formaler Analysemethoden). https://www.bsi.bund.de/DE/Themen/weitereThemen/Protokollananlyse/valikrypt.html. [Online; accessed 13-Oct-2013]. (Cited on page 44.)

Buttner, A. & Ziring, N. (2009). Common Platform Enumeration (CPE) - Specification. http://cpe.mitre.org/specification/2.1/cpe-specification_2.1.pdf. [Online; accessed 15-Oct-2013]. (Cited on page 89.)

Cederquist, J. & Dashti, M. T. (2011). Complexity of fairness constraints for the dolev-yao attacker model. In Chu, W. C., Wong, W. E., Palakal, M. J., & Hung, C.-C. (Eds.), *SAC*, (pp. 1502–1509). ACM. (Cited on page 59.)

Christey, S. & Martin, R. A. (2007). Vulnerability Type Distributions in CVE. http://cwe.mitre.org/documents/vuln-trends/index.html. [Online; accessed 14-Oct-2013]. (Cited on page 88.)

Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., & Tacchella, A. (2002). Nusmv 2: An opensource tool for symbolic model checking. In Brinksma, E. & Larsen, K. G. (Eds.), *CAV*, volume 2404 of *Lecture Notes in Computer Science*, (pp. 359–364). Springer. (Cited on page 59.)

Clark, D. D. & Wilson, D. R. (1987). A comparison of commercial and military computer security models. In *In Proceedings 1987 IEEE Symposium on Security and Privacy*, (pp. 184–195). (Cited on page 89.)

Clarke, E. M., Talupur, M., & Veith, H. (2006). Environment abstraction for parameterized verification. In Emerson, E. A. & Namjoshi, K. S. (Eds.), *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, (pp. 126–141). Springer. (Cited on page 62.)

Clarke, Jr., E. M., Grumberg, O., & Peled, D. A. (1999). *Model checking*. Cambridge, MA, USA: MIT Press. (Cited on pages 6, 16, 39, and 59.)

Clarkson, M. R. & Schneider, F. B. (2008). Hyperproperties. *Computer Security Foundations Symposium, IEEE*, *0*, 51–65. (Cited on page 59.)

Cleary, F. (2011a). Effectsplus 1st cluster event. Technical report, Waterford institute Of Technology. (Cited on page 129.)

Cleary, F. (2011b). Effectsplus 2nd cluster event. Technical report, Waterford institute Of Technology. (Cited on page 129.)

Cleaveland, R., Parrow, J., & Steffen, B. (1993). The concurrency workbench: A semantics-based tool for the verification of finite-state systems. In *TOPLAS 15*, (pp. 36–72). (Cited on page 58.)

Coppolino, L., D'Antonio, S., Romano, L., & Spagnuolo, G. (2010). An intrusion detection system for critical information infrastructures using wireless sensor network technologies. In *Critical Infrastructure (CRIS), 2010 5th International Conference on*, (pp. 1 –8). (Cited on page 61.)

Coppolino, L., Jäger, M., Kuntze, N., & Rieke, R. (2012). A Trusted Information Agent for Security Information and Event Management. In *ICONS 2012, The Seventh International Conference on Systems, February 29 - March 5, 2012 - Saint Gilles, Reunion Island* (pp. 6–12). IARIA. (Cited on pages 18, 128, 265, and 444.)

CSP EU FORUM (2013). Cyber Security & Privacy EU Forum 2013. `https://www.cspforum.eu/2013/programme/presentations-day-2`. [Online; accessed 16-Oct-2013]. (Cited on page 129.)

Cuppens, F., Cuppens-Boulahia, N., Sans, T., & Miège, A. (2004). A formal approach to specify and deploy a network security policy. In *Second Workshop on Formal Aspects in Security and Trust (FAST)*. (Cited on pages 72 and 90.)

Dan, G., Sandberg, H., Ekstedt, M., & Björkman, G. (2012). Challenges in power system information security. *IEEE Security & Privacy*, *10*(4), 62–70. (Cited on pages 87 and 121.)

Debar, H., Curry, D., & Feinstein, B. (2007). The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765 (Experimental). (Cited on page 120.)

Debar, H., Kheir, N., Cuppens-Boulahia, N., & Cuppens, F. (2010). Service dependencies in information systems security. In Kotenko, I. V. & Skormin, V. A. (Eds.), *MMM-ACNS*, volume 6258 of *Lecture Notes in Computer Science*, (pp. 1–20). Springer. (Cited on page 88.)

Delgado, N., Gates, A., & Roach, S. (2004). A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on Software Engineering*, *30*(12), 859–872. (Cited on page 118.)

Deming, W. E. (1993). *The new economics for industry, government, education / W. Edwards Deming*. Massachusetts Institute of Technology, Center for Advanced Engineering Study, Cambridge, MA :. (Cited on pages 14 and 31.)

Derepas, F. & Gastin, P. (2001). Model checking systems of replicated processes with SPIN. In Dwyer, M. B. (Ed.), *Proceedings of the 8th International SPIN Workshop on Model Checking Software (SPIN'01)*, volume 2057 of *Lecture Notes in Computer Science*, (pp. 235–251)., Toronto, Canada. Springer. (Cited on page 62.)

Deutsche Krankenhaus Gesellschaft (2008). Übersicht Gesundheitskarte Version 2008-03a. Technical report, Deutsche Krankenhaus Gesellschaft. [Online; accessed 14-Jun-2013]. (Cited on page 133.)

Dijkman, R. M. (2008). Diagnosing differences between business process models. In *Business Process Management (BPM 2008)*, volume

5240 of *LNCS*, (pp. 261–277). Springer. (Cited on page 118.)

Dijkman, R. M., Dumas, M., & Ouyang, C. (2008). Semantics and analysis of business process models in BPMN. *Information and Software Technology*, *50*(12), 1281–1294. (Cited on page 118.)

Döhring, M., Zimmermann, B., & Karg, L. (2011). Flexible workflows at design- and runtime using BPMN2 adaptation patterns. In *Business Information Systems (BIS 2011)*, volume 87 of *LNBIP* (pp. 25–36). Springer. (Cited on page 97.)

Durham, D., Boyle, J., Cohen, R., Herzog, S., Rajan, R., & Sastry, A. (2000). The COPS (Common Open Policy Service) Protocol. RFC 2748 (Proposed Standard). Updated by RFC 4261. (Cited on page 11.)

Edge, K. S., Raines, R. A., Grimaila, M. R., Baldwin, R. O., Bennington, R. W., & Reuter, C. E. (2007). The use of attack and protection trees to analyze security for an online banking system. In *HICSS*, (pp. 144). IEEE Computer Society. (Cited on page 86.)

Eichler, J. & Rieke, R. (2011). Model-based Situational Security Analysis. In *Proceedings of the 6th International Workshop on Models@run.time at the ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (MODELS 2011)*, volume 794 of *CEUR Workshop Proceedings* (pp. 25–36). RWTH Aachen. (Cited on pages 24, 100, 106, 112, and 389.)

Eilenberg, S. (1974). *Automata, Languages and Machines*, volume A. New York: Academic Press. (Cited on page 44.)

Emerson, E. A. (1990). Temporal and modal logic. In van Leeuwen, J. (Ed.), *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, (pp. 995–1072). Elsevier. (Cited on page 39.)

Esper contributors and EsperTech Inc. (2012). Esper – Complex Event Processing. `http://esper.codehaus.org/`. [Online; accessed 16-Sep-2012]. (Cited on pages 126 and 404.)

Evesti, A., Ovaska, E., & Savola, R. (2009). From security modelling to run-time security monitoring. In *European Workshop on Security in Model Driven Architecture (SECMDA 2009)*, (pp. 33–41). CTIT. (Cited on page 118.)

Fabian, B., Gürses, S., Heisel, M., Santen, T., & Schmidt, H. (2010). A comparison of security requirements engineering methods. *Requirements engineering*, *15*(1), 7–40. (Cited on page 61.)

Fayyad, U. M., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, *17*(3), 37–54. (Cited on page 118.)

Fenz, S. (2010). Ontology-based generation of it-security metrics. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, (pp. 1833–1839)., New York, NY, USA. ACM. (Cited on page 120.)

Ferraiolo, D. & Kuhn, R. (1992). Role-based access controls. (Cited on page 89.)

Ferraiolo, D. F., Kuhn, D. R., & Chandramouli, R. (2003). *Role-Based Access Control*. Computer Security Series. Boston: Artech House. (Cited on page 89.)

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, R., & Chandramouli, R. (2001). Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, *4*(3), 224–274. http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf. (Cited on page 89.)

Festag, A., Noecker, G., Strassberger, M., Lübke, A., Bochow, B., Torrent-Moreno, M., Schnaufer, S., Eigner, R., Catrinescu, C., & Kunisch, J. (2008). NoW—network on wheels: Project objectives, technology and achievements. In *Proceedings of the 6th International Workshop on Intelligent Transportation*. TU Hamburg. (Cited on pages 13 and 133.)

Firesmith, D. (2003). Engineering security requirements. *Journal of Object Technology*, *2*(1), 53–68. (Cited on pages 18, 60, 66, and 240.)

FIRST.org, Inc. (2013). Common Vulnerability Scoring System. `http://www.first.org/cvss`. [Online; accessed 13-Oct-2013]. (Cited on pages 73 and 89.)

Fisler, K., Krishnamurthi, S., Meyerovich, L. A., & Tschantz, M. C. (2005). Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, (pp. 196–205)., New York, NY, USA. ACM. (Cited on page 89.)

Frankova, G., Seguran, M., Gilcher, F., Trabelsi, S., Dörflinger, J., & Aiello, M. (2011). Deriving business processes with service level agreements from early requirements. *Journal of Systems and Software*, *84*(8), 1351–1363. (Cited on page 117.)

Fraunhofer SIT (2009). *Simple Homomorphism Verification Tool – Manual*. Darmstadt: Fraunhofer Institute for Secure Information Technology SIT. (Cited on pages 16 and 37.)

Fraunhofer SIT (2011). Project EVITA (E-safety Vehicle Intrusion proTected Applications). `http://www.evita-project.org/`. [Online; accessed 13-Oct-2013]. (Cited on pages 13, 48, 87, and 133.)

Fuchs, A. & Rieke, R. (2009). Identification of authenticity requirements in systems of systems by functional security analysis. In *Workshop on Architecting Dependable Systems (WADS 2009), in Proceedings of the 2009 IEEE/IFIP Conference on Dependable Systems and Networks, Supplemental Volume*. (Cited on pages 66, 68, and 239.)

Fuchs, A. & Rieke, R. (2010). Identification of Security Requirements in Systems of Systems by Functional Security Analysis. In A. Casimiro, R. de Lemos, & C. Gacek (Eds.), *Architecting Dependable Systems VII*, volume 6420 of *Lecture Notes in Computer Science* (pp. 74–96). Springer. (Cited on pages 17 and 239.)

Gaber, C., Hemery, B., Achemlal, M., Pasquet, M., & Urien, P. (2013). Synthetic logs generator for fraud detection in mobile transfer services. In *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems (CTS2013)*. (Cited on pages 112, 128, and 444.)

gematik (2007a). Einführung der Gesundheitskarte – Fachkonzept Versichertenstammdatenmanagement (VSDM), Version 2.2.0. Spezifikation, gematik. (Cited on page 45.)

gematik (2007b). Übergreifendes Sicherheitskonzept der Telematikinfrastruktur, Version 1.9.0. Spezifikation, gematik. (Cited on pages 7, 45, and 52.)

gematik (2009). Pressemitteilung: eGK besteht Online-Test. (Cited on page 7.)

Gerlach, M. (2005). Trusted Network on Wheels . *ERCIM News*, (63), 32–33. (Cited on pages 6 and 52.)

Gerth, R., Peled, D., Vardi, M. Y., & Wolper, P. (1996). Simple on-the-fly automatic verification of linear temporal logic. In Dembinski, P. & Sredniawa, M. (Eds.), *Protocol Specification, Testing, and Verification XV '95*, (pp. 3–18). Chapman & Hall. (Cited on pages 16, 40, 43, and 59.)

Giorgini, P., Massacci, F., Mylopoulos, J., & Zannone, N. (2004). Requirements engineering meets trust management: Model, methodology, and reasoning. In *In Proc. of iTrust 04, LNCS 2995*, (pp. 176–190). Springer-Verlag. (Cited on page 61.)

Godefroid, P. (1997). Model checking for programming languages using verisoft. In *Proceedings of the 24th ACM symposium on principles of programming languages*, (pp. 174–186). ACM Press. (Cited on page 45.)

Grimm, R. & Ochsenschläger, P. (2001). Binding Cooperation. A Formal Model for Electronic Commerce. *Computer Networks*, *37, Issue 2*, 171–193. Preliminary version `http://sit.sit.fraunhofer.de/smv/publications/download/GMD_report_96.pdf`. (Cited on page 7.)

Gürgens, S., Ochsenschläger, P., & Rudolph, C. (2002). Authenticity and provability - a formal framework. In *Infrastructure Security Conference InfraSec 2002*, volume 2437 of *LNCS*, (pp. 227–245). Springer. (Cited on pages 48 and 59.)

Gürgens, S., Ochsenschläger, P., & Rudolph, C. (2005). On a formal framework for security properties. *Computer Standards & Interfaces*, *27*, 457–466. (Cited on pages 18 and 59.)

Guttman, J. D. & Herzog, A. L. (2005). Rigorous automated network security management. *International Journal of Information Security*, *4*(1-2), 29–48. (Cited on page 89.)

Guttman, J. D., Herzog, A. L., & Ramsdell, J. D. (2003). Information flow in operating systems: Eager formal methods. IFIP WG 1.7

Workshop on Issues in the Theory of Security. (Cited on page 59.)

Haidar, D. A., Cuppens-Boulahia, N., Cuppens, F., & Debar, H. (2006). An extended rbac profile of xacml. In Juels, A., Damiani, E., & Gabillon, A. (Eds.), *SWS*, (pp. 13–22). ACM. (Cited on page 90.)

Haley, C. B., Laney, R. C., Moffett, J. D., & Nuseibeh, B. (2008). Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Software Eng.*, *34*(1), 133–153. (Cited on page 60.)

Halpern, J. Y. (2003). *Reasoning about Uncertainty*. The MIT Press. (Cited on page 131.)

Harrison, M. A., Ruzzo, W. L., & Ullman, J. D. (1975). On protection in operating systems. In *SOSP '75: Proceedings of the fifth ACM symposium on Operating systems principles*, (pp. 14–24)., New York, NY, USA. ACM Press. (Cited on page 89.)

Hartel, P., Butler, M., Currie, A., Henderson, P., Leuschel, M., Martin, A., Smith, A., Ultes-Nitsche, U., & Walters, B. (1999). Questions and Answers About Ten Formal Methods. In *Proc. 4th Int. Workshop on Formal Methods for Industrial Critical Systems*, volume II, (pp. 179–203)., Pisa, Italy. ERCIM, STAR/CNR. (Cited on pages 16 and 58.)

Hatebur, D. & Heisel, M. (2009). A foundation for requirements analysis of dependable software. In *Proceedings of the International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, LNCS 5775, (pp. 311–325). Springer. (Cited on page 60.)

Hatebur, D., Heisel, M., & Schmidt, H. (2008). Analysis and component-based realization of security requirements. In *Proceedings of the International Conference on Availability, Reliability and Security (AReS)*, (pp. 195–203). IEEE Computer Society. (Cited on page 60.)

Hawley, M., Howard, P., Koelle, R., & Saxton, P. (2013). Collaborative security management: Developing ideas in security management for air traffic control. In *Proceedings of 2013 International Conference on Availability, Reliability and Security, ARES 2013* (pp. 808–806). IEEE Computer Society. (Cited on page 6.)

Heinemann, A., Oetting, J., Peters, J., Rieke, R., Rochaeli, T., Ruppert, M., Steinemann, B., & Wolf, R. (2006). Enforcement of security policies within the sicari-platform. Technical Report PF5, SicAri Consortium. (Cited on page 361.)

Herfert, M., Schmidt, A. U., Ochsenschläger, P., Repp, J., Rieke, R., Schmucker, M., Vettermann, S., Böttge, U., Escaleira, C., & Rüdiger, D. (2004). Implementierung von Security Policies in offenen Telekollaborationen. In Horster, P. (Ed.), *D-A-CH Security 2004*, (pp. 37–39). Syssec. (Cited on pages 44 and 68.)

Herrmann, D. (2007). *Complete Guide to Security And Privacy Metrics: Measuring Regulatory Compliance, Operational Resilience, and*

*Roi*. Auerbach Publishers, Incorporated. (Cited on page 89.)

Hopcroft, J. E. & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation* (first ed.). Reading, Mass.: Addison-Wesley. (Cited on page 10.)

Hutchison, A., Dennie, K., Khan, H., et al. (2013). D2.2.1 - Tool Adaptation. Technical report, FP7-257475 MASSIF European project. (Cited on pages 134 and 135.)

Hutchison, A. & Rieke, R. (2011). Management of Security Information and Events in Future Internet. In *2011 Workshop on Cyber Security and Global Affairs, Budapest*. (Cited on pages 129 and 421.)

Hutchison, A. & Rieke, R. (2012). Measuring Progress in Cyber-Security: An Open Architecture for Security Measurement Consolidation. In *2012 Workshop on Cyber Security and Global Affairs and Global Security Forum, Barcelona*. (Cited on pages 119, 129, and 403.)

Ingols, K., Chu, M., Lippmann, R., Webster, S., & Boyer, S. (2009). Modeling modern network attacks and countermeasures using attack graphs. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, (pp. 117–126). (Cited on pages 21 and 88.)

Ingols, K., Lippmann, R., & Piwowarski, K. (2006). Practical attack graph generation for network defense. In *ACSAC*, (pp. 121–130). IEEE Computer Society. (Cited on page 88.)

Innerhofer-Oberperfler, F. & Breu, R. (2006). Using an enterprise architecture for it risk management. In Eloff, J. H. P., Labuschagne, L., Eloff, M. M., & Venter, H. S. (Eds.), *ISSA*, (pp. 1–12). ISSA, Pretoria, South Africa. (Cited on page 120.)

Intelligent Transportation Systems Committee of the IEEE Vehicular Technology Society (2006). IEEE Trial-Use Standard for Wireless Access in Vehicular Environments– Security Services for Applications and Management Messages. *IEEE Std 1609.2-2006*. (Cited on page 133.)

Ip, C. N. & Dill, D. L. (1999). Verifying Systems with Replicated Components in Murφ. *Formal Methods in System Design*, *14*(3), 273–310. (Cited on page 62.)

Iso Iec (2005). ISO/IEC 27001:2005 - Information technology - Security techniques - Information security management systems - Requirements. *ISOIEC*. (Cited on page 3.)

Iso Iec (2009). ISO/IEC 27004:2009 - Information technology - Security techniques - Information security management - Measurement. *ISOIEC*. (Cited on page 120.)

Jansen, W. A. (2009). *Directions in security metrics research*. Gaithersburg, MD: National Institute of Standards and Technology. (Cited on page 89.)

Jantzen, M. (1985). Extending Regular Expressions with Iterated Shuffle. *Theor. Comput. Sci.*, *38*, 223–247. (Cited on pages 62 and 100.)

Jaquith, A. (2007). *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional. (Cited on page 89.)

Jedrzejowicz, J. & Szepietowski, A. (2001). Shuffle languages are in P. *Theor. Comput. Sci.*, *250*(1-2), 31–53. (Cited on page 62.)

Jha, S., Sheyner, O., & Wing, J. M. (2002). Two formal analyses of attack graphs. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada,* (pp. 49–63). IEEE Computer Society. (Cited on page 87.)

Kaindl, H., Jäntti, M., Mannaert, H., Nakamatsu, K., & Rieke, R. (2012). Requirements Engineering for Software vs. Systems in General. In *ICONS 2012, The Seventh International Conference on Systems, February 29 - March 5, 2012 - Saint Gilles, Reunion Island* (pp. 190–192). IARIA. (Cited on page 68.)

Kalam, A. A. E. & Deswarte, Y. (2006). Multi-orbac: a new access control model for distributed, heterogeneous and collaborative systems. In *8th IEEE International Symposium on Systems and Information Security (SSI 2006)*. (Cited on page 90.)

Kazhamiakin, R., Pistore, M., & Santuari, L. (2006). Analysis of communication models in web service compositions. In *World Wide Web (WWW 2006)*, (pp. 267–276). ACM. (Cited on page 118.)

Khan, M. A., Banerjee, M., & Rieke, R. (2013). An update logic for information systems. *International Journal of Approximate Reasoning*, (0), –. (Cited on page 68.)

Kheir, N., Cuppens-Boulahia, N., Cuppens, F., & Debar, H. (2010). A service dependency model for cost-sensitive intrusion response. In Gritzalis, D., Preneel, B., & Theoharidou, M. (Eds.), *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, (pp. 626–642). Springer. (Cited on page 88.)

Kissel, R. (2013). Glossary of key information security terms. NIST Interagency Reports NIST IR 7298 Revision 2, National Institute of Standards and Technology. (Cited on page 3.)

Kordy, B., Mauw, S., Radomirović, S., & Schweitzer, P. (2011). Foundations of attack-defense trees. In *Proceedings of the 7th International conference on Formal aspects of security and trust*, FAST'10, (pp. 80–95)., Berlin, Heidelberg. Springer-Verlag. (Cited on page 86.)

Kordy, B., Pietre-Cambacedes, L., & Schweitzer, P. (2013). Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *CoRR*, *abs/1303.7397*. (Cited on page 88.)

Kotenko, I. & Chechulin, A. (2012). Attack modeling and security evaluation in SIEM systems. In *International Transactions on Systems Science and Applications*, volume 8. SIWN Press. (Cited on pages 11, 21, 81, 82, and 88.)

Kotenko, I., Chechulin, A., & Doynikova, E. (2011). Analytical attack modeling. Deliverable D4.3.1, MASSIF Project. (Cited on page 88.)

Kotenko, I., Chechulin, A., & Novikova, E. (2012). Attack Modelling and Security Evaluation for Security Information and Event Management. In Samarati, P., Lou, W., & Zhou, J. (Eds.), *SECRYPT*, (pp. 391–394). SciTePress. (Cited on pages 88 and 120.)

Kotenko, I., Saenko, I., Polubelova, O., & Doynikova, E. (2013). The ontology of metrics for security evaluation and decision support in siem systems. In *Proceedings of 2013 International Conference on Availability, Reliability and Security, ARES 2013* (pp. 638–645). IEEE Computer Society. (Cited on page 89.)

Kotenko, I. & Stepashkin, M. (2006). Analyzing Network Security using Malefactor Action Graphs. *International Journal of Computer Science and Network Security*, *6*. (Cited on page 87.)

Kotenko, I. & Ulanov, A. (2007). Multi-agent Framework for Simulation of Adaptive Cooperative Defense against Internet Attacks. In *In Proceedings of International Workshop on Autonomous Intelligent Systems: Agents and Data Mining (AIS-ADM-07). Lecture Notes in Artificial Intelligence, Vol.4476*. (Cited on page 87.)

Kotenko, I. V., Stepashkin, M., & Doynikova, E. (2011). Security analysis of information systems taking into account social engineering attacks. In *PDP*, (pp. 611–618). (Cited on page 86.)

Kuntze, N., Rieke, R., Diederich, G., Sethmann, R., Sohr, K., Mustafa, T., & Detken, K.-O. (2010). Secure mobile business information processing. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, (pp. 672 –678)., Hongkong, China. IEEE/IFIP. (Cited on page 95.)

Kurshan, R. P. (1994). *Computer-Aided Verification of Coordinating Processes* (first ed.). Princeton, New Jersey: Princeton University Press. (Cited on pages 39 and 58.)

Lakhnech, Y., Bensalem, S., Berezin, S., & Owre, S. (2001). Incremental verification by abstraction. In Margaria, T. & Yi, W. (Eds.), *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, (pp. 98–112). Springer. (Cited on page 62.)

Landwehr, C. E. (1981). Formal models for computer security. *ACM Comput. Surv.*, *13*(3), 247–278. (Cited on page 6.)

Laprie, J.-C. (1995). Dependable computing: concepts, limits, challenges. In *Proceedings of the Twenty-Fifth international conference on Fault-tolerant computing*, FTCS'95, (pp. 42–54)., Washington, DC, USA. IEEE Computer Society. (Cited on page 4.)

Liu, L., Yu, E., & Mylopoulos, J. (2002). Analyzing security requirements as relationships among strategic actors. In *2nd Symposium on Requirements Engineering for Information Security (SREIS'02)*. (Cited on page 61.)

Llanes, M., Prieto, E., Diaz, R., , Coppolino, L., Sergio, A., Cristaldi, R., Achemlal, M., Gharout, S., Gaber, C., Hutchison, A., & Dennie, K. (2011). Scenario requirements (public version). Deliverable

D2.1.1, FP7-257475 MASSIF European project. (Cited on pages 49 and 50.)

Luallen, M. E. (2011). Managing Insiders in Utility Control Environments. A SANS Whitepaper in Association with SANS SCADA Summits, Q1, 2011, SANS. (Cited on page 114.)

Maggi, F. M., Montali, M., Westergaard, M., & van der Aalst, W. M. P. (2011). Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *Business Process Management (BPM 2011)*, volume 6896 of *LNCS*, (pp. 132–147). Springer. (Cited on page 119.)

Manadhata, P. K. & Wing, J. M. (2011). A formal model for a system's attack surface. *Moving Target Defense*, 1–28. (Cited on page 88.)

Martin, B. et al. (2013). Open Sourced Vulnerability Database. `http://www.osvdb.org/`. [Online; accessed 13-Oct-2013]. (Cited on page 88.)

Martin, R. A. & Barnum, S. (2008). Common weakness enumeration (CWE) status update. *j-SIGADA-LETTERS*, *28*(1), 88–91. (Cited on pages 83 and 88.)

Martinelli, F., Mori, P., & Vaccarelli, A. (2005). Towards continuous usage control on grid computational services. In *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS 2005)*. IEEE. (Cited on page 118.)

Massart, T. & Meuter, C. (2006). Efficient online monitoring of LTL properties for asynchronous distributed systems. Technical report, Université Libre de Bruxelles. (Cited on page 118.)

MASSIF project consortium (2013a). Acquisition and evaluation of the results. Deliverable D2.3.3, FP7-257475 MASSIF European project. (Cited on page 134.)

MASSIF project consortium (2013b). Project MASSIF (MAnagement of Security information and events in Service InFrastructures). `http://www.massif-project.eu/`. [Online; accessed 13-Oct-2013]. (Cited on pages 14 and 23.)

Mauw, S. & Oostdijk, M. (2006). Foundations of attack trees. In *Proceedings of the 8th international conference on Information Security and Cryptology*, ICISC'05, (pp. 186–198)., Berlin, Heidelberg. Springer-Verlag. (Cited on page 86.)

McCoy, D. W. (2002). Business Activity Monitoring: Calm Before the Storm. `http://www.gartner.com/resources/105500/105562/105562.pdf`. [Online; accessed 15-Jan-2014]. (Cited on page 117.)

Mead, N. R. (2007). How To Compare the Security Quality Requirements Engineering (SQUARE) Method with Other Methods . Technical Report CMU/SEI-2007-TN-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. (Cited on pages 17 and 60.)

Mead, N. R. & Hough, E. D. (2006). Security requirements engineering for software systems: Case studies in support of software engineering education. In *CSEET '06: Proceedings of the 19th Conference on Software Engineering Education & Training*, (pp. 149–158)., Washington, DC, USA. IEEE Computer Society. (Cited on pages 17 and 60.)

Melik-Merkumians, M., Moser, T., Schatten, A., Zoitl, A., & Biffl, S. (2010). Knowledge-based runtime failure detection for industrial automation systems. In *Workshop Models@run.time*, (pp. 108–119). CEUR. (Cited on pages 117 and 120.)

Mell, P., Scarfone, K., & Romanosky, S. (2007). *CVSS: A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. FIRST: Forum of Incident Response and Security Teams. (Cited on page 89.)

Mellado, D., Blanco, C., Sánchez, L. E., & Fernández-Medina, E. (2010). A systematic review of security requirements engineering. *Computer Standards & Interfaces*, *32*(4), 153–165. (Cited on page 61.)

Mellado, D., Fernández-Medina, E., & Piattini, M. (2006). Applying a security requirements engineering process. In *Proceedings of the 11th European conference on Research in Computer Security*, ESORICS'06, (pp. 192–206)., Berlin, Heidelberg. Springer-Verlag. (Cited on page 60.)

Mellado, D., Fernández-Medina, E., & Piattini, M. (2007). A common criteria based security requirements engineering process for the development of secure information systems. *Computer Standards & Interfaces*, *29*(2), 244–253. (Cited on pages 17 and 60.)

Mendling, J. (2008). *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, volume 6 of *LNBIP*. Springer. (Cited on page 118.)

Milner, R. (1989). *Communication and Concurrency*. International Series in Computer Science. NY: Prentice Hall. (Cited on page 62.)

Milner, R. (1999). *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press. (Cited on page 36.)

Mitchell, C. (2005). *Trusted Computing*. Iet. (Cited on page 62.)

MITRE Corporation (2013a). Common Attack Pattern Enumeration and Classification. `http://capec.mitre.org/`. [Online; accessed 13-Oct-2013]. (Cited on page 87.)

MITRE Corporation (2013b). Common Vulnerabilities and Exposures. `http://cve.mitre.org/`. [Online; accessed 13-Oct-2013]. (Cited on pages 73 and 88.)

MITRE Corporation (2013c). Common Weakness Scoring System. `http://cwe.mitre.org/cwss/`. [Online; accessed 13-Oct-2013]. (Cited on page 89.)

MITRE Corporation (2013d). The Common Weaknesses Enumeration (CWE) Initiative. `http://cwe.mitre.org/`. [Online; accessed 13-

Oct-2013]. (Cited on pages 83 and 88.)

Morin, B., Mé, L., Debar, H., & Ducassé, M. (2002). M2d2: a formal data model for ids alert correlation. In *Proceedings of the 5th international conference on Recent advances in intrusion detection*, RAID'02, (pp. 115–137)., Berlin, Heidelberg. Springer-Verlag. (Cited on pages 71 and 88.)

Morin, B., Mouelhi, T., Fleurey, F., Le Traon, Y., Barais, O., & Jézéquel, J.-M. (2010). Security-driven model-based dynamic adaptation. In *Automated Software Engineering (ASE 2010)*, (pp. 205–214). ACM. (Cited on page 117.)

Moses, T. (2005). eXtensible Access Control Markup Language (XACML), Version 2.0. Technical report, OASIS Standard. (Cited on page 89.)

Motahari-Nezhad, H. R., Saint-Paul, R., Casati, F., & Benatallah, B. (2011). Event correlation for process discovery from web service interaction logs. *The VLDB Journal*, *20*(3), 417–444. (Cited on page 101.)

Myers, B. K., Dutson, G. C., & Sherman, T. (2005). Utilizing Automated Monitoring for the Franzen Reservoir Dam Safety Program. In *25th USSD Annual Meeting and Conference Proceedings (2005)*. (Cited on page 61.)

Netjes, M., Reijers, H., & Aalst, W. P. v. d. (2006). Supporting the BPM life-cycle with FileNet. In *Exploring Modeling Methods for Systems Analysis and Design (EMMSAD 2006)*, (pp. 497–508). Namur University Press. (Cited on page 118.)

NetUnion (2003). Project CASENET (Computer-Aided solutions to SEcure electroNic commercE Transactions). `http://www.netunion.com/projects/casenet.php`. [Online; accessed 13-Oct-2013]. (Cited on page 44.)

Nicol, D. M., Sanders, W. H., & Trivedi, K. S. (2004). Model-based evaluation: From dependability to security. *IEEE Trans. Dependable Secur. Comput.*, *1*(1), 48–65. (Cited on page 9.)

Nicolett, M. & Kavanagh, K. M. (2009). Magic Quadrant for Security Information and Event Management. Gartner RAS Core Reasearch Note. (Cited on pages 123 and 379.)

Nicolett, M. & Kavanagh, K. M. (2010). Magic Quadrant for Security Information and Event Management. Gartner Reasearch. (Cited on page 121.)

NIST Computer Security Resource Center (2013a). Common Configuration Enumeration (CCE) Reference Data. `http://nvd.nist.gov/cce.cfm`. [Online; accessed 13-Oct-2013]. (Cited on page 89.)

NIST Computer Security Resource Center (2013b). National Vulnerability Database. `http://nvd.nist.gov/`. [Online; accessed 13-Oct-2013]. (Cited on pages 73 and 88.)

NIST Computer Security Resource Center (2013c). Official Common Platform Enumeration (CPE) Dictionary. `http://nvd.nist.gov/cpe.cfm`. [Online; accessed 13-Oct-2013]. (Cited on page 88.)

Nitsche, U. (1998). *Verification of Co-Operating Systems and Behaviour Abstraction*. PhD thesis, University of Frankfurt, Germany. (Cited on page 43.)

Nitsche, U. & Ochsenschläger, P. (1996). Approximately satisfied properties of systems and simple language homomorphisms. *Information Processing Letters*, *60*, 201–206. (Cited on pages 7, 42, 43, and 59.)

Noel, S., Jacobs, M., Kalapa, P., & Jajodia, S. (2005). Multiple Coordinated Views for Network Attack Graphs. In *IEEE Workshop on Visualization for Computer Security (VizSec'05)*, Los Alamitos, CA, USA. IEEE Computer Society. (Cited on page 88.)

Noel, S. & Jajodia, S. (2004). Managing attack graph complexity through visual hierarchical aggregation. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, (pp. 109–118)., New York, NY, USA. ACM Press. (Cited on page 88.)

Ochsenschläger, P. (1994). Verification of cooperating systems by simple homomorphisms using the product net machine. In Desel, J., Oberweis, A., & Reisig, W. (Eds.), *Workshop: Algorithmen und Werkzeuge für Petrinetze*, (pp. 48–53). Humboldt Universität Berlin. (Cited on page 42.)

Ochsenschläger, P., Repp, J., & Rieke, R. (2000). Abstraction and composition – a verification method for co-operating systems. *Journal of Experimental and Theoretical Artificial Intelligence*, *12*(4), 447–459. (Cited on pages 17, 41, 42, 68, and 195.)

Ochsenschläger, P., Repp, J., & Rieke, R. (2000a). The SH-Verification Tool. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, (pp. 18–22)., Orlando, FL, USA. AAAI Press. (Cited on pages 16, 17, and 196.)

Ochsenschläger, P., Repp, J., & Rieke, R. (2000b). Verification of Cooperating Systems – An Approach Based on Formal Languages. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, (pp. 346–350)., Orlando, FL, USA. AAAI Press. (Cited on pages 68 and 195.)

Ochsenschläger, P., Repp, J., & Rieke, R. (2002). *Simple Homomorphism Verification Tool – Tutorial*. Darmstadt: Fraunhofer Institute for Secure Telecooperation SIT. (Cited on page 16.)

Ochsenschläger, P., Repp, J., Rieke, R., & Nitsche, U. (1998). The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing, The International Journal of Formal Method*, *10*, 381–404. (Cited on pages 6, 16, 17, 35, and 167.)

Ochsenschläger, P. & Rieke, R. (2007). Abstraction Based Verification of a Parameterised Policy Controlled System. In Gorodetsky, V., Kotenko, I., & Skormin, V. A. (Eds.), *Computer Network Security*, volume 1 of *CCIS*. Springer. © Springer. (Cited on pages 17 and 223.)

Ochsenschläger, P. & Rieke, R. (2010). Uniform Parameterisation of Phase Based Cooperations. Technical Report SIT-TR-2010/1, Fraunhofer SIT. (Cited on pages 56, 57, and 62.)

Ochsenschläger, P. & Rieke, R. (2011). Security properties of self-similar uniformly parameterised systems of cooperations. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, (pp. 640–645). (Cited on pages 18 and 273.)

Ochsenschläger, P. & Rieke, R. (2012a). Reliability Aspects of Uniformly Parameterised Cooperations. In *ICONS 2012, The Seventh International Conference on Systems, February 29 - March 5, 2012 - Saint Gilles, Reunion Island* (pp. 25–34). IARIA. (Cited on pages 18 and 281.)

Ochsenschläger, P. & Rieke, R. (2012b). Security requirements for uniformly parameterised cooperations. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, (pp. 288–292). (Cited on pages 43, 68, and 281.)

Ochsenschläger, P. & Rieke, R. (2014). Construction Principles for Well-behaved Scalable Systems. In *ICONS 2014, The Ninth International Conference on Systems, February 23 - 27, 2014 - Nice, France* (pp. 32–39). IARIA. (Cited on pages 57, 58, and 63.)

Ochsenschläger, P., Rieke, R., & Velikova, Z. (2008). Die elektronische Krankenakte - Eine Sicherheitsstrategie. In Horster, P. (Ed.), *DACH Security 2008 - Bestandsaufnahme, Konzepte, Anwendungen, Perspektiven.*, (pp. 90–100). (Cited on pages 21, 90, and 95.)

Parekh, M., Stone, K., & Delborne, J. (2010). Coordinating intelligent and continuous performance monitoring with dam and levee safety management policy. In *Association of State Dam Safety Officials,Proceedings of Dam Safety Conference 2010*. (Cited on page 61.)

Parmelee, M. C. (2010). Toward an ontology architecture for cybersecurity standards. In da Costa, P. C. G. & Laskey, K. B. (Eds.), *STIDS*, volume 713 of *CEUR Workshop Proceedings*, (pp. 116–123). CEUR-WS.org. (Cited on page 88.)

Peled, D. A. (2001). *Software Reliability Methods* (1 ed.). Springer. (Cited on pages 16, 35, 36, and 59.)

Perrin, D. & Pin, J.-E. (2004). *Infinite Words*, volume Pure and Applied Mathematics Vol 141. Elsevier. (Cited on page 42.)

Peters, J. (2013). SicAri Platform. `http://sicari.sourceforge.net/`. [Online; accessed 13-Oct-2013]. (Cited on pages 21, 44, 84, 94, 134, 361, and 362.)

Peters, J., Rieke, R., Rochaeli, T., Steinemann, B., & Wolf, R. (2005). Protocols for policy negotiation. Technical Report PE3, Reportnr.: 05i018-FIGD, SicAri Consortium. (Cited on page 361.)

Peters, J., Rieke, R., Rochaeli, T., Steinemann, B., & Wolf, R. (2007). A Holistic Approach to Security Policies – Policy Distribution with XACML over COPS. In *Proc. of the Second International Workshop on Views On Designing Complex Architectures (VODCA 2006)*, volume 168, (pp. 143–157). Elsevier. (Cited on pages 21, 85, and 362.)

Petri, C. A. (1962). Kommunikation mit Automaten. Dissertation, TH Darmstadt. (Cited on pages 16, 100, and 118.)

Phanse, K. S. (2003). Policy-Based Quality of Service Management in Wireless Ad Hoc Networks. Dissertation, Virginia Polytechnic Institute and State University. (Cited on page 90.)

Philipps-Universität Marburg (2013). Project ACCEPT (Anomaliemanagement in Computersystemen durch Complex Event Processing Technologie). `http://accept-projekt.de/`. [Online; accessed 13-Oct-2013]. (Cited on pages 14 and 121.)

Phillips, C. A. & Swiler, L. P. (1998). A graph-based system for network-vulnerability analysis. In *NSPW '98, Proceedings of the 1998 Workshop on New Security Paradigms, September 22-25, 1998, Charlottsville, VA, USA*, (pp. 71–79). ACM Press. (Cited on page 87.)

Ponnappan, A., Yang, L., Pillai, R., & Braun, P. (2002). A Policy Based QoS Management System for the IntServ/DiffServ Based Internet. In *Proc. of the IEEE 3th International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, (pp. 159ff)., Los Alamitos, CA, USA. IEEE Computer Society. (Cited on page 90.)

Prelude - CS Group (2014). Prelude Security Information and Event Management. http://www.prelude-ids.com. [Online; accessed 12-Jun-2014]. (Cited on pages 120, 126, and 404.)

Prieto, E., Diaz, R., Romano, L., Rieke, R., & Achemlal, M. (2012). MASSIF: A Promising Solution to Enhance Olympic Games IT Security. In C. K. Georgiadis et al. (Eds.), *Global Security, Safety and Sustainability & e-Democracy*, volume 99 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering* (pp. 139–147). Springer Berlin Heidelberg. (Cited on pages 25, 111, and 411.)

Racz, N., Weippl, E., & Seufert, A. (2010). A Frame of Reference for Research of Integrated Governance, Risk and Compliance (GRC). In Decker, B. D. & Schaumüller-Bichl, I. (Eds.), *Communications and Multimedia Security*, volume 6109 of *Lecture Notes in Computer Science*, (pp. 106–117). Springer. (Cited on pages 110, 126, and 404.)

Randell, B. (2003). On failures and faults. In Araki, K., Gnesi, S., & Mandrioli, D. (Eds.), *FME*, volume 2805 of *Lecture Notes in Computer Science*, (pp. 18–39). Springer. (Cited on pages 4, 106, and 122.)

Repp, J. & Rieke, R. (2011). Formal specification of security properties. Deliverable D4.2.1, FP7-257475 MASSIF European project. (Cited on page 14.)

Repp, J. & Rieke, R. (2013). Predictive security analyser. Deliverable D4.2.3, FP7-257475 MASSIF European project. (Cited on page 109.)

Repp, J., Rieke, R., & Steinemann, B. (2005). Evaluierung von Sicherheitszielen auf Basis von Policies. Technical Report PE5, SicAri Consortium. (Cited on page 361.)

Richter, J., Kuntze, N., & Rudolph, C. (2010). Security Digital Evidence. In *2010 Fifth International Workshop on Systematic Approaches to Digital Forensic Engineering*, (pp. 119–130). IEEE. (Cited on page 62.)

Rieke, R. (2002). Projects CASENET and SKe a framework for secure e-government. Invited talk at Telecities 2002 Winter Conference, Sienna, Italy. (Cited on page 68.)

Rieke, R. (2003). Development of formal models for secure e-services. In *Eicar Conference 2003*. (Cited on pages 17 and 203.)

Rieke, R. (2004a). Formale Spezifikation von Zielen und Voraussetzungen. Technical Report PE2, SicAri Consortium. (Cited on page 361.)

Rieke, R. (2004b). Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration. In *In U.E. Gattiker (Ed.), Eicar 2004 Conference CD-rom: Best Paper Proceedings*, Copenhagen. EICAR e.V. (Cited on pages 20 and 293.)

Rieke, R. (2006). Modelling and Analysing Network Security Policies in a Given Vulnerability Setting. In *Critical Information Infrastructures Security, First International Workshop, CRITIS 2006, Samos Island, Greece*, volume 4347 of *LNCS*, (pp. 67–78). Springer. © Springer. (Cited on pages 20 and 327.)

Rieke, R. (2007a). Improving Resilience of Critical Information Infrastructures against Complex Threats. Invited talk at IFIP WG 10.4 Dependable Computing and Fault Tolerance, 51st Meeting, Guadeloupe, France. (Cited on page 95.)

Rieke, R. (2007b). Wie ausführbare Modelle helfen, komplexe Systeme zu verstehen und sicherer zu steuern. Vortragsreihe: Modelle für die Sicherheit und Zuverlässigkeit von Systemen, J.W.Goethe-Universität Frankfurt. (Cited on page 95.)

Rieke, R. (2008a). Abstraction-based analysis of known and unknown vulnerabilities of critical information infrastructures. *International Journal of System of Systems Engineering (IJSSE)*, *1*, 59–77. (Cited on pages 21 and 341.)

Rieke, R. (2008b). Upcoming information security threats - an end-user perspective -. Invited talk at 1st FORWARD Workshop, Goteborg, Sweden. (Cited on page 95.)

Rieke, R. (2009a). Operational models for security and dependability in electronic health systems. In Breu, R., Mitchell, J. C., Sztipanovits, J., & Winter, A. (Eds.), *09073 Abstracts Collection – Model-Based Design of Trustworthy Health Information Systems*, number 09073 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. (Cited on pages 7 and 45.)

Rieke, R. (2009b). ProOnline-VSDD Wissenschaftliche Begleitung der Tests von Online-Prüfung und -Aktualisierung der Versichertenstammdaten der eGK, Arbeitspaket Formale Modellierung und Sicherheitsanalyse der Kommunikationsinfrastruktur, Endbericht - Version 1.1. Technical report, Fraunhofer SIT. (Cited on pages 7, 13, 45, 68, and 133.)

Rieke, R. (2010a). Challenges for Systems of Systems Security Information and Event Management. In *2010 Workshop on Cyber Security and Global Affairs,ETH,Zurich*. (Cited on pages 129 and 421.)

Rieke, R. (2010b). Identification of Security Requirements for Vehicular Communication Systems. CAST-Workshop on Mobile Security for Intelligent Cars, Darmstadt, Germany. (Cited on page 68.)

Rieke, R. (2010c). Management of security information and events in service infrastructures. Talk at the ICT 2010 Effectsplus networking session, Brussels. (Cited on page 129.)

Rieke, R. (2011). SIEM systems of the future. Talk at the Effectsplus Trustworthy ICT Research Roadmap Session Cluster Meeting, Brussels. (Cited on page 129.)

Rieke, R. (2012a). Advanced security monitoring: Challenges, advances, and foundations - The MASSIF project. Talk at Cyber Security & Privacy EU Forum 2012, Berlin, Germany. (Cited on pages 129 and 411.)

Rieke, R. (2012b). Enhancing situational awareness, security and trustworthiness of processes in systems of systems. Invited talk at Second International Workshop 'Scientific Analysis and Policy Support for Cyber Security', St. Petersburg, Russia. (Cited on pages 129 and 411.)

Rieke, R., Coppolino, L., Hutchison, A., Prieto, E., & Gaber, C. (2012). Security and reliability requirements for advanced security event management. In I. Kotenko & V. Skormin (Eds.), *Computer Network Security*, volume 7531 of *Lecture Notes in Computer Science* (pp. 171–180). Springer Berlin Heidelberg. (Cited on pages 25, 111, 112, and 422.)

Rieke, R. & Ebinger, P. (2008). Eine Sicherheitsarchitektur und deren Werkzeuge zur ubiquitären Internetnutzung: SicAri ; Erfolgskontrollbericht. Technical report, Fraunhofer. (Cited on pages 14, 44, 84, 94, 134, 361, and 362.)

Rieke, R. & Giot, R. (2013). Predictive security analysis - concepts, implementation, first results in industrial scenario. Talk at Cyber Security & Privacy EU Forum 2013, Brussels, Belgium. (Cited on pages 129 and 433.)

Rieke, R., Perez, E. P., & Debar, H. (2012). Research and roadmapping report year 2. Deliverable D1.2.2, FP7-257475 MASSIF European project. (Cited on page 129.)

Rieke, R., Perez, E. P., Debar, H., & Gharout, S. (2011). Research and roadmapping report year 1. Deliverable D1.2.1, FP7-257475 MASSIF European project. (Cited on page 129.)

Rieke, R., Perez, E. P., Debar, H., Hutchison, A., Achemlal, M., & Jimenez, R. (2013). Research and Roadmapping Report Year 3. Deliverable D1.2.3, FP7-257475 MASSIF European project. (Cited on page 129.)

Rieke, R., Prieto, E., Diaz, R., Debar, H., & Hutchison, A. (2012). Challenges for advanced security monitoring – the MASSIF project. In S. Fischer-Hübner, S. Katsikas, & G. Quirchmayr (Eds.), *Trust, Privacy and Security in Digital Business*, volume 7449 of *Lecture Notes in Computer Science* (pp. 222–223). Springer Berlin / Heidelberg. (Cited on pages 112, 128, 129, and 444.)

Rieke, R., Repp, J., & Zhdanova, M. (2012). Process model and dynamic simulation and analysis modelling framework. Deliverable D4.2.2, FP7-257475 MASSIF European project. (Cited on page 100.)

Rieke, R., Repp, J., Zhdanova, M., & Eichler, J. (2014). Monitoring security compliance of critical processes. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22th Euromicro International Conference on*, (pp. 525–560). IEEE Computer Society. (Cited on pages 24, 25, 106, 108, and 443.)

Rieke, R., Schütte, J., & Hutchison, A. (2012). Architecting a security strategy measurement and management system. In *Proceedings of the Workshop on Model-Driven Security*, MDsec '12, (pp. 2:1–2:6)., New York, NY, USA. ACM. (Cited on pages 24, 109, and 403.)

Rieke, R. & Steinemann, B. (2007). Projektbericht SimuSec-NoW, Ein Modell zur Simulation einer Warnmeldung-Anwendung für Fahrzeuge mit NoW-Technologie - Studie im Auftrag von BMW im Rahmen des BMBF Projektes Network on Wheels (NoW) -. Technical report, Fraunhofer SIT. (Cited on pages 13 and 133.)

Rieke, R. & Stoynova, Z. (2010). Predictive security analysis for event-driven processes. In *Computer Network Security*, volume 6258 of *LNCS* (pp. 321–328). Springer. (Cited on pages 23, 99, and 379.)

Rieke, R., Zhdanova, M., Repp, J., Giot, R., & Gaber, C. (2013). Fraud detection in mobile payment utilizing process behavior analysis.

In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, (pp. 662–669). IEEE Computer Society. (Cited on pages 23, 25, 114, and 433.)

Ritchey, R. W. & Ammann, P. (2000). Using model checking to analyze network vulnerabilities. In *IEEE Symposium on Security and Privacy*, (pp. 156–165). IEEE Computer Society. (Cited on page 87.)

Romano, L., Antonio, S. D., Formicola, V., & Coppolino, L. (2012). Enhancing SIEM technology to protect critical infrastructures. In *CRITIS 2012, the seventh CRITIS Conference on Critical Information Infrastructures Security*. (Cited on pages 112, 128, and 444.)

Roy, A., Kim, D. S., & Trivedi, K. S. (2012). Attack countermeasure trees (act): towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, *5*(8), 929–943. (Cited on page 86.)

Rozinat, A. & van der Aalst, W. (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems*, *33*(1), 64 – 95. (Cited on page 119.)

Rozinat, A., Wynn, M. T., van der Aalst, W. M. P., ter Hofstede, A. H. M., & Fidge, C. J. (2009). Workflow simulation for operational decision support. *Data & Knowledge Engineering*, *68*(9), 834–850. (Cited on page 118.)

Ruddle, A., Ward, D., Weyl, B., Idrees, S., Roudier, Y., Friedewald, M., Leimbach, T., Fuchs, A., Gürgens, S., Henniger, O., Rieke, R., Ritscher, M., Broberg, H., Apvrille, L., Pacalet, R., & Pedroza, G. (2009). Security requirements for automotive on-board networks based on dark-side scenarios. EVITA Deliverable D2.3, EVITA project. (Cited on pages 14, 48, 87, and 134.)

Sakarovitch, J. (2009). *Elements of Automata Theory*. Cambridge University Press. (Cited on pages 23, 33, and 41.)

Sandhu, R. (1998). Role activation hierarchies. In *Proceedings of the third ACM workshop on Role-based access control*. ACM Press. (Cited on page 89.)

Sarbinowski, H. (2002). Project SKE (Durchgängige Sicherheitskonzeption mit dynamischen Kontrollmechanismen für e-Service-Prozesse). `http://www.egov-zentrum.fraunhofer.de/projects_extern_detail.php3?sessionid=775647c320baeda188711eae0ecca6e7&id=12`. (Cited on pages 14 and 134.)

Scarfone, K. & Mell, P. (2009). An analysis of cvss version 2 vulnerability scoring. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, (pp. 516–525)., Washington, DC, USA. IEEE Computer Society. (Cited on page 89.)

Schiefer, J., Rozsnyai, S., Rauscher, C., & Saurer, G. (2007). Event-driven rules for sensing and responding to business situations. In

Jacobsen, H.-A., Mühl, G., & Jaeger, M. A. (Eds.), *DEBS*, volume 233 of *ACM International Conference Proceeding Series*, (pp. 198–205). ACM. (Cited on page 120.)

Schiffmann, M. (2005). A Complete Guide to the Common Vulnerability Scoring System (CVSS). http://www.first.org/cvss/cvss-guide.html. (Cited on page 89.)

Schneider, F. B. (2000). Enforceable security policies. *ACM Transactions on Information and System Security*, *3*(1), 30–50. (Cited on page 118.)

Schneider, S. (1996). Security Properties and CSP. In *IEEE Symposium on Security and Privacy*, (pp. 174–187). IEEE Computer Society. (Cited on pages 33 and 52.)

Schneier, B. (1999). Attack Trees: Modeling Security Threats. *Dr. Dobb's Journal*. (Cited on page 86.)

Schneier, B. (2000). *Secrets and Lies: Digital Security in a Networked World*. John Wiley and Sons. (Cited on page 86.)

Schütte, J., Rieke, R., & Winkelvos, T. (2012). Model-based security event management. In I. Kotenko & V. Skormin (Eds.), *Computer Network Security*, volume 7531 of *Lecture Notes in Computer Science* (pp. 181–190). Springer Berlin Heidelberg. (Cited on pages 110, 119, 129, and 403.)

Securosis (2010). Monitoring up the Stack: Adding Value to SIEM. White paper, Securosis L.L.C., Phoenix, AZ. (Cited on page 121.)

Securosis (2011). Applied Network Security Analysis: Moving from Data to Information. White paper, Securosis L.L.C., Phoenix, AZ. (Cited on page 121.)

Seguran, M., Hebert, C., & Frankova, G. (2008). Secure workflow development from early requirements analysis. In *European Conference on Web Services (ECOWS 2008)*, (pp. 125–134). IEEE. (Cited on page 117.)

Serban, C. & McMillin, B. (1996). Run-time security evaluation (RTSE) for distributed applications. In *Symposion on Security and Privacy*, (pp. 222–232). IEEE. (Cited on page 118.)

Shameli-Sendi, A., Ezzati-Jivan, N., Jabbarifar, M., & Dagenais, M. (2012). Intrusion response systems: survey and taxonomy. *SIGMOD Rec*, *12*, 1–14. (Cited on pages 126 and 404.)

Sheyner, O., Haines, J. W., Jha, S., Lippmann, R., & Wing, J. M. (2002). Automated generation and analysis of attack graphs. In *2002 IEEE Symposium on Security and Privacy, May 12-15, 2002, Berkeley, California, USA*, (pp. 273–284). IEEE Comp. Soc. Press. (Cited on page 87.)

Shirey, R. (2007). Internet Security Glossary, Version 2. RFC 4949 (Informational). (Cited on page 3.)

Skopik, F., Ma, Z., Smith, P., & Bleier, T. (2012). Designing a cyber attack information system for national situational awareness. In N. Aschenbruck, P. Martini, M. Meier, & J. Tölle (Eds.), *Future Security*, volume 318 of *Communications in Computer and Information Science* (pp. 277–288). Springer. (Cited on pages 110, 126, and 404.)

Sobocinski, P. (2007). A well-behaved lts for the pi-calculus: (abstract). *Electr. Notes Theor. Comput. Sci.*, *192*(1), 5–11. (Cited on page 36.)

Software, L. (2010). Common event format configuration guide. (Cited on page 121.)

Spanoudakis, G., Kloukinas, C., & Androutsopoulos, K. (2007). Towards security monitoring patterns. In *Symposium on Applied computing (SAC 2007)*, (pp. 1518–1525). ACM. (Cited on page 118.)

Steele, Jr., G. L. (1990). *Common LISP: the language (2nd ed.)*. Newton, MA, USA: Digital Press. (Cited on page 109.)

Stirling, C. (1989). An introduction to modal and temporal logics for CCS. In Yonezawa, A. & Ito, T. (Eds.), *Concurrency: Theory, Language, and Architecture*, volume 391 of *Lecture Notes in Computer Science*. Springer Verlag. (Cited on page 58.)

Stroetmann, K. A. & Lilischkis, S. (2007). ehealth strategy and implementation activities in germany. (Cited on pages 7, 45, 52, and 133.)

Suzuki, I. (1988). Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, *28*(4), 213–214. (Cited on page 63.)

Swiler, L. P., Phillips, C., Ellis, D., & Chakerian, S. (2001). Computer-attack graph generation tool. In *DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume 2,June 12 - 14, 2001, Anaheim, California*, (pp. 1307–1321). IEEE Computer Society. (Cited on page 87.)

Tallon, P. (2008). Inside the adaptive enterprise: an information technology capabilities perspective on business process agility. *Information Technology and Management*, *9*(1), 21–36. (Cited on page 97.)

Talupur, M. (2006). *Abstraction Techniques for Parameterized Verification*. PhD thesis, Computer Science Department, Carnegie Mellon University. CMU-CS-06-169. (Cited on page 62.)

Tjoa, S., Jakoubi, S., Goluch, G., Kitzler, G., Goluch, S., & Quirchmayr, G. (2011). A formal approach enabling risk-aware business process modeling and simulation. *IEEE Transactions on Services Computing*, *4*(2), 153–166. (Cited on page 117.)

Toktar, E., Jamhour, E., & Maziero, C. (2004). RSVP Policy Control using XACML. In *Proc. of the IEEE 5th International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, (pp. 87ff)., Los Alamitos, CA, USA. IEEE Computer Society. (Cited on page 90.)

Tsigritis, T. & Spanoudakis, G. (2008). Diagnosing runtime violations of security & dependability properties. In *Software Engineering and*

*Knowledge Engineering (SEKE 2008)*, (pp. 661–666). KSI. (Cited on page 118.)

Turner, D. (2007). Symantec Internet Security Threat Report: Trends for January–June 07. Technical report, Symantec Corporation. (Cited on page 10.)

Uribe, T. E. (2000). Combinations of Model Checking and Theorem Proving. In *FroCoS '00: Proceedings of the Third International Workshop on Frontiers of Combining Systems*, (pp. 151–170)., London, UK. Springer. (Cited on page 62.)

van der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin: Springer. (Cited on pages 97 and 118.)

van der Aalst, W. M. P. (2013). Business process management: A comprehensive survey. *ISRN Software Engineering*, 37. (Cited on page 119.)

van der Aalst, W. M. P., van Dongen, B. F., Günther, C., Rozinat, A., Verbeek, H. M. W., & Weijters, A. J. M. M. (2009). Prom: The process mining toolkit. In *BPM 2009 Demonstration Track*, volume 489, (pp. 1–4). CEUR. (Cited on pages 109 and 118.)

van Lamsweerde, A. (2004). Elaborating security requirements by construction of intentional anti-models. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, (pp. 148–157)., Washington, DC, USA. IEEE Computer Society. (Cited on pages 8, 47, and 60.)

Verbeek, H., Buijs, J., Dongen, B., & Aalst, W. (2011). Xes, xesame, and prom 6. In P. Soffer & E. Proper (Eds.), *Information Systems Evolution*, volume 72 of *Lecture Notes in Business Information Processing* (pp. 60–75). Springer Berlin Heidelberg. (Cited on pages 109 and 118.)

Verissimo, P., Neves, N., Goller, A., Limancero, A. R., González, S., Torres, R., Romano, L., D'Antonio, S., Debar, H., Rieke, R., Stoynova, Z., Kotenko, I., Chechulin, A., Jimenez-Peris, R., Soriente, C., Kheir, N., & Viinikka, J. (2012). Massif architecture document. Technical report, FP7-257475 MASSIF European project. (Cited on page 109.)

Vianello, V., Gulisano, V., Jimenez-Peris, R., Patino-Martinez, M., Torres, R., Diaz, R., & Prieto, E. (2013). A scalable siem correlation engine and its application to the olympic games it infrastructure. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, (pp. 625–629). IEEE Computer Society. (Cited on pages 112, 128, and 444.)

Viega, J. & McGraw, G. (2002). *Building Secure Software.* Boston: Addison-Wesley Professional Computer Series. (Cited on pages 64 and 203.)

Viking project consortium (2012). Project VIKING. `http://www.vikingproject.eu/`. [Online; accessed 13-Oct-2013]. (Cited on pages 87 and 121.)

Wang, J., Whitley, J. N., Phan, R. C.-W., & Parish), D. J. (2011). Unified parametrizable attack tree. *International Journal for Information Security Research (IJISR)*, *1*, 20–26. (Cited on page 87.)

Wang, L., Jajodia, S., Singhal, A., & Noel, S. (2010). k-zero day safety: measuring the security risk of networks against unknown attacks. In *Proceedings of the 15th European conference on Research in computer security*, ESORICS'10, (pp. 573–587)., Berlin, Heidelberg. Springer-Verlag. (Cited on pages 21, 81, and 88.)

Wang, W., Hidvégi, Z., Bailey, Jr., A. D., & Whinston, A. B. (2000). E-process design and assurance using model checking. *IEEE Computer*, *33*(10), 48–53. (Cited on page 44.)

Weber, M. & Kindler, E. (2003). The petri net markup language. In *Petri Net Technology for Communication-Based Systems*, volume 2472 of *LNCS* (pp. 124–144). Springer. (Cited on page 109.)

Weinman, J. (2011). Axiomatic Cloud Theory. `http://www.joeweinman.com/Resources/Joe_Weinman_Axiomatic_Cloud_Theory.pdf`. (Cited on pages 8 and 52.)

Weldemariam, K. & Villafiorita, A. (2011). Procedural security analysis: A methodological approach. *Journal of Systems and Software*, *84*(7), 1114–1129. (Cited on page 117.)

Wolter, C., Menzel, M., Schaad, A., Miseldine, P., & Meinel, C. (2009). Model-driven business process security requirement specification. *Journal of Systems Architecture*, *55*(4), 211–223. (Cited on page 117.)

Xie, F. & Liu, H. (2007). Unified property specification for hardware/-software co-verification. In *COMPSAC (1)*, (pp. 483–490). IEEE Computer Society. (Cited on page 58.)

Zegzhda, P., Zegzhda, D., & Nikolskiy, A. (2012). Using graph theory for cloud system security modeling. In I. Kotenko & V. Skormin (Eds.), *Computer Network Security*, volume 7531 of *Lecture Notes in Computer Science* (pp. 309–318). Springer Berlin / Heidelberg. (Cited on pages 33 and 52.)

Zhu, B., Joseph, A., & Sastry, S. (2011). Taxonomy of Cyber Attacks on SCADA Systems. In *Proceedings of CPSCom 2011: The 4th IEEE International Conference on Cyber, Physical and Social Computing, Dalian, China*. (Cited on pages 61 and 121.)

Zonouz, S. A., Khurana, H., Sanders, W. H., & Yardley, T. M. (2009). Rre: A game-theoretic intrusion response and recovery engine. In *DSN*, (pp. 439–448). IEEE. (Cited on page 86.)

Zonouz, S. A., Sharma, A., Ramasamy, H. V., Kalbarczyk, Z. T., Pfitzmann, B., McAuliffe, K. P., Iyer, R. K., Sanders, W. H., & Cope, E. (2011). Managing business health in the presence of malicious attacks. In *DSN Workshops*, (pp. 9–14). IEEE. (Cited on page 86.)

# Part III

# PEER-REVIEWED PUBLICATIONS

*This calls to mind one of the most wonderful features of reasoning, and one of the most important philosophemes in the doctrine of science, of which, however, you will search in vain for any mention in any book I can think of; namely, that reasoning tends to correct itself, and the more so, the more wisely its plan is laid. Nay, it not only corrects its conclusions, it even corrects its premises.*

— Charles Sanders Peirce, *The First Rule of Logic (1898)*

# P1

# THE SH-VERIFICATION TOOL – ABSTRACTION-BASED VERIFICATION OF CO-OPERATING SYSTEMS

| | |
|---|---|
| **Title** | The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems |
| **Authors** | Peter Ochsenschläger, Jürgen Repp, Roland Rieke, and Ulrich Nitsche |
| **Publication** | *Formal Aspects of Computing, The International Journal of Formal Method*, 10:381–404, 1998. |
| **ISBN/ISSN** | ISSN 0934-5043 |
| **DOI** | `http://dx.doi.org/10.1007/s001650050023` |
| **Status** | Published |
| **Publisher** | Springer-Verlag London Limited |
| **Publication Type** | Journal |
| **Copyright** | 1998, BCS |
| **Contribution of Roland Rieke** | Co-Author with significant contribution.<br>Specific contributions are: (1) the design of the model checking algorithms for temporal logic properties; (2) implementation of these algorithms within the Simple Homomorphism Verification Tool (SHVT), e.g., construction of a Büchi-Automaton representing the property given by a Propositional Linear Temporal Logic (PLTL) formula, construction of the synchronous product of the automaton of property and system behaviour, construction of the complement automaton, construction of the intersection with the automaton representing the behaviour, and check whether the resulting automaton is empty. |

Table 6: Fact Sheet Publication *P1*

Publication P1 [Ochsenschläger, Repp, Rieke & Nitsche, 1998] addresses the following research question:

RQ1 *How can it be proven that components of cooperating systems securely work together?*

This paper gives an overview about the main functions of the SHVT. The aim of the SHVT is to support the verification of cooperating systems. Cooperating systems are specific distributed Systems of Systems (SoS) which are characterised by freedom of decision and loose coupling of their components. This causes a high degree of nondeterminism which has to be handled by the analysis methods. Typical examples of cooperating systems are telephone systems, communication protocols, smartcard systems, electronic money, and contract systems. In that context, verification is the proof that system components work together in a desired manner. At that, the main strength of the tool is the combination of an inherent fairness assumption in the satisfaction relation, an abstraction technique compatible with approximate satisfaction, and a suitable compositional and partial order method for the construction of only a partial state space.

Formal Aspects of Computing (1998) 3: 1–000
© 1998 BCS

# The SH-Verification Tool — Abstraction-Based Verification of Co-operating Systems

P. Ochsenschläger[a], J. Repp[a], R. Rieke[a] and U. Nitsche[b]

[a] GMD — German National Research Centre for Computer Science, Institute for Telecooperation Technology, Germany

[b] Department of Electronics and Computer Science, University of Southampton, Southampton,

**Keywords:** Simple language homomorphisms; Asynchronous product automata; Approximate satisfaction of safety and liveness properties; Model checking; Verification tools

**Abstract.** The sh-verification tool comprises computing abstractions of finite-state behaviour representations as well as automata and temporal logic based verification approaches. To be suitable for the verification of so called co-operating systems, a modified type of satisfaction relation (approximate satisfaction) is considered. Regarding abstraction, alphabetic language homomorphisms are used to compute abstract behaviours. To avoid loss of important information when moving to the abstract level, abstracting homomorphisms have to satisfy a certain property called simplicity on the concrete (i.e. not abstracted) behaviour. The well known state space explosion problem is tackled by a compositional method combined with a partial order method.

## 1. Introduction

The aim of the sh-verification tool (sh means simple homomorphisms, which will be explained below) is to support the verification of co-operating systems. By co-operating systems we mean distributed systems which are characterized by freedom of decision and loose coupling of their components. This causes a high

*Correspondence and offprint requests to*: Peter Ochsenschläger, GMD — German National Research Centre for Computer Science, Institute for Telecooperation Technology, Rheinstr. 75, D-64295 Darmstadt, Germany email: ochsenschlaeger@darmstadt.gmd.de degree of nondeterminism which is handled by our methods. Typical examples of co-operating systems are telephone systems, communication protocols, smart-card systems, electronic money, contract systems, etc.

In that context verification is the proof that system components work together in a desired manner. So the dynamic behaviour of the system has to be investigated. One usual approach is to start with a formal specification of the dynamic behaviour of the system which is represented by a *labelled transition system* (LTS), and then to prove properties of such an LTS. But for real life applications the corresponding LTS are often too complex to apply this naive approach.

In contrast to the immense number of transitions of such an LTS usually only a few characteristic actions of the system are of interest with respect to verification. So it is evident to define abstractions with respect to the actions of interest and to compute a representation of such an abstract behaviour, which usually is much smaller than the LTS of the specification. For such a small representation dynamic properties can be proven more efficiently. Now, under certain conditions, properties of the system specification can be deduced from properties of the abstract behaviour.

For such an approach the following questions have to be answered:

**Question 1:** What does it formally mean, that a system satisfies a property (especially in the context of co-operating systems)?

**Question 2:** How can we formally define abstractions?

**Question 3:** For what kind of abstractions is there a sufficiently strong relation between system properties and properties of the abstract behaviour?

**Question 4:** How can we compute a representation of the abstract behaviour efficiently?

The present article is an extended and completed version of [ORRN97].

## 2. Approximately Satisfied Properties

As a small but typical example to illustrate our answers to these questions, we consider a system that consists of a client and a server as its main components. The client sends requests to the server, expecting the server to produce particular results. Nevertheless, for some reasons, the server may not always respond a request by sending a result, but may, as well, reject a request. The main actions that are important with respect to the client's behaviour, are sending a request and receiving a result or rejection. These actions are depicted as $REQ$, $RES$, and $REJ$ in Figure 1. We will regard the whole system running properly, if the client, at no time, is prohibited completely from receiving a result after having sent a request.
For the moment, we regard the server as a black box; i.e. we neither consider its internal structure nor look at its internal actions. Not caring about particular actions of a specification when regarding the specification's behaviour is *behaviour abstraction*. If we define a suitable abstraction for the client/server system with respect to our correctness criterion, we only keep actions $REQ$, $RES$, and $REJ$

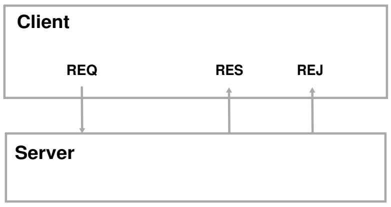


**Fig. 1.** Client Server Example

visible, hiding all other actions.

To formalise behaviour abstraction we use terms of *formal language theory*. An LTS is completely determined by the set of its paths starting at the initial state. This set is a formal language, called the *local language* of the LTS [Eil74]. Its letters are the transitions (state, transition label, successor-state) of the LTS. $\Sigma$ denotes the set of all transitions of the LTS. Consequently, there is a one-to-one correspondence between the LTS and its local language $L \subset \Sigma^*$ , where $\Sigma^*$ is the set of all sequences of elements of $\Sigma$ including the empty sequence $\epsilon$. Now behaviour abstraction can be formalized by *language homomorphisms*, more precisely by alphabetic language homomorphisms $h : \Sigma^* \to \Sigma'^*$ (**answer to question 2**). By these homomorphisms certain transitions are ignored and others are renamed, which may have the effect, that different transitions are identified with one another. A mapping $h : \Sigma^* \to \Sigma'^*$ is called a language homomorphism if $h(\epsilon) = \epsilon$ and $h(yz) = h(y)h(z)$ for each $y, z \in \Sigma^*$. It is called alphabetic, if $h(\Sigma) \subset \Sigma' \cup \{\epsilon\}$.

An automaton representation (*minimal automaton* [Eil74]) for the abstract behaviour of a specification (homomorphic image of the LTS's local language) can be computed by the sh-verification tool. Applying the abstraction described above to the concrete (i.e. not abstracted) behaviour of a specification of the client/server system leads to an automaton representation of abstract behaviour as presented in Figure 2. For this example $\Sigma' = \{REQ, RES, REJ\}$ .

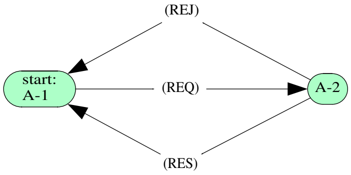


**Fig. 2.** Minimal Automaton.

The abstract behaviour obviously satisfies the correctness requirement mentioned above that, at no time, the client can be prohibited from receiving a result to a request. The usual concept of *linear satisfaction of properties* [AS85] (each infinite run of the system satisfies the property) is not suitable in this context since it considers also the extreme executions like “a request is always rejected”. Obviously, the problem occurs because no fairness constraints are considered. We put a very abstract notion of fairness into the satisfaction relation for properties, which considers that “independent of a finitely long computation of the system, it is always possible that a request is responded by result”. To formalise such “possibility properties”, which are of interest when considering what we call co-operating systems, the notion of *approximate satisfaction* of properties is defined in [NO96] (**answer to question 1**):

**Definition 2.1.** *An automaton approximately satisfies a property if and only if each finite path of transitions of the automaton can be continued to an infinite path, which satisfies the property.*

As it is well known [AS85], system properties are divided into two types: *safety* (what happens is not wrong) and *liveness* properties (eventually something desired happens). For safety properties linear satisfaction and approximate satisfaction are equivalent [NO96]. Approximately satisfied liveness properties are liveness properties with respect to the universe of a system's behaviour. They are related to linear satisfaction of properties under strong fairness constraints for the sake of adding behaviour invariant state information [NW97].

## 3. Simple Homomorphisms as an Abstraction Concept

It is now the question of main interest, whether, by investigating an abstract behaviour, we may verify the correctness of the underlying concrete behaviour. We will answer this question positively, requiring a restriction of the permitted abstraction techniques. To deduce approximately satisfied properties of a specification from properties of its abstract behaviour an additional property of abstractions is required: called *simplicity of homomorphisms* on a specification [Och92, Och94b]. Simplicity of homomorphisms on specifications is a very technical condition concerning the possible continuations of finite behaviours.

Concerning abstractions $h : \Sigma^* \to \Sigma'^*$ the crucial point are the liveness properties of a Language $L \subset \Sigma^*$ . To define simplicity formally we need $w^{-1}(L) = \{y \in \Sigma^* | wy \in L\}$ , the *set of continuations* of a word $w$ in a language $L$ [Eil74]. These continuations in some sense “represent” the liveness properties of $L$. Generally $h(x^{-1}(L))$ is a (proper) subset of $h(x)^{-1}(h(L))$ , but we want to have that $h(x^{-1}(L))$ “eventually” equals $h(x)^{-1}(h(L))$ .

**Definition 3.1.** *A homomorphism $h$ is called simple on $L$, if for each $x \in L$ there exists $w \in h(x)^{-1}(h(L))$ such that $w^{-1}(h(x^{-1}(L))) = (h(x)w)^{-1}(h(L))$ .*

For regular languages simplicity of a homomorphism is a decidable property. Necessary and sufficient conditions for a homomorphism to be simple exist on the state graph level which are practically motivated and can be checked very efficiently. We will discuss this in more detail subsequently. The following theorem [NO96] shows that approximate satisfaction of properties and simplicity of homomorphisms exactly fit together for verifying co-operating systems (**answer to question 3**):

**Theorem 3.2.** *Simple homomorphisms define exactly the class of such Abstractions, for which holds that each property is approximately satisfied by the abstract behaviour if and only if the “corresponding“ property is approximately satisfied by the concrete behaviour of the system.*

Formally, the “corresponding“ property is expressed by the inverse image of the abstract property with respect to the homomorphism.

Our verification method, which is based on the very general notions of approximate satisfaction of properties and simple language homomorphisms, does not depend on a specific formal specification method. It can be applied to all specification techniques with an LTS-semantics.

To point out and motivate in more detail the necessity of considering approximate satisfaction of properties and of restricting suitable abstraction techniques to simple homomorphisms, we have to look more closely at the structure the server may have.

The server's answer (result or rejection) to a client's request may depend on whether a resource is available. If the resource is free, the server will respond a request by sending a result, if the resource is locked when the server is requested, the server will reject the request. Assuming that the server cannot control the resource, there is no guarantee at all that the resource is not locked all the time the client sends a request. Therefore, for some quite extreme computation scenarios, a request may always be rejected. So the best we can expect is that, in principle, there is always the possibility that a request is answered by eventually producing a result. This type of requirements is exactly captured by the definition of approximate satisfaction of properties. We revisit the correctness criterion for the client/server specification: the client is never prohibited completely from receiving a result after having sent a request.

If the resource behaves properly, i.e. it would change infinitely often from state locked to state free and vice versa in an infinite amount of time, the client/server specification will be correct with respect to the above requirement. Hence, since Figure 2 represents the abstract behaviour of the specification, the abstract as well as the concrete behaviour meet the correctness requirement. One may conjecture that the satisfaction of correctness criteria is preserved when moving from the abstract to the concrete behaviour.

Let us now consider a resource not showing a proper behaviour. A formal specification in terms of Petri nets is given in Section 7. For some reason, the resource may eventually be locked forever. Indeed, we consider now a resource that contains an error. If the resource vanishes forever, the client will never receive a result again. Thus this modified client/server specification does not meet the correctness requirement anymore. Regarding that the modified system may behave correct as well as after some time may behave incorrect, when coming to abstraction, the correct behaviour hides the incorrect one. This is, because the incorrect behaviour is a subset of the correct one, and therefore, when brought together by looking only at actions $REQ$, $RES$, and $REJ$, the abstract behaviour of the system is still represented in Figure 2. Here, the considered correctness requirement is not preserved when changing the viewpoint from the abstract to the concrete behaviour.

The problem of a correct subbehaviour hiding an incorrect subbehaviour under abstraction can be most easily explained when looking at the strongly connected components of the LTS that represents the concrete behaviour. For the incorrect client/server specification, Figure 3 shows the LTS representing the behaviour of the client/server specification such that the strongly connected components of the LTS are differently marked. When drawing our attention to abstraction, the strongly connected component that contains the initial state corresponds to a correct abstract subbehaviour of the specification. The second strongly connected component, which is a bottom component (no outgoing transitions from this component), corresponds to an incorrect abstract subbehaviour. When computing the minimal representation for the abstract behaviour (it is naturally the *minimal* representation of the abstract behaviour that we are interested in the abstraction framework), the two strongly connected components are "shuffled" in such a way that the resulting LTS shows the maximal possible behaviour with respect to the shuffling. Hence the first component covers the second one and the incorrect behaviour is hidden.

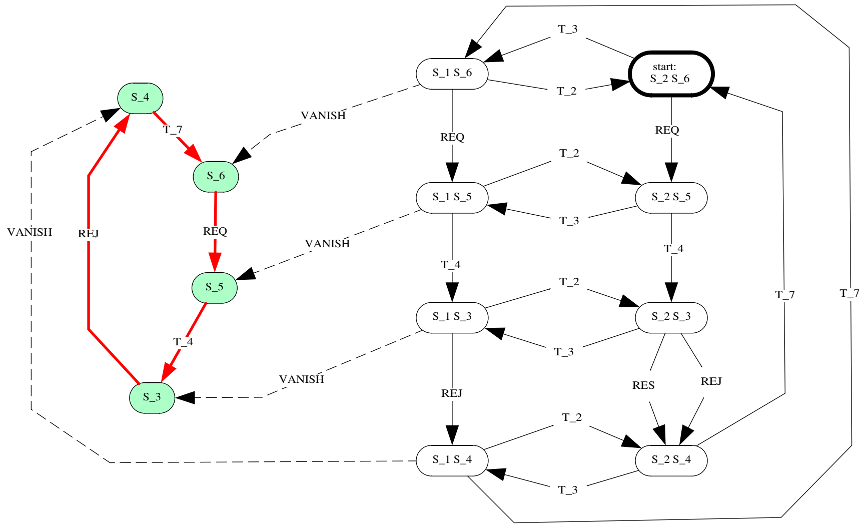


**Fig. 3.** LTS

In the considered example, the so far discussed problem can be detected easily when computing a graph representation of the strongly connected components of the concrete LTS, called the *component graph*. Each node in this graph representation is a strongly connected component and we have an arc from one node to another, if there exists a transition to move from one strongly connected component to the other. We label these nodes with abstract actions that can occur in the corresponding strongly connected components with respect to the defined abstraction.

The component graph of an LTS can be computed by the sh-verification tool and Figure 4 shows this graph representation for our example. Realizing that in the second node, which is a leaf because it represents a strongly connected bottom component, the action *RES* is missing compared to the first node. We obtain that in the strongly connected bottom component a request cannot be answered with a result anymore, which reveals exactly the violation of the requirement that we considered. There are reachable states wherefrom *REQ* can never be responded with *RES*.

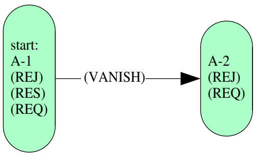


**Fig. 4.** Connected Components

It is rather obvious that we can only be interested in abstractions where hiding of an incorrect subbehaviour by a correct one cannot occur. For this purpose, simplicity of homomorphisms on behaviours has been defined. And, indeed, simplicity of homomorphisms is a necessary and sufficient condition for the preservation of approximately satisfied properties when changing the point of view from the abstract to the concrete behaviour.

Inspecting the strongly connected components of an LTS simplicity of an abstraction can be investigated. In [Och92, Och94b] the following sufficient condition for simplicity has been proven:

**Theorem 3.3.** *Let $L$ be a Language recognized by a finite automaton $\mathcal{A}$ and let $h$ be a homomorphism on $L$. If for each $x \in L$ there exists $y \in x^{-1}(L)$ leading to a dead component of $\mathcal{A}$ , such that each $z \in L$ with $h(z) = h(xy)$ leads to the same dead component, then $h$ is simple on $L$. This condition is satisfied for example, if each dead component contains a label $a$ of an edge with $h(a) \neq \epsilon$, such that no edge exists outside of this component, whose label has the same image $h(a)$. If $\mathcal{A}$ is strongly connected, then each homomorphism is simple on $L$.*

To prove non-simplicity a necessary condition for simplicity is needed.
If $h(x^{-1}(L)) = \{\epsilon\}$ for a homomorphism $h : \Sigma^* \to \Sigma'^*, L \subset \Sigma^*$ and $x \in L$ then $h$ is simple on $L$ in $x$ only if $h(x)^{-1}(h(L)) = \{\epsilon\}$. In earlier papers [Och88, Och90, Och91b] this situation has been formalized by so called deadlock languages. They consider abstract behaviours leading to states where no visible (under the abstraction) continuations exist.

**Definition 3.4.** *The deadlock language $DL$ of a language $L$ with respect to a homomorphism $h$ is defined by $DL = \{u \in h(L) \mid$ there exist $x \in L$ with $u = h(x)$ and $h(x^{-1}(L)) = \{\epsilon\}\}$ .*

The minimal automaton of the deadlock language is called *deadlock automaton.* It can be computed by the sh-verification tool. If in our erroneous example all but action *RES* are hidden by a homomorphism $t : \Sigma^* \to \Sigma''^*$ , with $\Sigma'' = \{RES\}$, the corresponding deadlock automaton, as well as the minimal automaton of $t(L)$, is shown in Figure 5. The deadlock automaton of the correct example is empty.

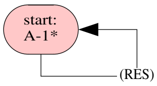


**Fig. 5.** Deadlock Automaton

To formulate a necessary condition for simplicity using deadlock languages the notion *termination language* is needed:

**Definition 3.5.** *The termination language $TL$ of a language $L$ with respect to a homomorphism $h$ is defined by $TL = \{u \in h(L) | u^{-1}(h(L)) = \{\epsilon\}\}$.*

It is easy to see that generally $TL \subset DL$ and that $TL = DL$ if the homomorphism $h$ is simple on $L$ [Och92, Och94b].

Concerning the homomorphism $t$ Figure 5 shows that $TL = \emptyset \neq DL$. So $t$ is not simple on $L$. To apply the necessary condition for simplicity to our homomorphism $h : \Sigma^* \to \Sigma'^*$ with $\Sigma' = \{REQ, RES, REJ\}$ we have to consider compositions of homomorphisms. Let $f : \Sigma_1^* \to \Sigma_2^*$ and $g : \Sigma_2^* \to \Sigma_3^*$ be mappings. The *composition* $g \circ f : \Sigma_1^* \to \Sigma_3^*$ is defined by $(g \circ f)(x) = g(f(x))$ for each $x \in \Sigma_1^*$. If $g$ and $f$ are homomorphisms then $g \circ f$ is a homomorphism too. The following theorems express the compatibility of simplicity with composition of homomorphisms [Och92, Och94b].

**Theorem 3.6.** *If $f$ is simple on $L \subset \Sigma_1^*$ and $g$ is simple on $f(L) \subset \Sigma_2^*$ then $g \circ f$ is simple on $L$.*

**Theorem 3.7.** *If $g \circ f$ is simple on $L \subset \Sigma_1^*$ then $g$ is simple on $f(L)$.*

Considering the homomorphism $t' : \Sigma'^* \to \Sigma''^*$ , defined by $t'(RES) = RES$ and $t'(X) = \epsilon$ for $X \neq RES$ , we have $t = t' \circ h$ . Now $t'$ is simple on $h(L)$ because the automaton in Figure 2 is strongly connected. By the above theorem simplicity of $h$ on $L$ would imply simplicity of $t$ on $L$, which is not true. So $h$ is not simple on $L$, and the defect of our erroneous specification can be detected by simplicity investigations of appropriate homomorphisms without using the complex decision algorithm for simplicity.

## 4. A Compositional Approach to Avoid State Space Explosion

Simple homomorphisms establish the coarsest, i.e. most abstract notion of system equivalence with respect to a given (abstract) requirement specification [NO96]. What still remains open is the question of how to construct an abstract behaviour to a given specification without an exhaustive construction of its state space.

To handle the well known state space explosion problem, *a compositional method* has been developed [Och94c, Och95, Och96] and implemented in the sh-verification tool. In case of well structured specifications, by applying a divide and conquer strategy this method allows to compute a representation of the abstract behaviour and to check simplicity of homomorphisms efficiently without having to compute the complex LTS of the complete specification (**answer to question 4**). This compositional method is combined with a *partial order method* based on *partially commutative languages* [Och97]. The main goal of our compositional method is to compute minimal automata of homomorphic images and to check simplicity of homomorphisms efficiently even in case of complex specifications. The fundamental idea is to embed each component of a structured system (X and Y in Figure 9) in a "simplified environment" (Y' and X' in Figure 6), which shows at the interface an "equivalent behaviour" compared to the rest of the system (shaded areas in Figure 6). This can be checked using special homomorphisms, called *boundary homomorphisms*. The complexity of this check is reduced by our partial order method.

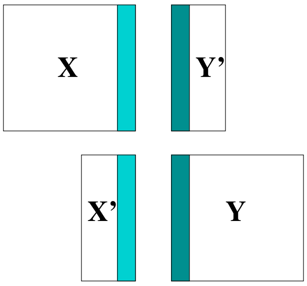


**Fig. 6.**

For each of these smaller systems minimal automata related to corresponding homomorphisms (which have to be finer than the boundary homomorphisms) are computed (Figure 7) and are composed (Figure 8) to obtain the desired automaton (Figure 9). This kind of composition is defined by the notion of *asynchronous product automata* and *co-operation products of formal languages*, a restricted kind of shuffle product. Simplicity of homomorphisms on co-operation products is guaranteed by a particular property of the boundary homomorphisms, which is called *co-operativity* For more details we refer to the next chapter and to [Och94c, Och95, Och96].

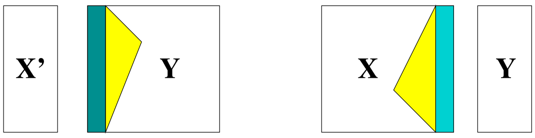


**Fig. 7.**

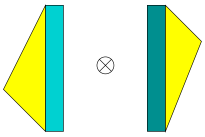


**Fig. 8.**

By that, compact representations of abstractions of system behaviour can be computed and simplicity of abstractions can be checked without investigating the complete behaviour of a complex system. In case of “well structured“ specifications this method causes considerable reductions of the state spaces. The smaller systems with “simplified environments” avoid a lot of interleavings of actions (“state space explosion”), which are not relevant with respect to the considered abstraction but which are instrumental in the complex dynamics of the system.

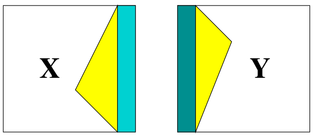


**Fig. 9.**

This approach can also be used iteratively and allows induction proofs for systems with several identical components [Och96]. Using our compositional method a connection establishment and release protocol has been verified by investigating automata with about 100 states instead of 100000 states.

## 5. Asynchronous Product Automata

As a formal basis for our compositional approach as well as a formal specification language we now define the notion of asynchronous product automata (APA), a very general class of communicating automata. APA can be regarded as families of automata (*elementary automata*), whose sets of states are cartesian products and whose elementary automata are "glued together" by common components of these products.

**Definition 5.1.** An *asynchronous product automaton (APA)* consists of a family of *sets of state components* $(Z_s)_{s\in\mathcal{S}}$, a family of *elementary automata* $(\Phi_t, \Delta_t)_{t\in\mathcal{T}}$ and a *neighbourhood relation* $N : \mathcal{T} \to \wp(\mathcal{S})$ ($\wp(X)$ denotes the set of all subsets of $X$). For each elementary automaton $(\Phi_t, \Delta_t)$

- $\Phi_t$ is its *alphabet* and
- $\Delta_t \subset \bigtimes_{s\in N(t)}(Z_s) \times \Phi_t \times \bigtimes_{s\in N(t)}(Z_s)$ is its set of *state transition relation.*

To avoid pathological cases we assume $\mathcal{S} = \bigcup_{t\in\mathcal{T}}(N(t))$ and $N(t) \neq \emptyset$ for all $t \in \mathcal{T}$. The *states* of an APA are elements of $\bigtimes_{s\in\mathcal{S}}(Z_s)$ with the *initial state* $q_0 = (q_{0s})_{s\in\mathcal{S}} \in \bigtimes_{s\in\mathcal{S}}(Z_s)$. Formally an APA $\mathcal{A}$ is defined by a quadruple $\mathcal{A} = ((Z_s)_{s\in\mathcal{S}}, (\Phi_t, \Delta_t)_{t\in\mathcal{T}}, N, q_o)$.

"Dynamics" of APA are defined by "occurrences" of elementary automata. An elementary automaton $(\Phi_t, \Delta_t)$ is *activated* in a state $p = (p_s)_{s\in\mathcal{S}} \in \bigtimes_{s\in\mathcal{S}}(Z_s)$ *with respect to an interpretation* $i \in \Phi_t$, if there exists $(q_s)_{s\in N(t)} \in \bigtimes_{s\in N(t)}(Z_s)$ with $((p_s)_{s\in N(t)}, i, (q_s)_{s\in N(t)}) \in \Delta_t$. An activated elementary automaton $(\Phi_t, \Delta_t)$ may *occur* and generates a *successor state* $q = (q_r)_{r\in\mathcal{S}} \in \bigtimes_{s\in\mathcal{S}}$ with $q_r = p_r$ for $r \in \mathcal{S} \setminus N(t)$ and $((p_s)_{s\in N(t)}, i, (q_s)_{s\in N(t)}) \in \Delta_t$.

In this case $(p, (t, i), q)$ denotes the corresponding *occurrence step.* The occurrence of an elementary automaton changes the state components of its neighbourhood.

A sequence of the form $w = (q_1, (t_1, i_1), q_2)(q_2, (t_2, i_2), q_3)...(q_n, (t_n, i_n), q_{n+1})$ with $n \geq 1$ is called an *occurrence sequence.* If such an occurrence sequence exist, then we say that $q_{n+1}$ is *reachable* from $q_1$. Additionally by definition each state is reachable from itself. $\mathcal{Q}$ (the *state space*) denotes the set of all states

$q \in \bigtimes_{s \in \mathcal{S}}(Z_s)$ reachable from the initial state $q_0$ and $\Sigma$ denotes the set of all occurrence steps, whose first component is an element of $\mathcal{Q}$. The set $L \subset \Sigma^*$ of all occurrence sequences starting with the initial state $q_0$ and containing the empty sequence $\varepsilon$ is called the *occurrence language* of the corresponding APA. $\Sigma$ can also be interpreted as the set of arcs of a directed graph, whose set of nodes is $\mathcal{Q}$ and whose arcs are labeled by pairs $(t, i)$ with $t \in \mathcal{T}$ and $i \in \Phi_t$. This graph is called the *reachability graph* of the corresponding APA. By that occurrence sequences are paths in the reachability graph and the occurrence language is a regular language (local language), if the reachability graph is finite. The occurrence language as well as the reachability graph is a complete description of the dynamic behaviour of an APA.

As an example we give an APA specification of our incorrect client/server example. It is an APA representation of the Petri net in Figure 12, consisting of three elementary automata, $\mathcal{T} = \{C, S, R\}$ , and four state components, $\mathcal{S} = \{CS, IS, SS, RS\}$ . Figure 10 shows the neighbourhood relation N.

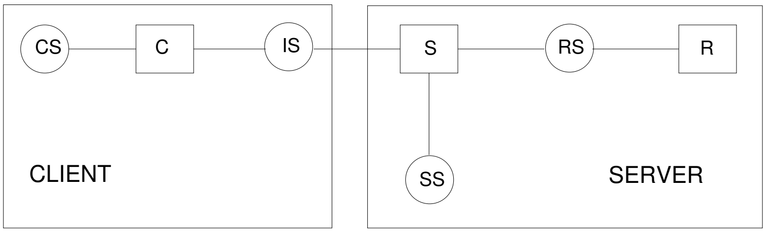


**Fig. 10.** Client / Server APA

State transitions of the elementary automaton C represent actions of the client. Correspondingly actions of the server and the resource manager are represented by state transitions of S and R respectively. CS and SS represent "internal" states of the client and the server. IS describes the the states of the client and server's interface. RS represents both, internal and interface states related to the resource manager. Formally the APA is defined as follows:

**state components:**
$Z_{CS} = Z_{SS} = \{idle, active\}, Z_{IS} = \{emp, req, res - rej\}$ ,
$Z_{RS} = \{avail, navail, vanished\}$

**imitial states:**
$q_{0CS} = q_{0SS} = idle, q_{0IS} = emp, q_{0RS} = avail$ .

**alphabets:**
$\Phi_C = \{REQ, T7\}$ , $\Phi_S = \{RES, REJ, T4\}$ , $\Phi_R = \{VANISH, T2, T3\}$ .

**state transition relations:**
$\Delta_C = \left\{ \begin{array}{l} ((idle, emp), REQ, (active, req)), \\ ((active, res - rej), T7, (idle, emp)) \end{array} \right\} \subset (Z_{CS} \times Z_{IS}) \times \Phi_C \times (Z_{CS} \times Z_{IS})$,

$$\Delta_S = \left\{ \begin{array}{l} ((idle, req, avail), T4, (active, emp, avail)), \\ ((idle, req, navail), T4, (active, emp, navail)), \\ ((idle, req, vanished), T4, (active, emp, vanished)), \\ ((active, emp, avail), RES, (idle, res-rej, avail)), \\ ((active, emp, avail), REJ, (idle, res-rej, avail)), \\ ((active, emp, navail), REJ, (idle, res-rej, navail)), \\ ((active, emp, vanished), REJ, (idle, res-rej, vanished)) \end{array} \right\} \subset (Z_{SS} \times$$

$$Z_{IS} \times Z_{RS}) \times \Phi_S \times (Z_{SS} \times Z_{IS} \times Z_{RS}),$$

$$\Delta_R = \{ \begin{array}{l} (avail, T3, navail), \\ (navail, T2, avail), \\ (navail, VANISH, vanished) \end{array} \} \subset Z_{RS} \times \Phi_R \times Z_{RS}.$$

State components correspond to markings of particular places of the Petri net, as for example $emp \in Z_{IS}$ denotes the empty marking of places S-4 and S-5 in Figure 12. The alphabets' elements correspond to the transitions of the Petri net. As the system is structured into three components given by the three elementary automata each alphabet represents the set of "local" actions of the corresponding component. Note that APA offer a very flexible concept for structuring specifications: decreasing the number of elementary automata increases the cardinality of the alphabets.

The reachability graph of our APA example is isomorphic to the LTS in Figure 3.

APA form a very general class of communicating automata. They are similar to asynchronous cellular automata introduced in [Zie89] and include different kinds of "simple" and "higher order" Petri nets as well as communicating automata as for example SDL [SSR89] or Estelle [BD87]. The above terminology is based on Petri nets: Elementary automata of APA correspond to transitions, state components correspond to places and states correspond to markings of places. By that the state transition relation is realized by the so called occurrence rule of a Petri net.

In terms of APA we now formulate our compositional method: A distributed system is an APA and the dynamical behaviour of the system is described by the occurrence language of that APA. A component (subsystem, module) of a system is defined by a subset $A \subset \mathcal{T}$. As we often consider the complement of $A$ ("rest of the system" with respect to $A$) we use the abbreviation $\overline{A}$ for $\mathcal{T} \setminus A$. To consider states of an APA restricted to a subset $Y \subset \mathcal{S}$ we define $q|Y = (q_s)_{s \in Y} \in \mathsf{X}_{s \in Y}(Z_s)$ for a state $q = (q_s)_{s \in \mathcal{S}} \in \mathsf{X}_{s \in \mathcal{S}}(Z_s)$. Two special homomorphisms $M_A$ (*module homomorphism*) and $R_A$ (*boundary homomorphism*) on the occurrence language $L \subset \Sigma^*$ of an APA are used to express the behaviour of a component $A$ of an APA and its behaviour at the interface to $\overline{A}$ respectively.

**Notation.** Let $RDA = N(A) \cap N(A')$. A homomorphism $M_A : \Sigma^* \to \Sigma^*_{MA}$ with $\Sigma_{MA} = M_A(\Sigma)$ is defined by

$$M_A((p,(t,i),q)) = \begin{cases} (p|N(A),(t,i),q|N(A)), & \text{if } t \in A \text{ and} \\ & N(t) \cap RDA \neq \emptyset, \\ (p|N(A) \setminus RDA,(t,i),q|N(A) \setminus RDA), & \text{if } t \in A \text{ and} \\ & N(t) \cap RDA = \emptyset, \\ \epsilon, & \text{if } t \in \overline{A}. \end{cases}$$

A homomorphism $R_A : \Sigma^* \to \Sigma^*_{RA}$ with $\Sigma_{RA} = R_A(\Sigma)$ is defined by

$$R_A((p,(t,i),q)) = \begin{cases} (p|RDA,q|RDA), & \text{if } t \in A \text{ and } N(t) \cap RDA \neq \emptyset, \\ \epsilon, & \text{if } t \in \overline{A} \text{ or } N(t) \cap RDA = \emptyset. \end{cases}$$

To compare homomorphisms with respect to their "degree of abstraction" we call a homomorphism $\phi : \Sigma^* \to \Delta^*$ *finer* than a homomorphism $\psi : \Sigma^* \to \Gamma^*$, if there exists a homomorphism $\nu : \Delta^* \to \Gamma^*$ with $\psi = \nu \circ \phi$. For this we use the notation $\phi\langle\psi$. In that case the homomorphic image $\phi(L)$ contains enough "information" to determine $\psi(L)$. As homomorphisms are used to describe abstractions we assume that they are alphabetic, i.e. $\phi(\Sigma) \subset \Delta \cup \{\epsilon\}$ for each homomorphism $\phi$.

**Notation.** Two homomorphisms "acting" on disjoint components of an APA can be "combined" obtaining a new homomorphism: Let $A \subset \mathcal{T}$ and let $f : \Sigma^* \to \Phi^*$ as well as $g : \Sigma^* \to \Gamma^*$ be homomorphisms with $M_A\langle f$ as well as $M_{\overline{A}}\langle g$, then the homomorphism $f \oplus g : \Sigma^* \to (\Phi \cup \Gamma)^*$ is defined by $(f \oplus g)((p,(t,i),q)) = f((p,(t,i),q))$ if $t \in A$ and $(f \oplus g)((p,(t,i),q)) = g((p,(t,i),q))$ if $t \in \overline{A}$. $f \oplus g$ is called the *direct sum* of $f$ and $g$. By the additional assumption $\Phi \cap \Gamma = \emptyset$ the direct sum of two homomorphisms is finer than both homomorphisms: There exist homomorphisms (projections) $\phi : (\Phi \cup \Gamma)^* \to \Phi$ and $\gamma : (\Phi \cup \Gamma)^* \to \Gamma^*$ with $f = \phi \circ (f \oplus g)$ and $g = \gamma \circ (f \oplus g)$.

The homomorphic image $(f \oplus g)(L)$ can be "constructed" using $f(L)$ and $g(L)$ if these two images contain enough information about the boundary behaviour of $A$ and $\overline{A}$ respectively, i.e. that $f\langle R_A$ and $g\langle R_A$. To formulate a corresponding theorem we need some further technical notions:

**Notation.** If $f\langle R_A$ and $g\langle R_A$, then there exist homomorphisms $\rho_A : \Phi^\star \to \Sigma^\star_{RA}$ and $\rho_{\overline{A}} : \Gamma^\star \to \Sigma^\star_{R\overline{A}}$ with $R_A = \rho_A \circ f$ and $R_{\overline{A}} = \rho_{\overline{A}} \circ g$.
Let $\Sigma_R = \Sigma_{RA} \cup \Sigma_{R\overline{A}}$ and let $\rho : (\Phi \cup \Gamma)^* \to \Sigma^*_R$ be the homomorphism defined by: $\rho(x) = \rho_A(x)$ if $x \in \Phi$ and $\rho(x) = \rho_{\overline{A}}(x)$ if $x \in \Gamma$.
Let $RC = \{z \in \Sigma^*_R|$ if $z = (p,q)y$ with $(p,q) \in \Sigma_R$ and $y \in \Sigma^*_R$, then $p = q_0|RDA$, and if $z = x(p,q)(p',q')y$ with $(p,q),(p',q') \in \Sigma_R$ and $x, y \in \Sigma^*_R$, then $p' = q\}$.
If $\Sigma_R$ is finite, then $RC$ is a regular language (local language). The definition of $RC$ depends on $\Sigma_R$. On account of $f\langle R_A$ and $g\langle R_{\overline{A}}$ this set can be determined using $f(L)$ and $g(L)$.

Under these assumptions the following holds:

**Theorem 5.2.** *Let $L \subset \Sigma^*$ be the occurrence language of an APA and $A \subset \mathcal{T}$. If $f : \Sigma^* \to \Phi^*$ and $g : \Sigma^* \to \Gamma^*$ are homomorphisms with $\Phi \cap \Gamma = \emptyset$ and $M_A\langle f\langle R_A$ as well as $M_{\overline{A}}\langle g\langle R_{\overline{A}}$, then $(f \oplus g)(L) = \phi^{-1}(f(L)) \cap \gamma^1(g(L)) \cap \rho^{-1}(RC)$.*

By this representation $(f \oplus g)(L)$ is a regular set if $L$ is regular. In [Och96]

$\phi^{-1}(f(L)) \cap \gamma^{1}(g(L)) \cap \rho^{-1}(RC)$ is called the *cooperation product* of $f(L)$ and $g(L)$. It is easy to construct a finite automaton recognizing $(f \oplus g)(L)$ using corresponding automata for $f(L)$ and $g(L)$. Concerning simplicity of $f \oplus g$ we have

**Theorem 5.3.** *If $f$ and $g$ are simple on $L$ by the same assumptions as in the above theorem, then $f \oplus g$ is simple on $L$ too.*

The proofs of these two theorems as well as the proofs of the other theorems of this chapter can be found in [Och94c, Och96]. Essential to the statements of this chapter is “locality” of occurrence steps, i.e. that state changes only occur in the neighbourhood of the corresponding elementary automata. Therefore occurrence sequences may be “rearranged” without changing certain homomorphic images. Such “rearrangements” are the main proof techniques for these theorems.

The above theorems form one half of our compositional method. They show how abstractions of the behaviour of components of an APA can be “composed”. But so far the representation of $(f \oplus g)(L)$ depends on $f(L)$ and $g(L)$. How can $f(L)$ and $g(L)$ be determined without using the (complex) occurrence language $L$ of the complete system ? To achieve this we “embed” the components $A$ and $\overline{A}$ in “simplified environments”. Since a component of an APA can be viewed as an APA too we now have to define how two APA can be “composed”.

The “gluing together” of elementary automata mentioned in the definition of APA can also be applied to arbitrary APA.

**Definition 5.4.** Let therefore $\mathcal{A}k = ((Zk_s)_{s \in \mathcal{S}k}, (\Phi k_t, \Delta k_t)_{t \in \mathcal{T}k}, Nk, qk_0)$ with $k \in \{1, 2\}$ be two APA with $\mathcal{T}1 \cap \mathcal{T}2 = \emptyset$ and $Z1_s = Z2_s$ as well as $q1_{0s} = q2_{0s}$ for all $s \in \mathcal{S}1 \cap \mathcal{S}2$. Now the *asynchronous product* $\mathcal{A}1 \otimes \mathcal{A}2$ is defined by $\mathcal{A}1 \otimes \mathcal{A}2 = ((Z_s)_{s \in \mathcal{S}}, (\Phi k_t, \Delta k_t)_{t \in \mathcal{T}}, N, q_0)$ with $\mathcal{S} = \mathcal{S}1 \cup \mathcal{S}2, \mathcal{T} = \mathcal{T}1 \cup \mathcal{T}2$, $Z_s = Zk_s$ and $q_{0s} = qk_{0s}$ for all $s \in \mathcal{S}k$ and $(\Phi_t, \Delta_t) = (\Phi k_t, \Delta k_t)$ for all $t \in \mathcal{T}k$ and $N(t) = Nk(t)$, where $k \in \{1, 2\}$. We also say that $\mathcal{A}1 \otimes \mathcal{A}2$ is constructed from $\mathcal{A}1$ and $\mathcal{A}2$ by *gluing together at the common state components* $\mathcal{S}1 \cap \mathcal{S}2$.

If $A$ and $\overline{A}$ are complementary components of an APA, then this APA is the asynchronous product of $A$ and $\overline{A}$. In terms of boundary behaviour the following theorem gives a sufficient condition to “embed” a component of an APA in different “environments” without changing its behaviour.

Let $\mathcal{X}$, $\mathcal{Y}'$, $\mathcal{X}'$ and $\mathcal{Y}$ be four APA, for which the asynchronous products $\mathcal{X} \otimes \mathcal{Y}'$, $\mathcal{X}' \otimes \mathcal{Y}$ and $\mathcal{X} \otimes \mathcal{Y}$ are defined. Let $X$, $Y'$, $X'$ and $Y$ are the corresponding index sets of their elementary automata and $SX$, $SY'$, $SX'$ and $SY$ the index sets of their state components. Additionally we assume that $SX \cap SY' = SX' \cap SY = SX \cap SY$. $LXY'$, $LX'Y$ as well as $LXY$ may denote the occurrence languages of $\mathcal{X} \otimes \mathcal{Y}'$, $\mathcal{X}' \otimes \mathcal{Y}$ and $\mathcal{X} \otimes \mathcal{Y}$ respectively.

**Theorem 5.5.** *If $R_X(LXY') = R_{X'}(LX'Y)$ and $R_{y'}(LXY') = R_Y(LX'Y)$, then $M_X(LXY) = M_X(LXY')$ and $M_Y(LXY) = M_Y(LX'Y)$.*

Let $\mathcal{X} \otimes \mathcal{Y}$ be a representation of the APA considered in the first two theorems, then $Y = \overline{X}$ and $L = LXY$. If $Y'$ and $X'$ are “simplified versions” of $Y$ and $X$ respectively then $LXY'$ and $LX'Y$ can be “less complex” (with an essentially smaller state space) than $L$. Now applying the above theorem $f(L)$ and $g(L)$ can be determined using $LXY'$ and $LX'Y$ instead of $L$ because $M_X\langle f$ and $M_{\overline{X}}\langle g$.

To derive simplicity of homomorphisms on $L$ from investigations on $LXY'$ and $LX'Y$ we need a “cooperating property” of APA [Och96]:

**Definition 5.6.** Let $LXY' \subset \Xi^*$ be the occurrence language of $\mathcal{X} \otimes \mathcal{Y}'$ and let $f : \Xi^* \to \Phi^*$ be a homomorphism. $\mathcal{X}$ is called *cooperative in* $\mathcal{X} \otimes \mathcal{Y}'$ *with respect to* $f$, if $M_X\langle f,\ \Phi \cap M_{Y'}(\Xi) = \emptyset$ and if for each $x \in LXY'$ there exists a finite subset $H \subset (f \oplus M_{Y'})(x)^{-1}((f \oplus M_{Y'})(LXY'))$ with $\epsilon \in H$ such that for each $u \in H$ either
$u^{-1}((f \oplus M_{Y'})(x^{-1}(LXY'))) = ((f \oplus M_{Y'})(x)u)^{-1}((f \oplus M_{Y'})(LXY'))$
or
$u^{-1}(H) \cap M_{Y'}(\Xi) = ((f \oplus M_{Y'})(x)u)^{-1}((f \oplus M_{Y'})(LXY')) \cap M_{Y'}(\Xi)$ and
$u^{-1}(H) \cap \Phi \neq \emptyset$ if $((f \oplus M_{Y'})(x)u)^{-1}((f \oplus M_{Y'})(LXY')) \cap \Phi \neq \emptyset$.

In combination with the previous theorem the following two theorems [Och96] allow to derive simplicity of $f$ and $g$ on $L = LXY$ from investigations on $LXY'$ and $LX'Y$.

**Theorem 5.7.** *Let* $r$ *and* $s$ *be homomorphisms with* $M_X\langle r\langle R_X$, $M_Y\langle s\langle R_Y$, $r(LXY') = M_{X'}(LX'Y)$ *and* $s(LX'Y) = M_{Y'}(LXY')$. *If* $\mathcal{X}$ *is cooperative in* $\mathcal{X} \otimes \mathcal{Y}'$ *with respect to* $r$ *and if* $\mathcal{Y}$ *is cooperative in* $\mathcal{X}' \otimes \mathcal{Y}$ *with respect to* $s$, *then* $r \oplus s$ *is simple on* $LXY$.

**Theorem 5.8.** *Let* $r$ *and* $s$ *be homomorphisms with* $M_X\langle r\langle R_X$, $M_Y\langle s$. *If* $r \oplus s$ *is simple on* $LXY$ *then* $M_X \oplus s$ *is simple on* $LXY$.

Using the partial order method developed in [Och97] $\mathcal{X}'$ and $\mathcal{Y}'$ can be computed efficiently on the basis of $\mathcal{X} \otimes \mathcal{Y}$ and simplicity of $f \oplus g$ on $L$ can be checked directly.

## 6. Temporal Logic and Abstraction

Our verification approach can also be combined with *temporal logic* [Nit94a, Nit94c, Nit94d, Nit94b, Nit95, Nit98]. In terms of temporal logic, the automaton of Figure 2 approximately satisfies the formula $\mathcal{G}(\mathcal{F}(RES))$ ($\mathcal{G}$: always-operator, $\mathcal{F}$: eventually-operator; thus $\mathcal{G}(\mathcal{F}(RES))$ means ”infinitely often result”), but the system in Figure 3 does not. This is indeed the case because the abstracting homomorphism is not simple. Using an appropriate type of *model checking*, approximate satisfaction of temporal logic formulae can be checked by the sh-verification tool.

The Algorithm for checking approximate satisfaction of propositional linear-time temporal logic formulae (PLTL- formulae) is based on the algorithm for linear satisfaction of PLTL-formulae by Gerth, Peled, Vardi, and Wolper [GPVW96]. The key construction of the algorithm is the construction of a Büchi-automaton $BA$ to a PLTL-formula $\eta$.

The *temporal logic formulae* (TL-formulae) we use are constructed as follows:

- *True* and *False* are atomic TL-formulae.

- The edge-labels of the automaton representing the concrete or abstracted behaviour of the system we are checking are atomic TL-formulae. In addition, $\varepsilon$ is an atomic TL-formula (atomic proposition). $\varepsilon$ is satisfied for a concrete action if and only if the action is mapped to the empty word by the abstraction.
- Formulae can be combined using the usual Boolean operators $\wedge$, $\vee$, $\neg$ and combinations thereof $\Rightarrow$ and $\Leftrightarrow$.
- Formulae can be combined using the usual temporal logic operators $\mathcal{G}$ (always), $\mathcal{F}$ (eventually), $\mathcal{U}$ (until), $\mathcal{B}$ (before) and $\mathcal{X}$ (next).
- Internally we use a temporal operator $\mathcal{V}$ which is the dual of the until-operator $\mathcal{U}$ ($\phi\mathcal{V}\psi$ is equivalent to $\neg((\neg\phi)\mathcal{U}(\neg\psi))$).
- In the automata that are generated during the model-checking optionally the before-operator $\mathcal{B}$ ($\phi\mathcal{B}\psi$ is equivalent $\neg((\neg\phi)\mathcal{U}\psi)$) can be used instead of $\mathcal{V}$ for better readability.

**Algorithm.** To check whether a behaviour $B$ satisfies the property $\mathrm{P}_\eta$ represented by a PLTL-formula $\eta$ approximately, one has to check whether $\mathrm{pre}(B) = \mathrm{pre}(B \cap \mathrm{P}_\eta)$. Herein, "pre(...)" designates the set of all finitely long prefixes of $\omega$-words in "...". Since $\mathrm{pre}(B) \supseteq \mathrm{pre}(B \cap \mathrm{P}_\eta)$ always holds, this can be reduced to $\mathrm{pre}(B) \subseteq \mathrm{pre}(B \cap \mathrm{P}_\eta)$. Algorithmically, we have to check whether $\mathrm{pre}(B)$ $\cap$ $\mathrm{C}(B \cap \mathrm{P}_\eta)$ is the empty set. "C(...)" denotes the complement of "..." with respect to $\Sigma^*$ ($\Sigma$ is the set of all actions of the system, i.e. the alphabet of the $\omega$-languages $B$ and $\mathrm{P}_\eta$).

An example for the automata used in the above construction as implemented in our tool is given in the appendix.

Our experience in practical examples shows that the combination of computing a minimal automaton of an LTS and model checking on this abstraction is significantly faster than direct model checking on the LTS.

The preservation result for approximately satisfied properties (Theorem 3.2) can be formulated in terms of PLTL using a syntactic transformation on PLTL-formulae [Nit94a, Nit98]. An example is given in section 8.

## 7. The Tool

As mentioned above, our verification method does not depend on a specific formal specification technique. For practical use the sh-verification tool has to be combined with a specification tool generating labeled transition systems. We have done this using the product net machine, and we are now implementing a specification environment based on asynchronous product automata (APA). Figure 11 shows the structure of the tool.

The product net machine is a tool for the design and analysis of product nets [Och91a]. Product nets [BOP89, OP95] are high level Petri nets with individual tokens. Figure 12 shows a product net specification of our client/server example, where the resource may eventually be locked forever. The shaded places are initially marked. In Figure 12 we do not use most of product nets' possible features. Indeed it is just a product net representation of a Petri net. The LTS

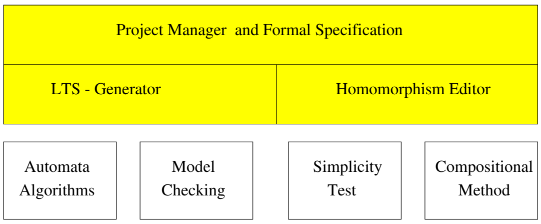


**Fig. 11.** The sh-verification tool

of Figure 3 is computed by the product net machine; it is the reachability graph of the product net in Figure 12.

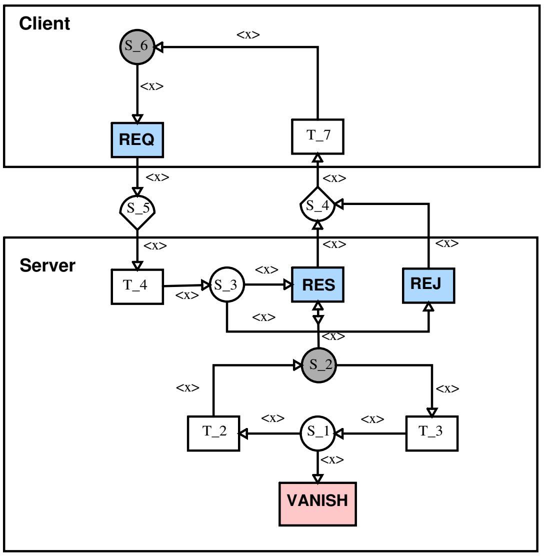


**Fig. 12.** Client Server Example

Practical experiences have been gained with large specifications, for example with ISDN-, XTP-, and smartcard protocols and by investigating service interactions in intelligent telecommunication systems [Klu92, Sch92, Gie93, Och93, OP93, Neb94, Och94a, OP95, CDGE$^{+}$96, CDF$^{+}$96]. Now our interest is focused on the verification of binding co-operations including electronic money and contract systems.

**Technical Requirements.** The sh-verification tool and the product net machine are both implemented in Allegro Common Lisp. The software is freely available (currently for Solaris and Windows NT) for non commercial purposes, but cannot be distributed via anonymous FTP, because of restrictions in the license agreement for the runtime library of the lisp system. For more information please contact the authors.

## 8. A Case Study

To demonstrate our method on a more realistic example we consider a model of the basic call process of an intelligent telephone network (IN). This call process, named the *basic call state model*(BCSM), is currently being standardized. This standardization process is structured in eight steps. Each step leads to a more detailed BCSM. These differently detailed standardization steps are called *capability sets* (CS). The currently standardized capability set is CS-1 (see [Q.1b] and especially [Q.1a]).

In this section, we verify a product net specification of the BCSM; throughout this section, we assume capability set CS-1 when briefly talking of BCSM [Q.1b, Q.1a]. We do not present the specification itself, but relate finite-state systems that we computed as abstractions of the BCSM specification's behaviour to automata descriptions in the standardization paper [Q.1a]. The complete specification can be found in [DFGE$^+$95]. Before starting with the verification steps, we give a brief introduction to the BCSM.

The BCSM handles the basic call process of an IN. This call process is internally structured, distinguishing caller and callee. The part of the BCSM related to a caller is named *originating* BCSM; abbreviated: O-BCSM. The callee oriented part is named *terminating* BCSM, or T-BCSM. Services in the IN, as for example call forwarding, are add-on features. The interface between BCSM and services is the *service logic*. The general structure of an IN, including caller and callee, is depicted in Figure 13. Dashes represent communication channels.

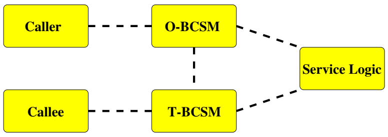


**Fig. 13.** The basic structure of an IN.

The BCSM is some kind of a finite-state system. The states are called *points in call* (PIC). Added to PIC are *detection points* (DP) from where the service logic may be invoked. The finite-state system that represents the BCSM is described graphically as well as textually in the standard [Q.1a]. A product net specification of the originating as well as the terminating BCSM was established in the SERVINT-project [NO95, DFGE$^+$95, DFGE$^+$96]. There, ambiguities in the standard, and contradictions between the textual and graphical description of the BCSM are resolved.

Two abstractions of the BCSM specification's behaviour leaving visible only the actions related to the O-BCSM and T-BCSM respectively, lead to exactly the resolved finite-state systems of the standard [Q.1a] representing O-BCSM and T-BCSM. Since all abstractions mentioned are obtained by applying an abstracting homomorphism that is simple on the concrete behaviour, approximately satisfied properties of the abstract behaviour represent corresponding approximately satisfied properties of the concrete behaviour. This observation, in principle, is sufficient to verify the correctness of the specification in comparison to [Q.1a]:

the components of the BCSM behave in their concrete environment in the same way as they would behave in an idealized environment. In the subsequent paragraphs, we look more closely at the T-BCSM's behaviour, checking explicitly some properties.

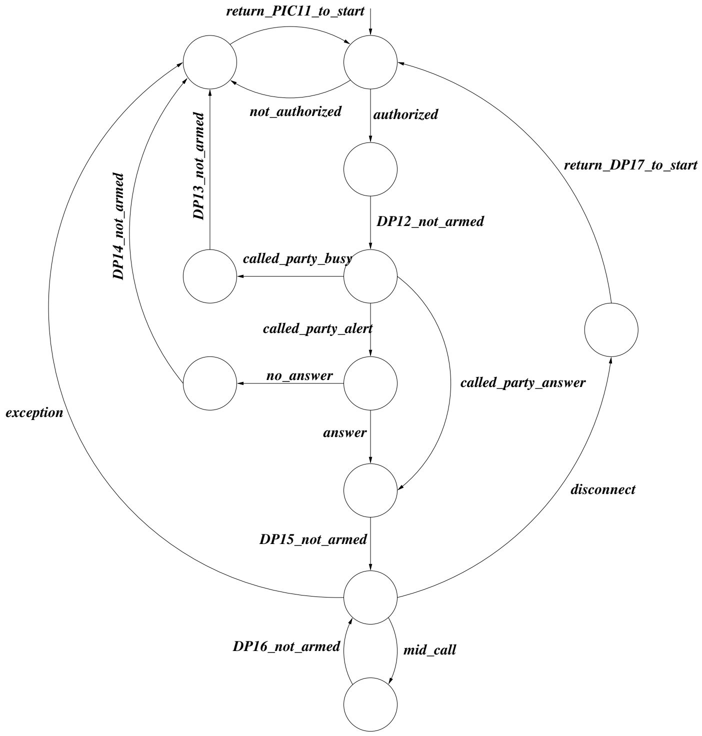


**Fig. 14.** T-BCSM.

After applying a homomorphism that is the identity function on actions relevant to the T-BCSM and that takes all other actions to the empty word, we obtain the finite-state system in Figure 14 that represents the abstract behaviour of the whole BCSM specification that is related to the T-part. As it is verified by our tool the abstracting homomorphism is simple on the concrete behaviour. It extracts the T-BCSM's behaviour when the T-BCSM is embedded in the environment that is the O-BCSM. We obtain exactly the behaviour as presented in the standard [Q.1a]. Consequently all properties that the standard demands the T-BCSM to satisfy are indeed satisfied for the T-BCSM in the BCSM specification. Nevertheless we check explicitly a property of the T-BCSM by firstly applying another abstraction step.

Figure 14 contains some actions that are interruptions to the straightforward calling process. These actions are *not_authorized*, *called_party_busy*, *no_answer*, and *exception*. Whenever one of these actions occurs, an exception handling is necessary. The exception handling is performed by PIC 11. Occurrence of the action *return_PIC11_to_start* indicates that a successful exception handling has taken place. To check the property "whenever an interruption of the calling process occurs, an exception handling takes place", we can define a suitable abstraction on the behaviour presented in Figure 14 that keeps visible the interruption and the exception handling, and check a suitable temporal logic formula on the abstract behaviour.

A suitable abstraction on T-BCSM's behaviour is defined by the homomorphism that maps *not_authorized*, *called_party_busy*, *no_answer*, and *exception* on the abstract action *interruption*, that maps action *return_PIC11_to_start* on the abstract action *exception_handling*, and that takes all other actions to the empty word. The resulting abstract behaviour is depicted in Figure 15.

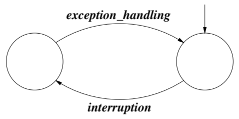


**Fig. 15.** An abstraction of the T-BCSM.

Obviously, $\mathcal{G}(interruption \Rightarrow \mathcal{X}\, exception\_handling)$ represents an approximately satisfied property of the behaviour presented in Figure 15. Let $h$ be the abstracting homomorphism on the concrete behaviour that generates the abstract behaviour in Figure 14 and let $h'$ be the abstracting homomorphism on this abstract behaviour that generates the more abstract behaviour presented in Figure 15. Because $h$ is simple on the concrete behaviour and $h'$ is simple on the behaviour presented in Figure 14 (both behaviours have a strongly connected finite-state representation), $h \circ h'$ is simple on the BCSM specification's behaviour (Theorem 3.6). According to the syntactic transformation of PLTL-formulae [Nit98] we obtain, that

$$\mathcal{G}(\varepsilon \vee (interruption \Rightarrow (\varepsilon\, \mathcal{U}(\neg\varepsilon \wedge \mathcal{X}(\varepsilon\, \mathcal{U}\, exception\_handling)))))$$

is an approximately satisfied property of the BCSM specification's behaviour. Simplification of this formula leads to

$$\mathcal{G}(interruption \Rightarrow \varepsilon\, \mathcal{U}\, exception\_handling).$$

If we interpret this result, we find that the reasonable computations of the BCSM specification (in this context the reasonable computations are once again the fair ones) satisfy the *correct exception handling property*. This illustrates how stepwise abstraction can be performed which can be regarded as an inverse stepwise refinement.

## 9. Conclusions

We have presented the basic functionality of the sh-verification tool in this article. The tool is equipped with the main features necessary to verify specifications of co-operating systems of industrial size. It comprises a satisfaction relation with an inherent fairness assumption and an abstraction concept adequate for the particular, practically useful satisfaction relation. Our verification method, which is based on the very general notions of approximate satisfaction of properties and simple language homomorphisms, does not depend on a specific formal specification method. It can be applied to all those specification techniques having an LTS-semantics.

Summarizing, using simple abstractions and approximate satisfaction verification can be done in two ways and is supported by our tool:

- System properties are explicitly given by temporal logic formulae or Büchi-automata. They can be checked on the abstract behaviour (under a simple homomorphism).
- Specifications of different abstraction levels are compared by corresponding simple homomorphisms. In that case system properties are given implicitly.

There exists a variety of verification tools which can be found in the literature. Some are model-checking based, others are proof system based. We consider COSPAN [Kur94] to be closest to the sh-verification tool. COSPAN is automata based and contains a homomorphism based abstraction concept. Since the transition labels of automata in COSPAN are in a Boolean algebra notation, the abstraction homomorphisms are Boolean algebra homomorphisms which correspond to non-erasing alphabetic language homomorphisms on the automata level. The sh-verification tool, in addition, offers erasing homomorphisms as an abstraction concept. COSPAN also considers only linear satisfaction of properties. Thus fairness assumptions need to be made explicitly in this tool. Besides many other tools we want to name only two more. Since it was one of the first verification tools, CESAR should be mentioned [QS82]. A tool which uses the modal $\mu$-calculus as a specification language for properties [Sti89] is the concurrency workbench [CPS93].

We consider the main strength of our tool to be the combination of an inherent fairness assumption in the satisfaction relation, an abstraction technique compatible with approximate satisfaction, and a suitable compositional and partial order method for the construction of only a partial state space. The sh-verification tool's user interface and general handling has reached a level of maturity that enabled its successful application in the industrial area [NO95, DFGE$^+$95, DFGE$^+$96].

## References

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[BD87] S. Budkowski and P. Dembinski. An introduction to estelle. *Computer Networks and ISDN-Systems*, 14:3–23, 1987.

[BOP89] H. J. Burkhardt, P. Ochsenschläger, and R. Prinoth. Product nets — a formal description technique for cooperating systems. GMD-Studien 165, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, September 1989.

[CDF$^+$96] C. Capellmann, R. Demant, F. Fatahi, R. Galvez-Estrada, U. Nitsche, and P. Ochsenschläger. Verification by behavior abstraction: A case study of service interaction detection in intelligent telephone networks. In *Computer Aided Verification (CAV) '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 466–469, New Brunswick, 1996.

[CDGE$^+$96] C. Capellmann, R. Demant, R. Galvez-Estrada, U. Nitsche, and P. Ochsenschläger. Case study: Service interaction detection by formal verification under behaviour abstraction. In Tiziana Margaria, editor, *Proceedings of International Workshop on Advanced Intelligent Networks'96*, pages 71–90, Passau, March 1996.

[CPS93] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based tool for the verification of finite-state systems. In *TOPLAS 15*, pages 36–72, 1993.

[DFGE$^{+}$95] R. Demant, F. Fatahi, R. Galvez-Estrada, U. Nitsche, and P. Ochsenschläger. Abschlußbericht des GMD-/Telekom-Projekts Formale Spezifikations- und Verifikationsmethoden zur Behandlung der Service-Interaction-Problematik – SERVINT. Abschluß bericht, GMD, Dezember 1995.

[DFGE$^{+}$96] R. Demant, F. Fatahi, R. Galvez-Estrada, U. Nitsche, and P. Ochsenschläger. Zwischenbericht des GMD-/Telekom-Projekts Formale Spezifikations- und Verifikationsmethoden zur Behandlung der Service-Interaction-Problematik – SERVINT2. Zwischenbericht, GMD, Juli 1996.

[Eil74] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.

[Gie93] H. Giehl. Verifikation von Smartcard-Anwendungen mittels Produktnetzen. GMD-Studien 225, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, 1993.

[GPVW96] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembinski and M. Sredniawa, editors, *Protocol Specification, Testing, and Verification XV '95*, pages 3–18. Chapman & Hall, 1996.

[Klu92] W. Klug. OSI-Vermittlungsdienst und sein Verhältnis zum ISDN-D-Kanalprotokoll. Spezifikation und Analyse mit Produktnetzen. Arbeitspapiere der GMD 676, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, 1992.

[Kur94] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes.* Princeton University Press, Princeton, New Jersey, first edition, 1994.

[Neb94] M. Nebel. Ein Produktnetz zur Verifikation von Smartcard-Anwendungen in der STARCOS-Umgebung. GMD-Studien 234, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, 1994.

[Nit94a] U. Nitsche. Propositional linear temporal logic and language homomorphisms. In Anil Nerode and Yuri V. Matiyasevich, editors, *Logical Foundations of Computer Science '94, St. Petersburg*, volume 813 of *Lecture Notes in Computer Science*, pages 265–277. Springer Verlag, 1994.

[Nit94b] U. Nitsche. Simple homomorphisms and linear temporal logic. Arbeitspapiere der GMD 889, GMD – Forschungszentrum Informationstechnik, Darmstadt, Dezember 1994.

[Nit94c] U. Nitsche. A verification method based on homomorphic model abstraction. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, page 393, Los Angeles, 1994. ACM Press.

[Nit94d] U. Nitsche. Verifying temporal logic formulas in abstractions of large reachability graphs. In J. Desel, A. Oberweis, and W. Reisig, editors, *Workshop: Algorithmen und Werkzeuge für Petrinetze.* Humboldt Universität Berlin, 1994.

[Nit95] U. Nitsche. A finitary language semantics for propositional linear temporal logic (abstract). In *Preproceedings of the 2nd International Conference on Developments in Language Theory.* University of Magdeburg, 1995.

[Nit98] U. Nitsche. *Verification of Co-Operating Systems and Behaviour Abstraction.* PhD thesis, University of Frankfurt, Germany, 1998.

[NO95] U. Nitsche and P. Ochsenschläger. Zwischenbericht des GMD-/Telekom-Projekts Formale Spezifikations- und Verifikationsmethoden zur Behandlung der Service-Interaction-Problematik – SERVINT. Zwischenbericht, GMD, Juli 1995.

[NO96] U. Nitsche and P. Ochsenschläger. Approximately satisfied properties of systems and simple language homomorphisms. *Information Processing Letters*, 60:201–206, 1996.

[NW97] U. Nitsche and P. Wolper. Relative liveness and behavior abstraction (extended abstract). In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, Santa Barbara, CA, 1997.

[Och88] P. Ochsenschläger. Projektionen und reduzierte Erreichbarkeitsgraphen. Arbeitspapiere der GMD 349, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, Dezember 1988.

[Och90] P. Ochsenschläger. Modulhomomorphismen. Arbeitspapiere der GMD 494, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, Dezember 1990.

[Och91a] P. Ochsenschläger. Die Produktnetzmaschine. *Petri Net Newsletter*, 39:11–31, August 1991. Also appeared as a GMD Arbeitspapier Nr. 505, 1991.

[Och91b] P. Ochsenschläger. Modulhomomorphismen II. Arbeitspapiere der GMD 597, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, November 1991.

[Och92] P. Ochsenschläger. Verifikation kooperierender Systeme mittels schlichter Homomorphismen. Arbeitspapiere der GMD 688, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, Oktober 1992.

[Och93] P. Ochsenschläger. Verifikation verteilter Systeme mit Produktnetzen. *PIK*, 16:42–43, 1993.

[Och94a] P. Ochsenschläger. Kompositionelle Verifikation kooperierender Systeme. Arbeitspapiere der GMD 885, GMD – Forschungszentrum Informationstechnik, Darmstadt, Dezember 1994.

[Och94b] P. Ochsenschläger. Verification of cooperating systems by simple homomorphisms using the product net machine. In J. Desel, A. Oberweis, and W. Reisig, editors, *Workshop: Algorithmen und Werkzeuge für Petrinetze*, pages 48–53. Humboldt Universität Berlin, 1994.

[Och94c] P. Ochsenschläger. Verifikation von Smartcard-Anwendungen mit Produktnetzen. In *Tagungsband des 4. SmartCard Workshops*, Darmstadt, 1994.

[Och95] P. Ochsenschläger. Compositional verification of cooperating systems using simple homomorphisms. In J. Desel, H. Fleischhack, A. Oberweis, and M. Sonnenschein, editors, *Workshop: Algorithmen und Werkzeuge für Petrinetze*, pages 8–13. Universität Oldenburg, 1995.

[Och96] P. Ochsenschläger. Kooperationsprodukte formaler Sprachen und schlichte Homomorphismen. Arbeitspapiere der GMD 1029, GMD – Forschungszentrum Informationstechnik, Darmstadt, 1996.

[Och97] P. Ochsenschläger. Schlichte Homomorphismen auf präfixstabilen partiell kommutativen Sprachen. Arbeitspapiere der GMD 1106, GMD – Forschungszentrum Informationstechnik, Darmstadt, 1997.

[OP93] P. Ochsenschläger and R. Prinoth. Formale Spezifikation und dynamische Analyse verteilter Systeme mit Produktnetzen. In *Informatik aktuell Kommunikation in verteilten Systemen*, pages 456–470, München, 1993. Springer Verlag.

[OP95] P. Ochsenschläger and R. Prinoth. *Modellierung verteilter Systeme – Konzeption, Formale Spezifikation und Verifikation mit Produktnetzen.* Vieweg, Wiesbaden, 1995.

[ORRN97] P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche. The SH-verification tool. In *Proceedings of the 2nd International Workshop on Formal Methods for Industrial Critical Systems (FMICS'97)*, Cesena, Italy, 1997.

[Q.1a] Draft Revised ITU-T Recommendation Q.1214: *Distributed Functional Plane for Intelligent Network CS-1.* March 1995.

[Q.1b] ITU-T Recommendations Q.12xx – Q series: *Intelligent Network Recommendation.* 1992.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. volume 137 of *Lecture Notes in Computer Science*, pages 337–351, 1982.

[Sch92] S. Schremmer. ISDN-D-Kanalprotokoll der Schicht 3. Spezifikation und Analyse mit Produktnetzen. Arbeitspapiere der GMD 640, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, 1992.

[SSR89] R. Saracco, J. R. W. Smith, and R. Reed. *Telecommunication Systems' Engineering using SDL.* North Holland, 1989.

[Sti89] C. Stirling. An introduction to modal and temporal logics for CCS. In A. Yonezawa and T. Ito, editors, *Concurrency: Theory, Language, and Architecture*, volume 391 of *Lecture Notes in Computer Science.* Springer Verlag, 1989.

[Zie89] W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In *LNCS 363.* Springer Verlag, 1989.

## Appendix

As an example for the temporal logic algorithms described in chapter 6 we describe the steps performed by our tool to check whether the property $\mathcal{G}(\mathcal{F}(RES))$

is satisfied approximately by the "behaviour-automaton" of Figure 3.

On the automaton level, we have to perform 7 Steps:

1. Compute a Büchi-Automaton representing the property given by a PLTL-formula according to [GPVW96]. For the formula $\mathcal{G}(\mathcal{F}(RES))$ which represents the property that "a result $RES$ is infinitely often produced" the automaton construction is represented in two steps in Figure 16 and Figure 17.

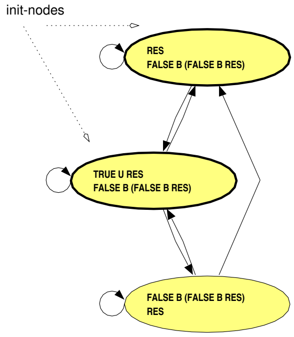


**Fig. 16.** Graph for $\mathcal{G}(\mathcal{F}(RES))$

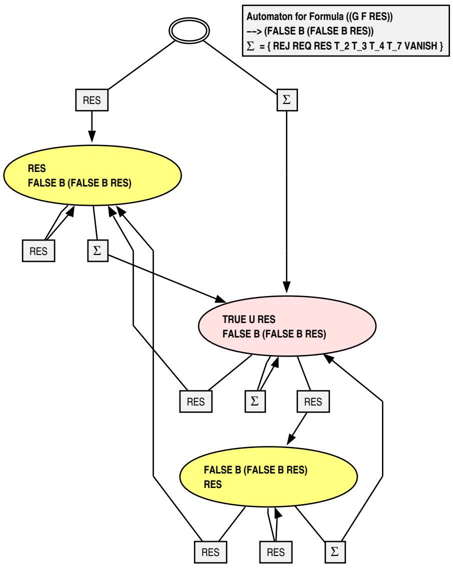

Automaton for Formula ((G F RES))
---> (FALSE B (FALSE B RES))
Σ = { REJ REQ RES T_2 T_3 T_4 T_7 VANISH }
RES
Σ
RES
FALSE B (FALSE B RES)
RES
Σ
TRUE U RES
FALSE B (FALSE B RES)
RES
Σ
RES
FALSE B (FALSE B RES)
RES
RES
RES
Σ

**Fig. 17.** Automaton for $\mathcal{G}(\mathcal{F}(RES))$

2. Construct the synchronous product of the automaton constructed so far and the automaton representing the behaviour of the considered system. The synchronous product is the construction of the intersection of languages on the automaton level. For the "property- automaton" of Figure 17 and the

"behaviour-automaton" of Figure 3, the "product-automaton" is represented in Figure 18.

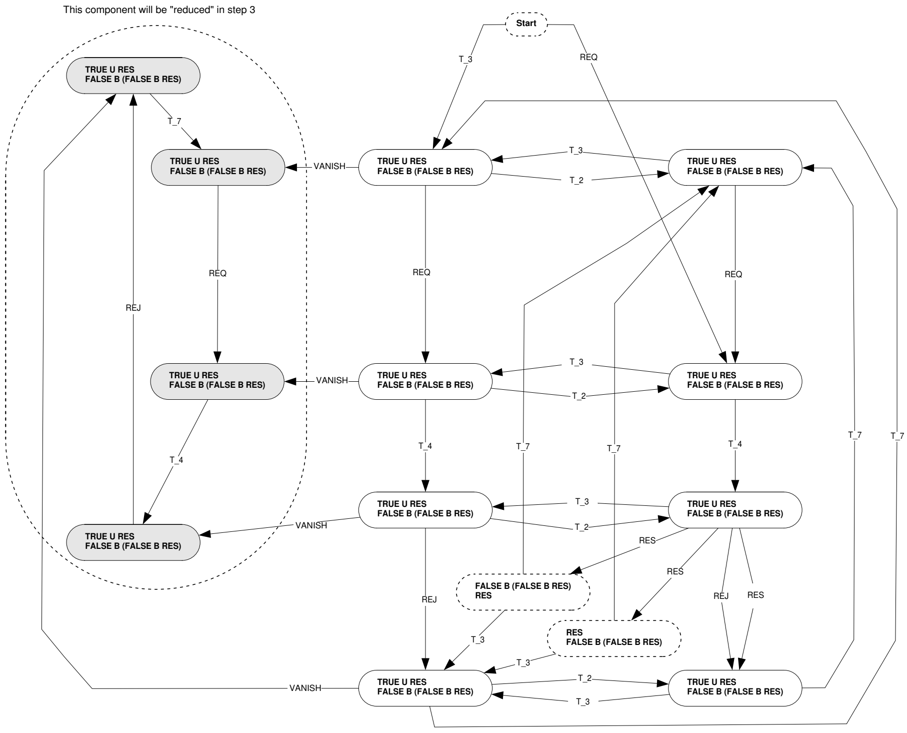


**Fig. 18.** The synchronous product automaton of Figure 3 and Figure 17

3. Reduce the resulting Büchi-Automaton (remove all states which are not reachable from the initial state or from which no cycle containing an accepting state is reachable).
4. Ignore acceptance conditions (make all states accepting) and do not interpret the automaton anymore as an automaton on infinite ($\omega$-) words but as one on finitely long words (this corresponds to the prefix construction ("pre(...)").
5. Construct the complement automaton (for a finite-word automaton, not an $\omega$-automaton).
6. Construct the intersection with the automaton representing the behaviour.
7. Check whether the resulting automaton is empty (does not accept words).

For the considered example of the behaviour in Figure 3 and the property given by the PLTL-formula $\mathcal{G}(\mathcal{F}(RES))$, the behaviour satisfies the property approximately.

Besides the algorithm described above that checks for approximate satisfaction we also have implemented algorithms for other kinds of satisfaction relations (PLTL and AGEF [Nit98]).

# P2

# THE SH-VERIFICATION TOOL

| | |
|---|---|
| **Title** | The SH-Verification Tool |
| **Authors** | Peter Ochsenschläger, Jürgen Repp and Roland Rieke |
| **Publication** | In *Proc. 13th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, pages 18–22, 2000. |
| **ISBN/ISSN** | ISBN 0-1-57735-113-4 |
| **URL** | `http://www.aaai.org/Papers/FLAIRS/2000/FLAIRS00-004.pdf` |
| **Status** | Published |
| **Publisher** | AAAI Press |
| **Publication Type** | Conference Proceedings (FLAIRS 2000) |
| **Copyright** | 2000, American Association for Artificial Intelligence `http://www.aaai.org` |
| **Contribution of Roland Rieke** | Co-Author with significant contribution, editor, and presenter at the 13th International Florida Artificial Intelligence Research Society Conference.<br>Specific contributions are: (1) temporal logic formula editor; (2) conception and implementation of an application oriented user-interface for input of cryptographic formulae and presentation of results in this syntax.<br>Related contributions: Roland Rieke also contributed to a related paper "Abstraction and composition – a verification method for co-operating systems" [Ochsenschläger et al., 2000b] that he presented at the same conference. He further contributed to an invited jounal paper "Verification of Cooperating Systems – An Approach Based on Formal Languages" [Ochsenschläger et al., 2000]. The approach published in Ochsenschläger et al. [2000] is supported by the SHVT as presented in P1 and this paper P2. |

Table 7: Fact Sheet Publication *P2*

Publication P2 [Ochsenschläger, Repp & Rieke, 2000a] addresses the following research question:

RQ1 *How can it be proven that components of cooperating systems securely work together?*

This paper gives an overview about the main components of the SHVT. With the help of an illustrative example, the usage of the methods described in P1 is shown. P2 contributes to research question RQ1 by the demonstration of the applicability of the methods developed in P1. Specifically, abstraction and temporal logic based reasoning is demonstrated. The SHVT's user interface and general handling has reached a level of maturity that enabled its successful application in the industrial area [Apel et al., 2007].

# The SH-Verification Tool

**Peter Ochsenschläger** and **Jürgen Repp** and **Roland Rieke**
SIT – Institute for Secure Telecooperation,
GMD – German National Research Center for Information Technology,
Rheinstr. 75, D-64295 Darmstadt, Germany
E-Mail: {ochsenschlaeger,repp,rieke}@darmstadt.gmd.de

## Abstract

The sh-verification tool supports a verification method for cooperating systems based on formal languages. It comprises computing abstractions of finite-state behaviour representations as well as automata and temporal logic based verification approaches. A small but typical example shows the steps for analysing its dynamic behaviour using the sh-verification tool.

**Keywords:** Cooperating Systems; Finite State Systems, Abstraction; Simple Language Homomorphisms; Formal Specification; Verification

## Introduction

The sh-verification tool [1] supports the method for verification of cooperating systems described in (Ochsenschläger, Repp, Rieke 1999). The reader is referred to this paper for notations, definitions and theorems. Figure 1 shows the structure of the tool. The main components of the system are the tools for specification, the analysis kernel, the tools for abstraction and the project manager. It is possible to extend the tool by different application oriented user interfaces. A small but typical example shows the steps for analysing a systems behaviour using the sh-verification tool.

## Specification

The presented verification method does not depend on a specific formal specification technique. For practical use the sh-verification tool has to be combined with a specification tool generating labeled transition systems LTS [2]. The current implementation uses product nets [3] (Burkhardt, Ochsenschläger, Prinoth 1989; Ochsenschläger Prinoth 1995) as specification environment. A second specification environment based on asynchronous product automata (APA), (Ochsenschläger *et al.* 1998) is planned.

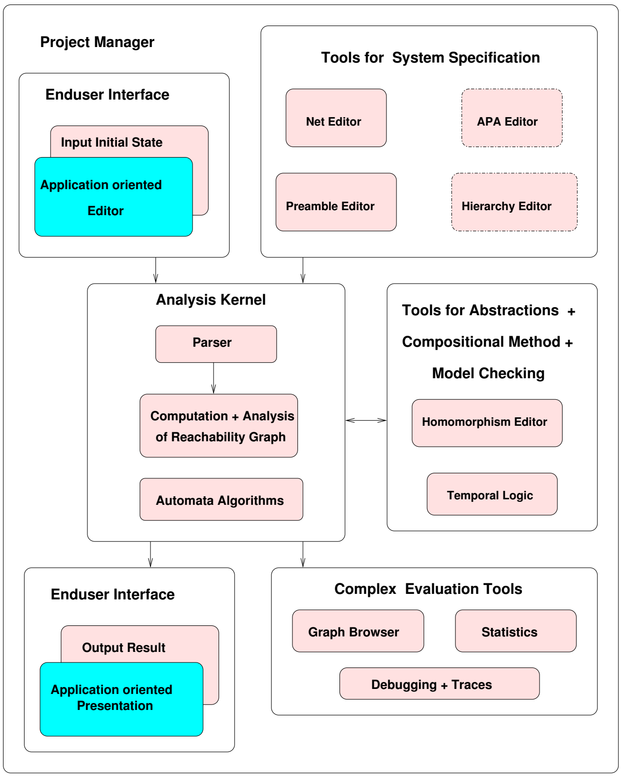


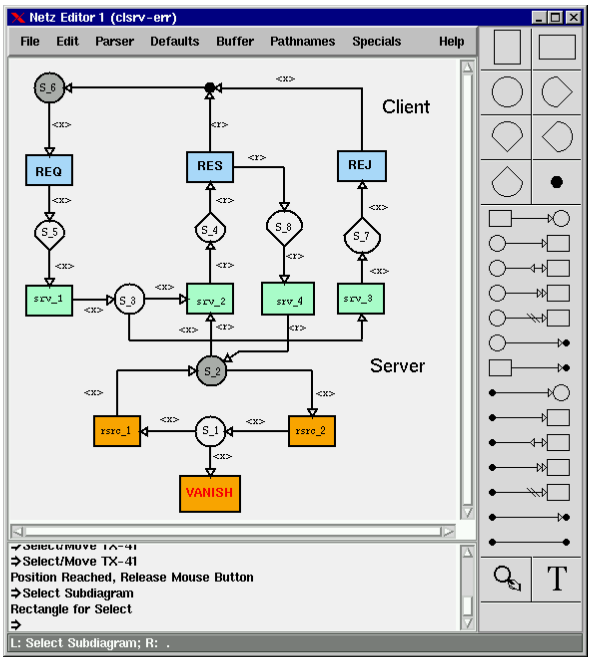
Figure 1: Components of the sh-verification tool

To illustrate the usage of the methods described in (Ochsenschläger, Repp, Rieke 1999) we consider an example of a system that consists of a client and a server as its main components. The client sends requests *REQ* to the server, expecting the server to produce particular results. Nevertheless, for some reasons, the server may not always respond to a request by sending

[1] sh abbreviates simple homomorphism

[2] The semantics of formal specification techniques for distributed systems is usually based on LTS.

[3] a special class of high level petri nets



➔Select/Move TX-41
➔Select/Move TX-41
Position Reached, Release Mouse Button
➔Select Subdiagram
Rectangle for Select
➔

L: Select Subdiagram; R: .

Figure 2: Client Server Example

a result *RES*, but may, as well, reject a request *REJ* (Figure 4).

Figure 2 shows a product net specification of this example. It is a global model for the systems behaviour. Note that the resource may eventually be locked forever. In Figure 2 we do not use most of product nets' possible features. Indeed, it is a product net representation of a Petri net.

Usually complex systems are specified hierarchically. This is supported by the *project manager* of the tool. (In our simple example the specification is flat.)

The LTS in Figure 3, which is the reachability graph of the product net in Figure 2, is computed by the tool. This LTS consists of two strongly connected components (marked by diffent colors). Usually the LTS of a specification is too complex for a complete graphical presentation; there are several features to inspect the LTS.

## Abstraction

In the example the important actions with respect to the client's behaviour, are sending a request and receiving a result or rejection.

We will regard the whole system running properly, *if the client, at no time, is prohibited completely from receiving a result after having sent a request* (correctness criterion).

For the moment, we regard the server as a black box; i.e. we neither consider its internal structure nor look

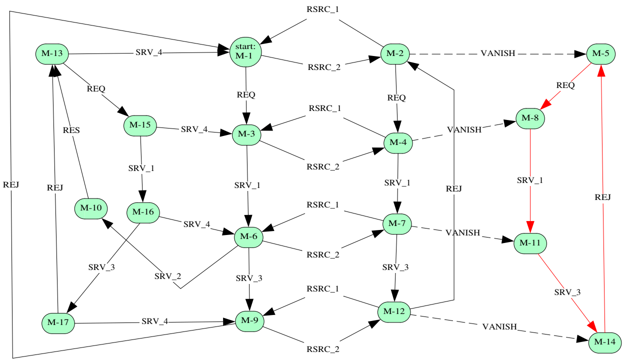


Figure 3: LTS

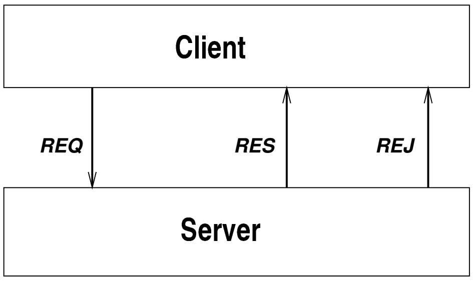


Figure 4: Client Server Abstract View

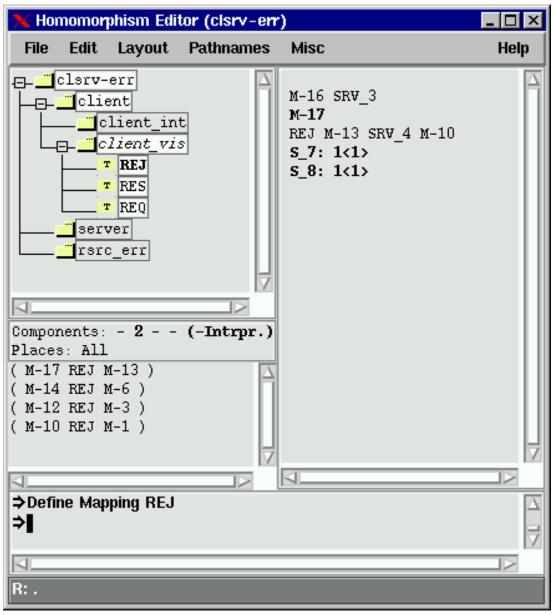
at its internal actions. Not caring about particular actions of a specification when regarding the specification's behaviour is *behaviour abstraction*. If we define a suitable abstraction for the client/server system with respect to our correctness criterion, we only keep actions *REQ*, *RES*, and *REJ* visible, hiding all other actions. This is supported by the homomorphism editor of the tool (Figure 5).

An automaton [4] representing the abstract behaviour of the specification can be computed by the sh-verification tool (Figure 6). It obviously satisfies the required property. The next step is to check whether the concrete behaviour also satisfies the correctness requirement mentioned above. For that purpose we have to prove simplicity of the defined homomorphism.

Simplicity of an abstraction can be investigated inspecting the strongly connected components of the LTS by a sufficient condition (Ochsenschläger, Repp, Rieke 1999). The component graph in Figure 7 (combined with the homomorphic images of the arc labels of the corresponding graph components) does not satisfy this condition, so nothing can be said about simplicity.

We now try to refine the homomorphism such that the sufficient condition for simplicity can be proven. Inspecting the edge between the two nodes of the com-

[4] *the minimal automaton*



```
M-16 SRV_3
M-17
REJ M-13 SRV_4 M-10
S_7: 1<1>
S_8: 1<1>
```

```
Components: - 2 - - (-Intrpr.)
Places: All
( M-17 REJ M-13 )
( M-14 REJ M-6 )
( M-12 REJ M-3 )
( M-10 REJ M-1 )
```

```
➔Define Mapping REJ
➔
R: .
```

Figure 5: Defining an Abstraction

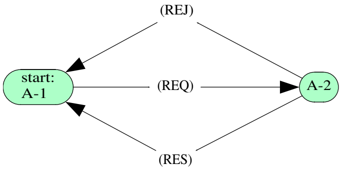


Figure 6: Minimal Automaton.

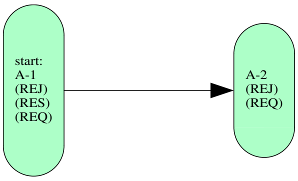


Figure 7: Component Graph

ponent graph shows that the action $VANISH$ causes the transitions between this two components (Figure 8). The refined homomorphism, which additionally keeps $VANISH$ visible, satisfies the sufficient condition for simplicity. Figure 9 shows the corresponding automaton. This automaton obviously violates the required property, so the systems behaviour does not satisfy this property.

These simplicity investigations, which are supported by the tool, detect the error in the specification. In (Ochsenschläger 1992; 1994a) a necessary condition for simplicity is given. It is based on so called deadlock languages and shows non-simplicity of our $REQ$-$RES$-$REJ$-homomorphism (Ochsenschläger *et al.* 1998).

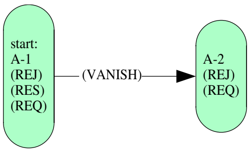


Figure 8: Component Graph

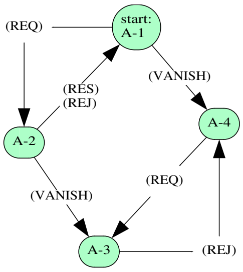


Figure 9: Minimal Automaton (with VANISH)

To handle the well known state space explosion problem *a compositional method* (Ochsenschläger 1996) is implemented in the sh-verification tool. This approach can also be used iteratively and provides a basis for induction proofs in case of systems with several identical components (Ochsenschläger 1996). Using our compositional method a connection establishment and release protocol has been verified by investigating automata with about 100 states instead of 100000 states.

## Temporal Logic

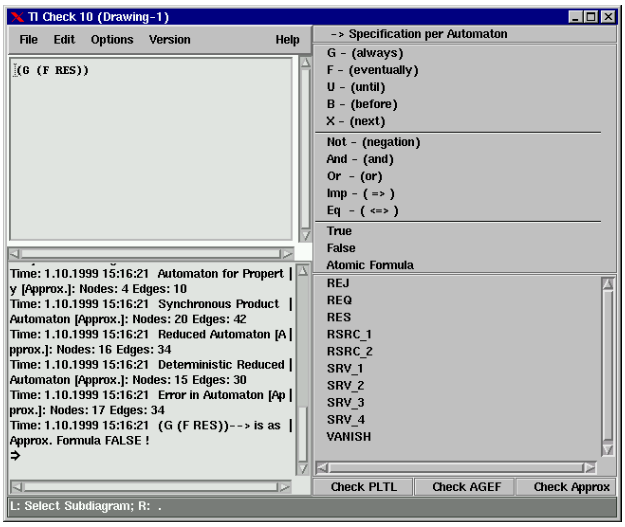
Our verification approach can also be combined with *temporal logic* (Ochsenschläger *et al.* 1998). In terms of temporal logic, the automaton of Figure 6 approximately satisfies (Ochsenschläger *et al.* 1998) the formula $\mathcal{G}(\mathcal{F}(RES))$ ($\mathcal{G}$: always-operator, $\mathcal{F}$: eventually-operator; thus $\mathcal{G}(\mathcal{F}(RES))$ means "infinitely often result"), but the system in Figure 3 does not. This is indeed the case because the abstracting homomorphism is not simple. Using an appropriate type of *model checking*, approximate satisfaction of temporal logic formulae can be checked by the sh-verification tool.

Our experience in practical examples shows that the combination of computing a minimal automaton of an LTS and model checking on this abstraction is significantly faster than direct model checking on the LTS.

## Applications

Practical experiences have been gained with large specifications:

- ISDN and XTP protocols (Klug 1992; Schremmer 1992; Ochsenschläger Prinoth 1993)



```
(G (F RES))
```

```
Time: 1.10.1999 15:16:21  Automaton for Propert |
y [Approx.]: Nodes: 4 Edges: 10
Time: 1.10.1999 15:16:21  Synchronous Product  |
Automaton [Approx.]: Nodes: 20 Edges: 42
Time: 1.10.1999 15:16:21  Reduced Automaton [A |
pprox.]: Nodes: 16 Edges: 34
Time: 1.10.1999 15:16:21  Deterministic Reduced |
Automaton [Approx.]: Nodes: 15 Edges: 30
Time: 1.10.1999 15:16:21  Error in Automaton [Ap |
prox.]: Nodes: 17 Edges: 34
Time: 1.10.1999 15:16:21  (G (F RES))--> is as  |
Approx. Formula FALSE !
⇒
```

Figure 10: Temporal Logic Formula Editor

- Smartcard systems (Nebel 1994; Ochsenschläger 1994b)
- Service interactions in intelligent telecommunication systems (Capellmann *et al.* 1996b; 1996a).
- The tool has also been applied to the analysis of cryptographic protocols (Basak 1999; Rudolph 1998). In this context an application oriented user-interface has been developed for input of cryptographic formulae and presentation of results in this syntax.
- Currently our interest is focused on the verification of binding cooperations including electronic money and contract systems. Recently some examples in that context have been investigated with our tool (Fox 1998; Roßmann 1998).

## Technical Requirements

The sh-verification tool is implemented in Allegro Common Lisp. An interpreter-based version of the software is freely available (currently for Solaris, Linux and Windows NT) for non commercial purposes (http://sit.gmd.de/META/projects.html). For investigation of large systems a compiler-based version of the tool is needed. For more information please contact the authors.

## Conclusions

We have presented the basic functionality of the sh-verification tool in this article. The tool is equipped with the main features necessary to verify specifications of cooperating systems of industrial size. It supports a verification method based on formal languages (Ochsenschläger, Repp, Rieke 1999).

There exists a variety of verification tools which can be found in the literature. Some are based on model checking, others use proof systems. We consider COSPAN (Kurshan 1994) to be closest to the sh-verification tool. COSPAN is automata based and contains a homomorphism based abstraction concept. Since the transition labels of automata in COSPAN are in a Boolean algebra notation, the abstraction homomorphisms are Boolean algebra homomorphisms which correspond to non-erasing alphabetic language homomorphisms on the automata level. The sh-verification tool, in addition, offers erasing homomorphisms as an abstraction concept. COSPAN also considers only linear satisfaction of properties. Thus fairness assumptions need to be made explicitly in this tool. A tool which uses the modal $\mu$-calculus as a specification language for properties (Stirling 1989) is the concurrency workbench (Cleaveland, Parrow, Steffen 1993). In (Hartel *et al.* 1999) ten tools in this area including ours are compared.

We consider the main strength of our tool to be the combination of an inherent fairness assumption in the satisfaction relation, an abstraction technique compatible with approximate satisfaction, and a suitable compositional and partial order method for the construction of only a partial state space. The sh-verification tool's user interface and general handling has reached a level of maturity that enabled its successful application in the industrial area.

## References

Basak, G. 1999. Sicherheitsanalyse von Authentifizierungsprotokollen – model checking mit dem SH-Verification tool. Diploma thesis, University of Frankfurt.

Burkhardt, H. J.; Ochsenschläger, P.; and Prinoth, R. 1989. Product nets — a formal description technique for cooperating systems. GMD-Studien 165, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Capellmann, C.; Demant, R.; Fatahi, F.; Galvez-Estrada, R.; Nitsche, U.; and Ochsenschläger, P. 1996a. Verification by behavior abstraction: A case study of service interaction detection in intelligent telephone networks. In *Computer Aided Verification (CAV) '96*, volume 1102 of *Lecture Notes in Computer Science*, 466–469.

Capellmann, C.; Demant, R.; Galvez-Estrada, R.; Nitsche, U.; and Ochsenschläger, P. 1996b. Case study: Service interaction detection by formal verification under behaviour abstraction. In Margaria, T., ed., *Proceedings of International Workshop on Advanced Intelligent Networks'96*, 71–90.

Cleaveland, R.; Parrow, J.; and Steffen, B. 1993. The concurrency workbench: A semantics-based tool for the verification of finite-state systems. In *TOPLAS 15*, 36–72.

Fox, S. 1998. Sezifikation und Verifikation eines Separation of Duty-Szenarios als verbindliche Telekoopertation im Sinne des Gleichgewichtsmodells. GMD Research Series 21, GMD – Forschungszentrum Informationstechnik, Darmstadt.

Hartel, P.; Butler, M.; Currie, A.; Henderson, P.; Leuschel, M.; Martin, A.; Smith, A.; Ultes-Nitsche, U.; and Walters, B. 1999. Questions and answers about ten formal methods. In *Proc. 4th Int. Workshop on Formal Methods for Industrial Critical Systems*, volume II, 179–203. Pisa, Italy: ERCIM.

Klug, W. 1992. OSI-Vermittlungsdienst und sein Verhältnis zum ISDN-D-Kanalprotokoll. Spezifikation und Analyse mit Produktnetzen. Arbeitspapiere der GMD 676, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Kurshan, R. P. 1994. *Computer-Aided Verification of Coordinating Processes*. Princeton, New Jersey: Princeton University Press, first edition.

Nebel, M. 1994. Ein Produktnetz zur Verifikation von Smartcard-Anwendungen in der STARCOS-Umgebung. GMD-Studien 234, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Ochsenschläger, P., and Prinoth, R. 1993. Formale Spezifikation und dynamische Analyse verteilter Systeme mit Produktnetzen. In *Informatik aktuell Kommunikation in verteilten Systemen*, 456–470. München: Springer Verlag.

Ochsenschläger, P., and Prinoth, R. 1995. *Modellierung verteilter Systeme – Konzeption, Formale Spezifikation und Verifikation mit Produktnetzen*. Wiesbaden: Vieweg.

Ochsenschläger, P.; Repp, J.; Rieke, R.; and Nitsche, U. 1998. The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing* 10:381–404.

Ochsenschläger, P.; Repp, J.; and Rieke, R. 1999. Verification of Cooperating Systems – An Approach Based on Formal Languages. Submitted to *FLAIRS-2000* Special Track on Validation, Verification & System Certification.

Ochsenschläger, P. 1992. Verifikation kooperierender Systeme mittels schlichter Homomorphismen. Arbeitspapiere der GMD 688, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Ochsenschläger, P. 1994a. Verification of cooperating systems by simple homomorphisms using the product net machine. In Desel, J.; Oberweis, A.; and Reisig, W., eds., *Workshop: Algorithmen und Werkzeuge für Petrinetze*, 48–53. Humboldt Universität Berlin.

Ochsenschläger, P. 1994b. Verifikation von Smartcard-Anwendungen mit Produktnetzen. In Struif, B., ed., *Tagungsband des 4. GMD-SmartCard Workshops*. GMD Darmstadt.

Ochsenschläger, P. 1996. Kooperationsprodukte formaler Sprachen und schlichte Homomorphismen. Arbeitspapiere der GMD 1029, GMD – Forschungszentrum Informationstechnik, Darmstadt.

Roßmann, J. 1998. Formale Analyse der Business-Phase des First Virtual Internet Payment Systems basierend auf Annahmen des Gleichgewichtsmodells. Diploma thesis, University of Frankfurt.

Rudolph, C. 1998. Analyse krypotgraphischer Protokolle mittels Produktnetzen basierend auf Modellannahmen der BAN-Logik. GMD Research Series 13/1998, GMD – Forschungszentrum Informationstechnik GmbH.

Schremmer, S. 1992. ISDN-D-Kanalprotokoll der Schicht 3. Spezifikation und Analyse mit Produktnetzen. Arbeitspapiere der GMD 640, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Stirling, C. 1989. An introduction to modal and temporal logics for CCS. In Yonezawa, A., and Ito, T., eds., *Concurrency: Theory, Language, and Architecture*, volume 391 of *Lecture Notes in Computer Science*. Springer Verlag.

# P3

# DEVELOPMENT OF FORMAL MODELS FOR SECURE E-SERVICES

| | |
|---|---|
| **Title** | Development of formal models for secure e-services |
| **Authors** | Roland Rieke |
| **Publication** | In *Eicar Conference 2003*. |
| **URL** | `http://sit.sit.fraunhofer.de/smv/publications/download/Eicar-2003.pdf` |
| **Status** | Published |
| **Publisher** | |
| **Publication Type** | Conference |
| **Copyright** | |
| **Contribution of Roland Rieke** | Author and presenter at the Eicar Conference 2003. |

Table 8: Fact Sheet Publication *P3*

Publication P3 [Rieke, 2003] addresses the following research question:

RQ1 *How can it be proven that components of cooperating systems securely work together?*

This paper provides an extensive example for the use of the methods and tool described in P1 and P2. From e-government applications provided by project partners from the city of Cologne a typical example of an e-service implementation was selected. This e-service was modelled, augmented by an attacker model, and analysed using the SHVT. It has been shown that even if the correct behaviour of an e-service is proven under assumptions about the interfaces to the environment and about reasonable input it is necessary to inspect the system behaviour and ask 'what if' questions to check the behaviour of the model against given attack patterns or slightly changed assumptions about the environment. A vulnerability - a race condition problem - was found that leads in the end to a misrouting effect. Race conditions are just the most security-relevant type of concurrency problem [Viega & McGraw, 2002].

SIT

Fraunhofer Institut
Sichere Telekooperation

# Development of formal models for secure e-services

Roland Rieke
SIT – Fraunhofer - Institute for Secure Telecooperation,
Rheinstr. 75, D-64295 Darmstadt, Germany
E-Mail: rieke@sit.fraunhofer.de

January 10, 2003

**Abstract**

A methodology for the development of formal models for e-services is presented. Verification of the correct behaviour when given expected input and check for security properties by adding selected attack patterns is shown. An example scenario of a typical e-service configuration is given and the dynamic behaviour of different variants is analysed. To improve security of a system providing a collection of e-services it is essential to make each e-service secure using a design and verification method based on formal methods and tools [1].

## 1 Introduction

The goal of the development of formal models of e-services is to achieve a systematic and verifiable improvement of security of the system providing the services.

Reliable security primitives already exist, but the security of complex applications essentially depends on the correct and consistent interplay of security

[1] This work was funded by the “Bundesministerium für Bildung und Forschung” in the context of the SKe project (http://www.ske-projekt.de/).

primitives and resource management.

To verify the correctness of a given e-service a formal model of its components and their interplay is usually analysed by computing its dynamic behaviour and automatically inspecting the generated state space for postulated safety and liveness properties.

To additionally prove some selected security properties one can add the formal specification of a potential attacker to the given model and check if the security properties hold.

In section 2 the used methods and tool are presented. The methodology for development and evaluation of formal security models for e-services is described.

In section 3 the selected portal scenario and analysed security properties are informally described and formally specified using the presented method and tool. After finding a violation of a security property an extended version of the portal scenario using an additional cryptographic protocol is analysed.

In section 4 some implemented features to support attack simulations on formal models are described.

Finally the results of this study are summarised and some perspectives for further development are presented.

# 2 Outline of methodology for the development of formal models for e-services

## 2.1 Methods and tool used

Modelling is based on asynchronous product automata (APA), a flexible operational specification concept for cooperating systems [10]. APA are supported by the SH verification tool developed at Fraunhofer SIT [9, 11]. The tool provides components for the complete cycle from formal specification to exhaustive validation.

An APA can be seen as a family of elementary automata. The set of all possible states of the whole APA is structured as a product set; each state is divided into state components. In the following the set of all possible states is called state set. The state sets of elementary automata consist of components of the state set of the APA. Different elementary automata are “glued” by shared components of their state sets. Elementary automata can “communicate” by changing shared state components [6].

A small example to illustrate how APA can be used to specify a system and how to explore the computed reachability graph with the SH verification tool is presented in the appendix.

## 2.2 Methodology for the development of formal models for e-services

In the context of modelling an appropriate abstraction level must be chosen, so that a verification of the relevant security measures is still possible. In refinements of the model, where a complete analysis is not possible for complexity reasons, interesting parts of the search space can still be explored by manual control using the simulation mode. Different models of possible attackers can be included in the specification and the combined model can be explored to find states where an attack succeeds.

The development of an executable formal model for an e-service application requires the steps shown in figure 1.

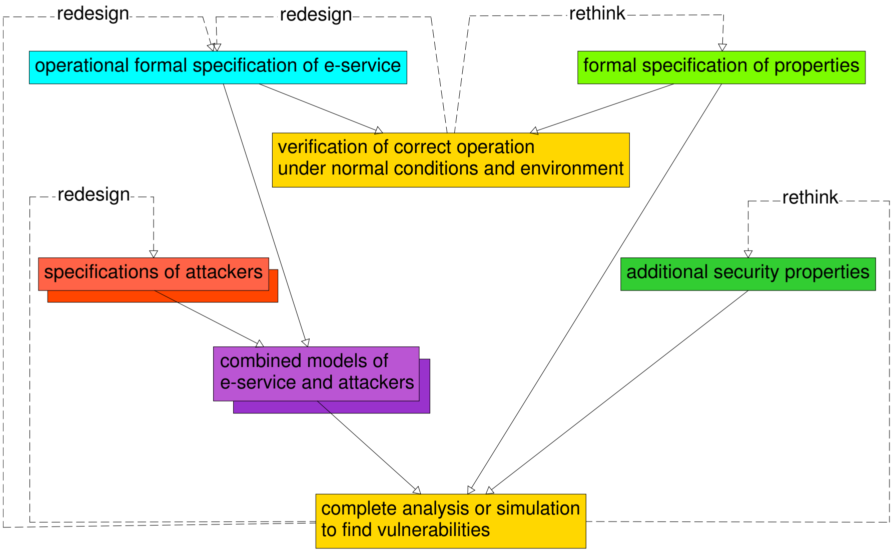

redesign
redesign
rethink
operational formal specification of e-service
formal specification of properties
verification of correct operation
under normal conditions and environment
redesign
rethink
specifications of attackers
additional security properties
combined models of
e-service and attackers
complete analysis or simulation
to find vulnerabilities

Figure 1: Development process for robust e-service models

**Operational formal specification of the e-service**

Derive an operational formal specification of the e-service from an informal specification of the required functionality (for example an UML model). Transfer the formal specification into the SH verification tool.
Note that specifications on different abstraction levels can be compared.

**Formal specification of properties**

System properties are explicitly given by break-conditions, temporal logic formulae or Büchi-automata. Temporal logic formulae can be checked on the abstract behaviour (under a simple homomorphism). A method for checking approximate satisfaction of properties fits exacly to the built-in simple homomorphism check [11].

**Verification of correct operation under normal conditions and environment**

- To find errors early in the analysis the check for given conditions during the computation of the reachability graph is implemented. Many safety properties (what happens is not wrong) can be checked this way.
- Computation of strongly connected components is very fast [8] and gives good insight into liveness behaviour (eventually something desired happens) of the model.
- Model checking can be used to search for particular states describing a violation of a security property.

**Specifications of attackers**

Adequate attackers [2] have to be specified here. But what kind of attacks should be considered here ?

A number of threat classes from the X.509 standard that computer networks face are detailed in [13]. These threat classes are: Identity interception, masquerade, replay, data interception, manipulation, repudiation, denial of service, misrouting and traffic analysis.

Many of these threats can be avoided in a given application scenario by choosing a decent cryptographic protocol for communication.

**Aditional security properties**

Security is not a single property of a protocol or an e-service. Depending on precisely what capabilities an attacker has, different properties for the system model have to be proven.

**Combined models of e-service and attackers**

The SH verification tool automatically "glues" together selected parts of e-service model and attacker. This is supported by the project management component of the tool.

**Complete analysis or simulation to find vulnerabilities**

If the attacker has too many alternatives the state space of the composition of the e-service specification and the attacker specification and their complex interplay may become too big to compute the complete behaviour. So we additionally need to be able to inspect selected parts of the state space. Simulation of typical paths of the reachability graph of the formal model under development is very useful and can be compared to the debugger used to develop software using standard programming

[2] Strong versus weak attacker:
A weak attacker might be able to start a replay attack, a very simple form of attack where some sequence of events or commands is observed, and then replayed in the attempt to trick the server to perform some action so that some vulnerability of the protocol can be exploited. A strong attacker might be able to manipulate the power line on the infrastructure or start some sort of denial of service attack to enforce a "reset" of an e-service. A very strong attacker might even be able to reboot the server with a completely different operating system.

languages. If attack patterns are already known simulation of those attacks in the extended formal model can clarify if the model resists this threat.

# 3 Development of a model for the portal scenario

Looking at some typical configurations that we modelled for usage in e-government applications we selected a typical example of an e-service implementation (called portal scenario) to develop a formal model using the above presented methodology.

An APA model of the portal scenario (see figure 2) except the firewall was implemented using the graphic editor of the SH verification tool. Domains of state components and functions used are defined in textual form in the preamble syntax of the tool.

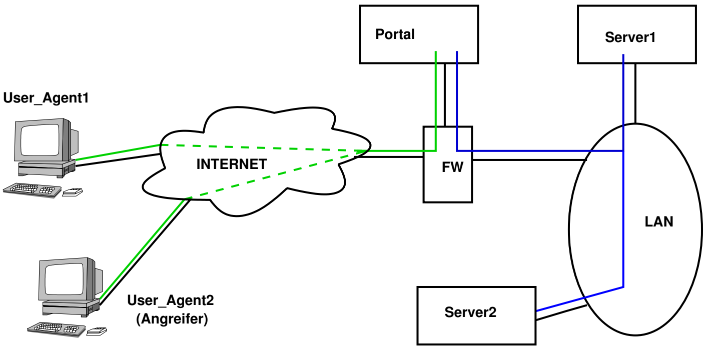


Figure 2: Portal scenario

## 3.1 Evaluation of portal scenario with attack simulation

A first version of portal scenario was entered into the SH verification tool together with the attacker model shown in figure 3. This is a weak attacker, it has the ability to log into the e-service as a normal client and tries to get some information sent to another client by just behaving like a normal client except it is reading everything it can get.

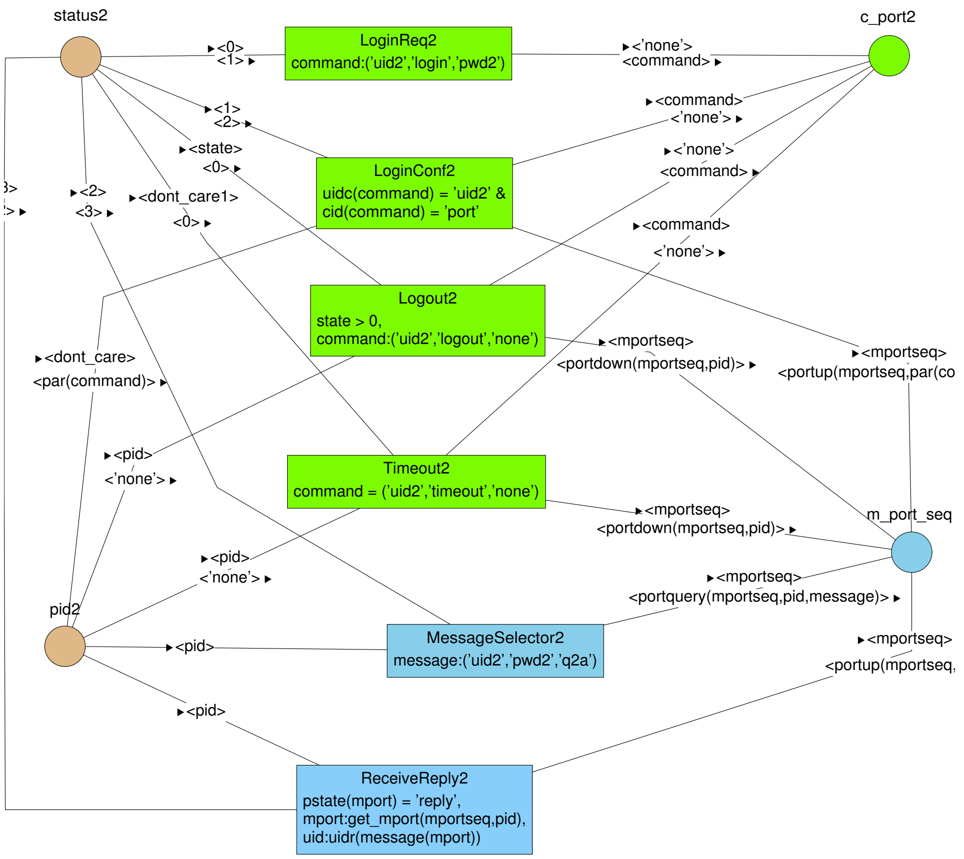
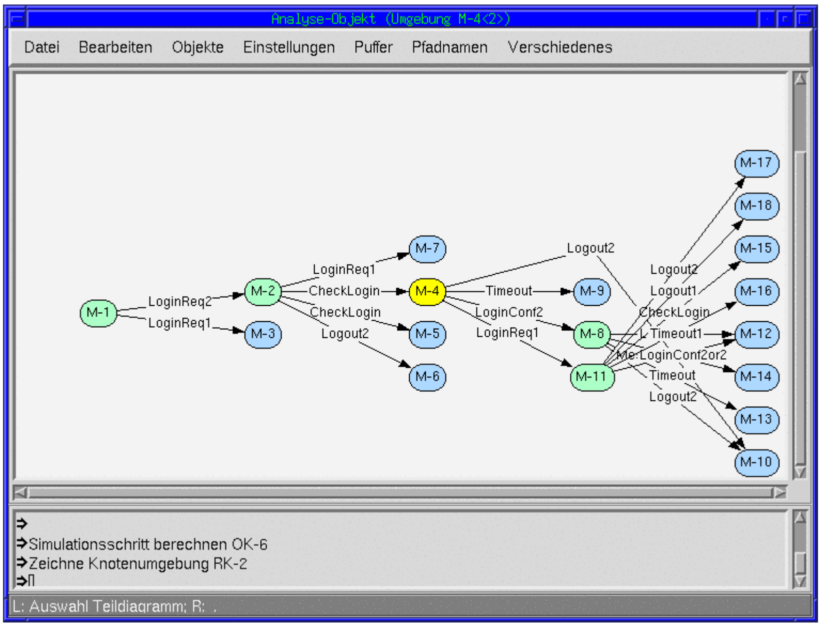


Figure 3: APA specification of attacker

Figure 4 shows the graphical interface to the simulation component of the tool during simulation of the portal model.



```
➔
➔Simulationsschritt berechnen OK-6
➔Zeichne Knotenumgebung RK-2
➔
```

Figure 4: Graphical simulation using SH verification tool

After checking some properties about the correct behaviour without the attacker, the attacker was added and the following security property was specified:

*No data from the server database produced for one agent must be delivered to another agent (the potential attacker).*

Trying to verify this security property a sequence of 13 steps was found that breaks the property and constitutes an attack (see figure 5).

This threat can be classified as *misrouting*. This is possible because it is assumed that there is no end-to-end protocol between client and server but different protocols for client-portal and portal-server communication. Now some problems with local management of routing information make the attack shown here possible.

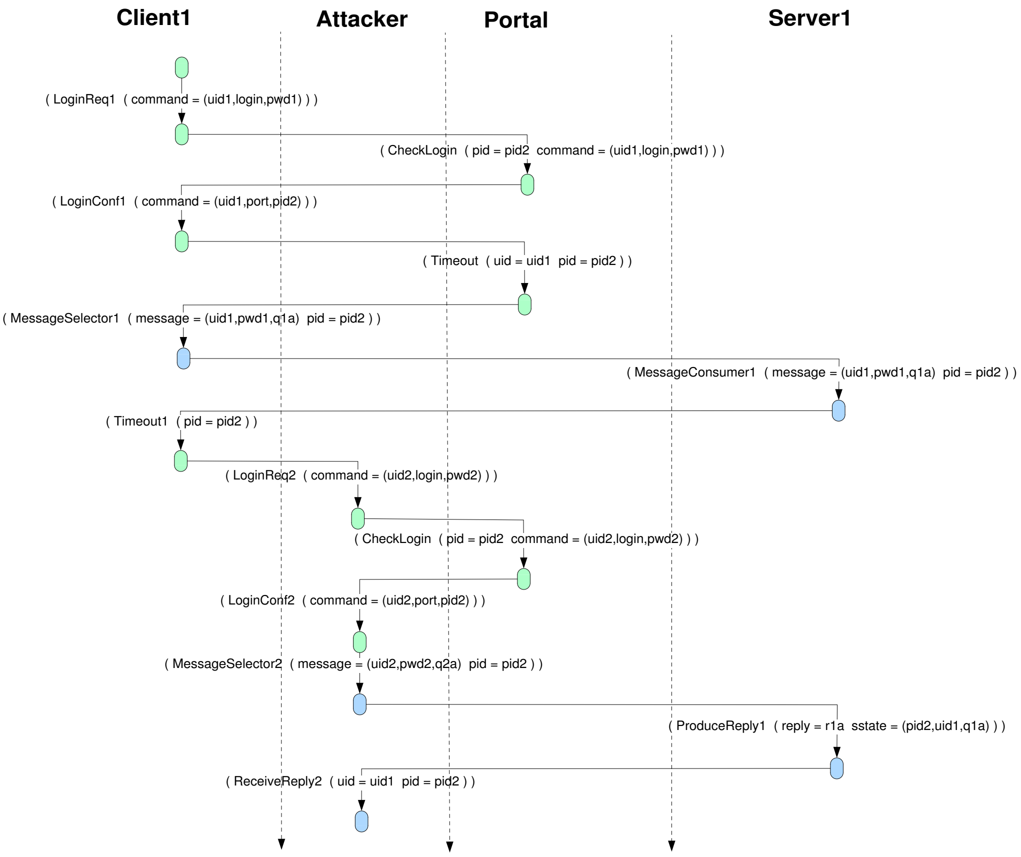

Client1
Attacker
Portal
Server1
( LoginReq1 ( command = (uid1,login,pwd1) ) )
( CheckLogin ( pid = pid2 command = (uid1,login,pwd1) ) )
( LoginConf1 ( command = (uid1,port,pid2) ) )
( Timeout ( uid = uid1 pid = pid2 ) )
( MessageSelector1 ( message = (uid1,pwd1,q1a) pid = pid2 ) )
( MessageConsumer1 ( message = (uid1,pwd1,q1a) pid = pid2 ) )
( Timeout1 ( pid = pid2 ) )
( LoginReq2 ( command = (uid2,login,pwd2) ) )
( CheckLogin ( pid = pid2 command = (uid2,login,pwd2) ) )
( LoginConf2 ( command = (uid2,port,pid2) ) )
( MessageSelector2 ( message = (uid2,pwd2,q2a) pid = pid2 ) )
( ProduceReply1 ( reply = r1a sstate = (pid2,uid1,q1a) ) )
( ReceiveReply2 ( uid = uid1 pid = pid2 ) )

Figure 5: Steps of successful attack

The integrated algorithms for computation of minimal automata [4] in the SH verification tool can be used to compute the local behaviour of the agents, the portal and the server (see figure 6).

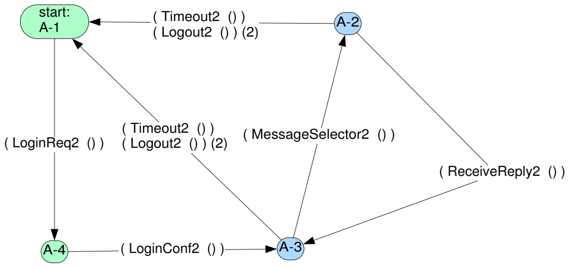


Figure 6: Local behaviour of attacker

On the basis of the computed graphs of the local behaviour of protocol agents it can be checked whether the APA model at this level of abstraction fits to the predefined behaviour of the protocol participants at another level given for example by an UML model.

### 3.2 Portal scenario with cryptographic protocol

The portal scenario was enhanced by adding a cryptographical protocol (abstracted) for client-portal communication and again possible attacks were searched. Figure 7 shows an overview of the components. A situation where two servers are computing answers for different queries, the portal has stored two different keys for client-portal communications and client2 (the attacker) has posted a query is shown.

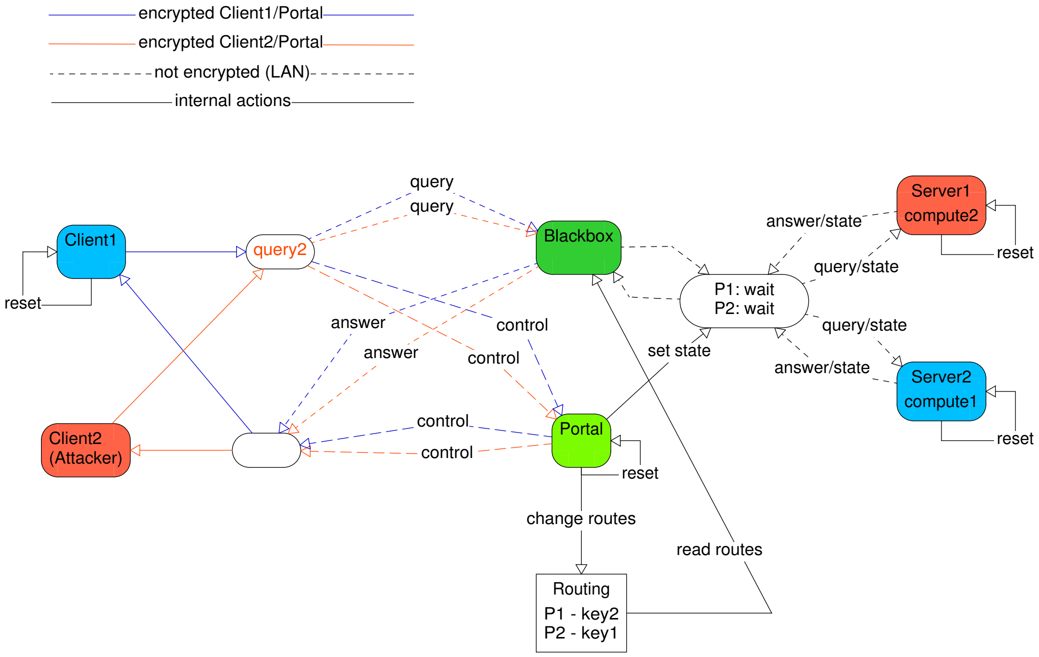
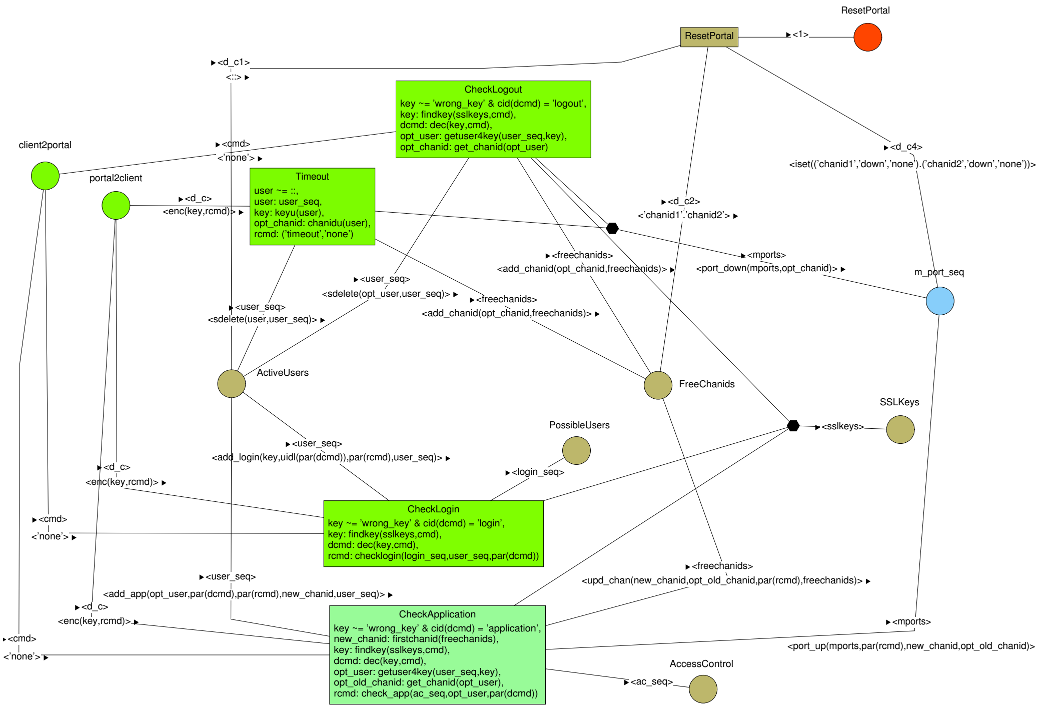


Figure 7: Portal scenario with cryptographic protocol

Figure 8 shows an APA specification of the enhanced portal component.



ResetPortal
ResetPortal
<1>
<d_c1>
<::>
CheckLogout
key ~= 'wrong_key' & cid(dcmd) = 'logout',
key: findkey(sslkeys,cmd),
dcmd: dec(key,cmd),
opt_user: getuser4key(user_seq,key),
opt_chanid: get_chanid(opt_user)
<d_c4>
<iset(('chanid1','down','none').('chanid2','down','none'))>
client2portal
<cmd>
<'none'>
portal2client
<d_c>
<enc(key,rcmd)>
Timeout
user ~= ::,
user: user_seq,
key: keyu(user),
opt_chanid: chanidu(user),
rcmd: ('timeout','none')
<d_c2>
<'chanid1'.'chanid2'>
<freechanids>
<add_chanid(opt_chanid,freechanids)>
<mports>
<port_down(mports,opt_chanid)>
m_port_seq
<user_seq>
<sdelete(opt_user,user_seq)>
<freechanids>
<add_chanid(opt_chanid,freechanids)>
<user_seq>
<sdelete(user,user_seq)>
ActiveUsers
FreeChanids
SSLKeys
<sslkeys>
PossibleUsers
<user_seq>
<add_login(key,uidl(par(dcmd)),par(rcmd),user_seq)>
<login_seq>
<d_c>
<enc(key,rcmd)>
CheckLogin
key ~= 'wrong_key' & cid(dcmd) = 'login',
key: findkey(sslkeys,cmd),
dcmd: dec(key,cmd),
rcmd: checklogin(login_seq,user_seq,par(dcmd))
<cmd>
<'none'>
<freechanids>
<upd_chan(new_chanid,opt_old_chanid,par(rcmd),freechanids)>
<user_seq>
<add_app(opt_user,par(dcmd),par(rcmd),new_chanid,user_seq)>
<d_c>
<enc(key,rcmd)>
CheckApplication
key ~= 'wrong_key' & cid(dcmd) = 'application',
new_chanid: firstchanid(freechanids),
key: findkey(sslkeys,cmd),
dcmd: dec(key,cmd),
opt_user: getuser4key(user_seq,key),
opt_old_chanid: get_chanid(opt_user),
rcmd: check_app(ac_seq,opt_user,par(dcmd))
<mports>
<port_up(mports,par(rcmd),new_chanid,opt_old_chanid)>
<cmd>
<'none'>
AccessControl
<ac_seq>

Figure 8: APA specifying the portal component

Communication channels protected by strong cryptography make poor targets. Attackers like to go after the programs at either end of a secure communications link because the end points are typically easier to compromise or try to utilise knowledge about faulty implementations of strong protocols or weak configuration at one side. If a strong protocol is used but some guidelines to use the protocol securely are not followed or some faulty implementation is used it is possible to break even a strong protocol like SSLv3 [12]. For example an attacker can try as a man-in-the-middle to downgrade a client/server pair to use a weaker version of the protocol or a weaker crypto suite and then exploit the known weaknesses.

In the enhanced portal scenario the attack described in section 3.1 was found again but in this case data delivered to the attacker are encrypted (with key of original receiver). So it seems that a weakened form of the given security property is sufficient:

*No data from the server database produced for one agent must be delivered to another agent (the potential attacker) except encrypted data that the attacker cannot decrypt.*

Nevertheless another more complex attack was found in the given model. It is a sequence of 21 steps including a reset of the portal and the cooperation of a second server.

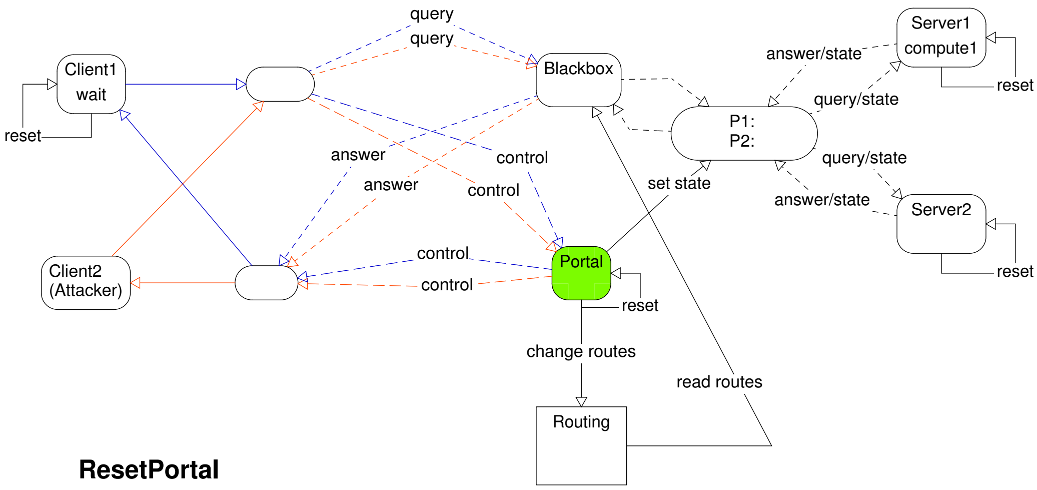


Figure 9: Attack sequence (step 10)

Figure 9 shows a situation after the reset of the portal component where client1 has posted a query and is waiting for an answer, the portal has an empty routing and crypto-key table and no state information and server1 is computing an answer for the previous query of client1.

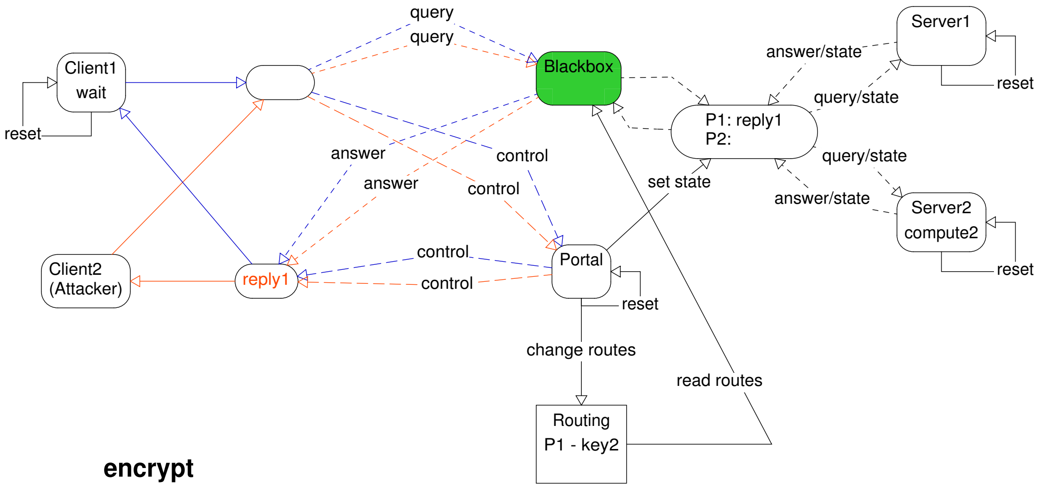


Figure 10: Attack sequence (step 21)

Figure 10 shows a follow-up situation after client2 has exchanged a key with the portal, logged in, posted a query that server2 is processing. Now server1 has produced a reply to the previous query (before the reset) of client1 and the portal assigns the wrong key from its routing table to this reply and directs the blackbox to forward the encrypted reply to client2.

The vulnerability found here can also be seen as a race condition problem that leads in the end to the misrouting effect. “Race conditions are just the most security-relevant type of concurrency problem.”[14]

# 4 Implemented features to support the development process of formal models

To implement support for the development process of formal models including debugging and attack simulations the following features and modifications have been implemented within the SH verification tool.

**Visualisation of simulation paths on graphical presentation:** Simple navigation through simulation paths by mouse-clicks is implemented.

**Different views:** APA can be viewed on different levels to hide unnecessary details.

**Pattern based specification of components:** Components of same type (for example several servers or clients with same functionality) can be specified once and instantiated many times [5]. A parser and compiler for pattern based specification have been implemented.

**Invariants (break-conditions):** To find errors early in the analysis, the check for given conditions during the computation of the reachability graph is implemented. So computation automatically stops if a state that matches a given condition (violation of invariant) is found.

**Project management:** A very flexible selection of variants of analysis scenarios was implemented. In a project tree components can be selected and deselected by mouse-click so it is easy for example to exchange libraries of symbolic crypto functions and analyse different versions and combinations of formal models.
Remark about composition of components:
It would be desirable to be able to verify different components of the e-service architecture separately and then combine the proofs to get less state space explosion during the verification. In the analysed examples however the functionality of the components is abstracted to a level where all modelled functionality influences the behaviour at the interface of the component, so it cannot (at least with our methods) be hidden somehow.

**Split state components:** Make it possible to insert an “intermediate layer” of attackers in the specification without changing the specification of the state components.

A feature found to be useful but not yet implemented is, to find out if attackers have a winning strategy against the other components of the modelled system. Alternating time temporal logic [3] would be useful for this purpose because it offers selective quantification over those paths that are possible outcomes of games, such as the game in which the system and the environment alternate moves. Currently we only find that for example there is a state where some invariant is broken, but not if attackers alone can enforce the whole system to reach that state.

**Related work**
The Murphi verification system [1] for example is a similar finite-state analysis tool but as far as we know it only implements fully automatic model checking and has no interactive graphical simulation mode as described in this work. For a comparison of an older version of the tool with other formal methods and tools see [7].

## 5 Conclusions

This study shows that even if the correct behaviour of an e-service is proven under assumptions about the interfaces to the environment and about reasonable input it is necessary to inspect the system behaviour and ask "what if" questions to check the behaviour of the model against given attack patterns or slightly changed assumptions about the environment.

Therefore a tool that can be used as sort of debugger on formal models is extremely helpful for the development process especially if the robustness of the model against given attacks is to be inspected and verified. If new attack methods are detected later it should be easy to check for vulnerabilities of the model by adding an appropriate module or intercepting some protocol. The SH verification tool with some additional features and modifications described in section 4 has been successfully applied for that purpose.

We have applied a similar approach to formal modelling and verification of security policy interaction issues in the MakoSi project [2] where the e-service modelled was an electronic whiteboard within a distributed collaborative engineering environment.

We are working to improve the current approach in the following ways:

More example scenarios will be analysed to find and classify common attack patterns that can be provided in standard libraries or example collections.

Features found to be useful during evaluation of new scenarios will be implemented within the SH verification tool. It would be nice for example to automatically find and check "similar" simulation paths when having changed some details of the specified system.

If follow-up projects support it, possible new methods for example to reduce the number of states that are explored when analysing the model or some theorem proving assistance will be implemented.

### Acknowledgements

I am grateful to the members of our research group META for previous work on APA and fruitful discussions on the subject and especially to Jürgen Repp for implementing most of the simulation support in the SH verification tool.

## References

[1] http://verify.stanford.edu/dill/murphi.html.

[2] http://www.makosi.de.

[3] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, Florida, October 1997.

[4] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.

[5] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and Provability - a Formal Framework. GMD Report 150, GMD – Forschungszentrum Informationstechnik GmbH, 2001.

[6] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and Provability - a Formal Framework. In *Infrastructure Security Conference 2002*, October 2002. Copyright: ©2002, Springer Verlag.

[7] P. Hartel, M. Butler, A. Currie, P. Henderson, M. Leuschel, A. Martin, A. Smith, U. Ultes-Nitsche, and B. Walters. Questions and answers about ten formal methods. In *Proc. 4th Int. Workshop on Formal Methods for Industrial Critical Systems*, volume II, pages 179–203, Pisa, Italy, July 1999. ERCIM, STAR/CNR.

[8] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass., first edition, 1979.

[9] P. Ochsenschläger, J. Repp, and R. Rieke. The SH-Verification Tool. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, pages 18–22, Orlando, FL, USA, May 2000. AAAI Press.

[10] Peter Ochsenschläger, Jürgen Repp, and Roland Rieke. Abstraction and composition – a verification method for co-operating systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 12:447–459, June 2000.

[11] Peter Ochsenschläger, Jürgen Repp, Roland Rieke, and Ulrich Nitsche. The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing, The International Journal of Formal Method*, 11:1–24, 1999.

[12] Eric Rescoria. *SSL and TLS: Designing and Building Secure Systems.* Addison-Wesley, Boston, 2001.

[13] Danny Smith. Selected Aspects of Computer Security in Open Systems. http://auscert.org.au/render.html?it=2255&cid=1920, The University of Queensland, 1993.

[14] John Viega and Gary McGraw. *Building Secure Software.* Addison-Wesley Professional Computer Series, Boston, 2002.

## 6 Appendix

A small example is used to illustrate how APA can be used to specify a system and how to explore the computed reachability graph with the SH verification tool. Let us assume we want to solve the following problem:

Given the puzzle in figure 11 construct an APA that computes all possible positions reachable by shifting numbered squares to the empty square from the initial state shown in figure 11 on the left and find out if the state on the right is reachable.

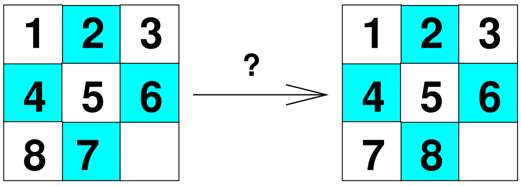


Figure 11: Is it possible to change positions of the *8* and *7*

The idea is to represent the actual state of the puzzle by 9 state components corresponding to the 9 locations in the puzzle and to model the shifting of a square by a state transition of an elementary automaton between each two positions.

Each elementary automaton has the form given in figure 12. This graphical representation shows an elementary automaton named $A\_1\_2$ with two neighbour state components $S1$ and $S2$. The circles represent state components and a box corresponds to one elementary automaton. The full specification of an APA includes the transition relations of the elementary automata and the initial state. A state transition of automaton $A\_1\_2$ may only change the content of directly connected state components $S1$ and $S2$ representing two neighbour positions in the 8-puzzle example.

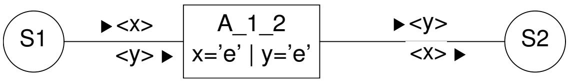


Figure 12: The elementary automaton A_1_2

In this example $x$ is bound to the content of $S1$ and $y$ to the content of $S2$. The inscription `x = 'e' | y = 'e'` in the box represents a restriction for the possible transitions of $A\_1\_2$. If one of the state components contains the value 'e' representing the empty square then the value of the other state component can be moved to this state component and vice versa.

The whole APA for this example is given in figure 13. Note that the squares with numbers are not part of the APA they just illustrate the initial state.

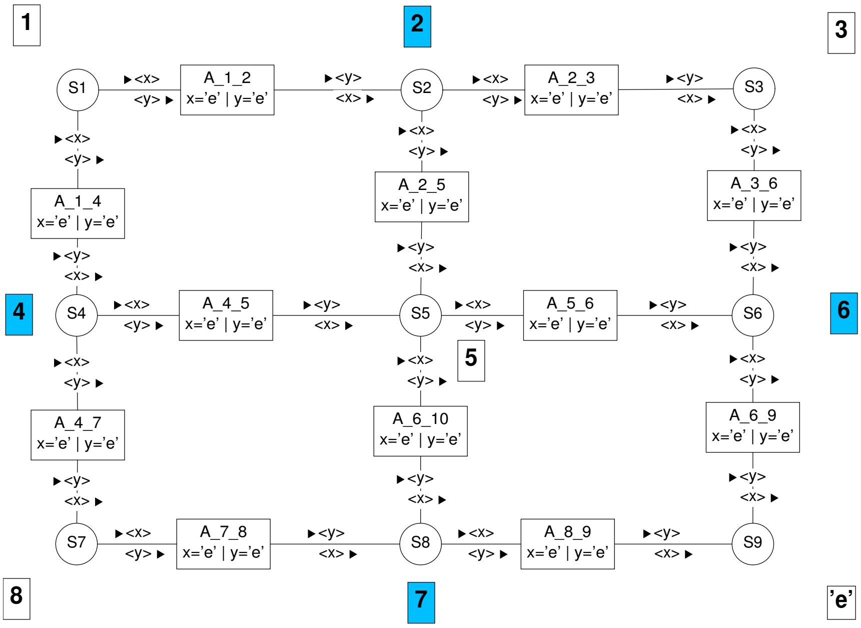


Figure 13: An APA with 12 elementary automata and 9 state components

Alternative behaviour is represented here by the asyncronicity of the possible transitions of the elementary automata that are neighbours to the empty square. For example in the initial position the automata labelled *A*_6_9 as well as *A*_8_9 can act. Both alternatives are evaluated by the tool. This situation can be inspected by starting a simulation and visualising the alternatives by drawing the node environment of the first node generated. Then on this drawn node simulation can be continued by selecting some other node in the direction to be inspected and compute and draw the environment of that node.

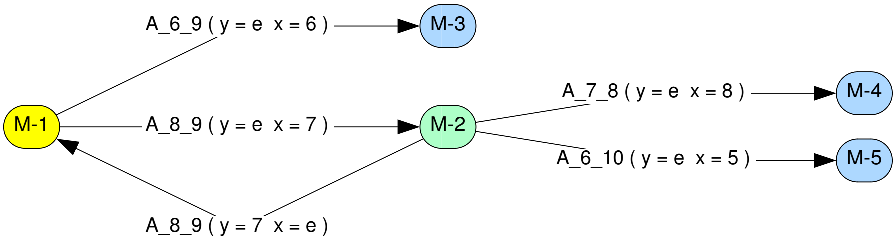


Figure 14: Simulation of 8-puzzle example

Note that there is a built-in check for equal states, in figure 14 the state following $M$-2 when shifting the square with content 7 back to the original position is identified with $M$-1.

One way to find out if the 8-puzzle has the property asked for in figure 11 is, to run a complete analysis [3] of the example and then inspect the generated reachability graph by search queries.

To find out if the state with changed positions of the *8* and *7* is reachable it is sufficient to evaluate the following query that describes the searched state:

```
(S1:<1>) & (S2:<2>) & (S3:<3>) &
(S4:<4>) & (S5:<5>) & (S6:<6>) &
(S7:<7>) & (S8:<8>);
```

This query will find no states matching, that is you can never reach such a state by using the given operations. q.e.d.

The example in figure 15 shows how the 8-puzzle problem can be modelled with the descriptional complexity shifted from the graphical structure - a complex graph of elementary automata - to a simple structure using only one elementary automaton but complex data structures and preamble functions. It furthermore illustrates how the choice-operator can be used instead of multiple asynchronous elementary automata to model alternative behaviour.

[3] The complete analysis of this example takes about 2.5 hours on a P700 using the Lisp-Version with compiler included. The reachability graph has 181440 different states. 483840 transitions are computed.

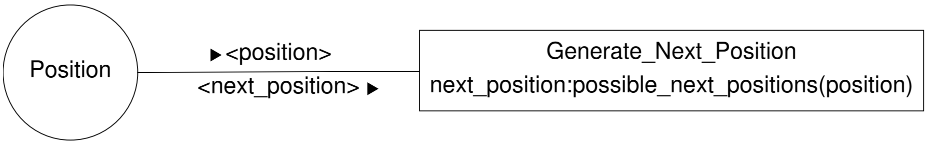


Figure 15: A different model of the 8-puzzle

Here one elementary automaton is used to compute the possible follow-up positions starting from the initial state. The whole puzzle is represented in a data structure using a 9-tuple named *Position*, each tuple element representing one square. A complex preamble function *possible_next_positions* is used to compute a sequence of the possible follow-up positions from a given position.

Alternative behaviour is represented here by the choice operator ":", a special syntactical form in the inscription of the elementary automaton. In this case *next_position* is set to one element chosen from the sequence generated by *possible_next_positions*(*position*). The computation of the reachability graph generates all possible choices at this point.

Note that this construct is often used in the modelling of agents in the protocol specification of e-services in the following sections. The agents have some internal state and from that all possible follow-up states are computed. The usage of the choice operator makes sure that all of them are explored.

# P4

# ABSTRACTION BASED VERIFICATION OF A PARAMETERISED POLICY CONTROLLED SYSTEM

| | |
|---|---|
| **Title** | Abstraction Based Verification of a Parameterised Policy Controlled System |
| **Authors** | Peter Ochsenschläger and Roland Rieke |
| **Publication** | In Vladimir Gorodetsky, Igor Kotenko, and Victor A. Skormin, editors, *Computer Network Security – Fourth International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2007 St. Petersburg, Russia, September 2007, Proceedings*, volume 1 of *Communications in Computer and Information Science*, pages 228–241, 2007. |
| **ISBN/ISSN** | ISBN 978-3-540-73985-2 |
| **DOI** | `http://dx.doi.org/10.1007/978-3-540-73986-9_19` |
| **Status** | Published |
| **Publisher** | Springer |
| **Publication Type** | Conference Proceedings (CCIS, Vol. 1) |
| **Copyright** | 2007, Springer |
| **Contribution of Roland Rieke** | Co-Author ranking equally, editor, and presenter at the MMM-ACNS conference 2007 in St. Petersburg.<br>Specific contributions are: (1) the analysed collaboration scenario; (2) modelling and analysis of the scenario in the SHVT. |

Table 9: Fact Sheet Publication *P4*

Publication P4 [Ochsenschläger & Rieke, 2007] addresses the following research question:

RQ2 *How can finite state verification techniques be extended to prove properties independently of concrete parameters?*

This paper extends the tool supported verification techniques presented in P1 and P2 by an approach to verify entire families of critical systems, independent of the exact number of replicated components. This is demonstrated by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario. Verification results for configurations with fixed numbers of components are used to choose an appropriate property preserving abstraction that provides the basis for an inductive proof that generalises the results for a family of systems with arbitrary settings of parameters. The inductive proof uses the construction of the behaviour of the parameterised system to show that it results in identical abstract system behaviour for any given parameter configuration. This allows the verification of parameterised systems by constructing abstract systems that can be model checked.

# Abstraction Based Verification of a Parameterised Policy Controlled System

Peter Ochsenschläger and Roland Rieke [*]

Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany
{ochsenschlaeger,rieke}@sit.fraunhofer.de

**Abstract.** Safety critical and business critical systems are usually controlled by policies with the objective to guarantee a variety of safety, liveness and security properties. Traditional model checking techniques allow a verification of the required behaviour only for systems with very few components. To be able to verify entire families of systems, independent of the exact number of replicated components, we developed an abstraction based approach to extend our current tool supported verification techniques to such families of systems that are usually parameterised by a number of replicated identical components. We demonstrate our technique by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario. Verification results for configurations with fixed numbers of components are used to choose an appropriate property preserving abstraction that provides the basis for an inductive proof that generalises the results for a family of systems with arbitrary settings of parameters.

**Key words:** Formal analysis of security and liveness properties, security modelling and simulation, security policies, parameterised models.

## 1 Introduction

In a typical policy controlled system, a set of policy rules, posing restrictions on the system's behaviour, is used to enforce the required security objectives, such as confidentiality, integrity and availability. For safety critical systems as well as for business critical systems or parts thereof, assuring the correctness - conformance to the intended purpose - is imperative. These systems must guarantee a variety of safety, liveness and security properties.

*The problem approached.* Traditional model checking techniques can be used to analyse such systems and to understand and verify how they behave subject to different policy constraints. However, because of well known state explosion problems, the usage of these techniques is limited to systems with very few components. In this paper we propose an extension of these techniques to a

[*] Part of the work presented in this paper was developed within the project SicAri being funded by the German Ministry of Education and Research.

particularly interesting class of systems called *parameterised systems*. A parameterised system describes a family of systems that are finite-state in nature but scalable. A formal specification of a parameterised system thus covers a family of systems, each member of which has a different number of replicated components. Instances of the family can be obtained by fixing the parameters. Extensions of model checking techniques are required that support verification of properties that are valid independently of given concrete parameters.

*Contributions.* To be able to verify entire families of critical systems, independent of the exact number of replicated components, we developed an *abstraction based approach* to extend our current tool supported verification techniques to such parameterised systems. Abstraction is a fundamental and widely-used verification technique. It can be used to reduce the verification of a property over a concrete system, to checking a related property over a simpler abstract system [1]. In this paper however we need an inductive proof on the construction of the behaviour of the parameterised system to show that it results in identical abstract system behaviour for any given parameter configuration. This allows the verification of parameterised systems by constructing abstract systems that can be model checked.

In the case of our abstraction based approach, the key problem is the choice of an appropriate abstraction that, (1) is *property preserving*, (2) results in identical abstract system behaviour for any given parameter configuration, and, (3) is sufficiently precise to express the required properties at the chosen abstraction level. To solve this problem, we

- compute the system behaviour and verify the required properties for some configurations with fixed numbers of components;
- we then use the results to choose an appropriate property preserving abstraction that results in identical abstract system behaviour for any given parameter configuration;
- based on this abstraction, we provide an inductive proof (by hand) that generalises the results for a family of systems with arbitrary settings of parameters.

In this paper we demonstrate our technique by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario.

The subsequent paper is structured as follows. In Sect. 2 we review some related work. Section 3 introduces a collaboration scenario that we will use throughout this paper to illustrate the usage of the proposed method for analysis of parameterised models. Section 4 describes the formal modelling technique, the abstraction based verification concept and the verification tool while Sect. 5 presents an exemplary verification of the collaboration scenario. Finally, the paper ends with conclusions and an outlook in Sect. 6.

## 2 Related Work

*Analysis of security policies.* The research in the field of security policies has gained increasing attention in the past few years. Many research papers appeared that investigated security policies on its own and abstracted from the systems needed to enforce these policies. These activities concentrated on the examination of specific properties of policies like consistency, freedom of conflicts, information flow implications and effects to system safety. This allows shifting the attention from specifics of computer system towards the analysis of properties that are inherent to the policy itself.

In the information flow analysis approach presented in [2] for the *SELinux* system, a labelled transition system (LTS) is generated from the policy specifications that models the information flow policy. Temporal logic formulas are used to specify the security goals. The *NuSMV* (`http://nusmv.irst.itc.it/`) model-checker verifies the security goals on this LTS.

A method to enforce rigorous automated network security management using a network access control policy is presented in [3]. This method is illustrated using examples based on enforcement strategy by distributed packet filtering and confidentiality/authenticity goals enforced by IPsec mechanisms.

In [4] a model-based approach focussing on the validation of network security policies and the interplay of threats and vulnerabilities and system's behaviour is proposed. This approach is based on Asynchronous Product Automata (APA) [5]. APA are also used as a basis of the work presented in this paper.

*Verification approaches for parameterised systems.* An extension to the *Mur$\varphi$* verifier to verify systems with replicated identical components through a new data type called RepetitiveID (with restricted usage) is presented in [6]. The verification is performed by explicit state enumeration in an abstract state space where states do not record the exact numbers of components. Mur$\varphi$ automatically checks the soundness of this abstraction and translates the system description to an abstract state graph for a system of a fixed size. During the verification of this system, Mur$\varphi$ uses a run-time check to determine if the result can be generalised for a family of systems. The soundness of the abstraction algorithm is guaranteed by the restrictions on the use of repetitiveIDs. These restrictions allow Mur$\varphi$ to decide which components are abstractable using the repetition constructors, enforce symmetry in the system, which enables the automatic construction of abstract states, and, enforce the repetitive property in the system, which enables the automatic construction of the abstract successors. A typical application area of this tool are cache coherence protocols. Many cache coherence protocols satisfy the above restrictions.

The aim of [7] is an abstraction method through symmetry which works also when using variables holding references to other processes which is not possible in Mur$\varphi$. An implementation of this approach for the *SPIN* model-checker (`http://spinroot.com/`) is described.

In [8] a methodology for constructing abstractions and refining them by analysing counter-examples is presented. The method combines abstraction, model-checking and deductive verification and in particular, allows to use the set of reachable states of the abstract system in a deductive proof even when the abstract model does not satisfy the specification and when it simulates the concrete system with respect to a weaker simulation notion than Milner's. The tool *InVeSt* supports this approach and makes use of *PVS* (`http://pvs.csl.sri.com/`) and *SMV* (`http://www.cs.cmu.edu/ modelcheck/smv.html`). This approach however does not consider liveness properties.

In [9] a technique for automatic verification of parameterised systems based on process algebra *CCS* [10] and the logic modal *mu-calculus* [11] is presented. This technique views processes as property transformers and is based on computing the limit of a sequence of mu-calculus formula generated by these transformers.

The above-mentioned approaches demonstrate, that finite state methods combined with deductive methods can be applied to analyse parameterised systems. The approaches differ in varying amounts of user intervention and their range of application. A survey of a number of approaches to combine model checking and theorem proving methods is given in [12].

Characteristic of our approach is the flexibility of abstractions defined by language homomorphisms and the consideration of liveness properties.

## 3 Collaboration Scenario

There are manifold uses and aspects of the terms *policy* in general and *security policy* specifically. In the context of this paper we use the concepts of the eXtensible Access Control Markup Language (XACML [13]) to express a security policy, but for readability we use a much simpler syntax.

We consider three *roles* (classes of collaboration partners with a uniform security policy) in this scenario namely *trustworthy clients* (TC), *observers* (OB) and a *manager* (M) representing the collaboration infrastructure. There is only one role player for the manager but an unspecified number of role players for the two types of clients. The set of *subjects* is defined by $subject = \{trustworthy\ client, observer, manager\}$. For our collaboration scenario we now assume that a group of trustworthy clients hold a session. The session can be in state *public* (*pub*) or *confidential* (*conf*). The set of possible session states is thus defined by $s\_state = \{pub, conf\}$. The initial session state is *pub*. We furthermore assume that the set of possible *actions* is defined by $action = \{join, leave, close, open\}$ and that the following policy rules govern the session.

$rule_1$ When the session is in state *pub*, then observers are permitted to *join*.
$rule_2$ Observers are permitted to *leave* at any time.
$rule_3$ When no observers participate in the session, then the manager can *close* the session (change state to *conf*).
$rule_4$ The manager can *open* the session (change state to *pub*) at any time.

To be able to decide whether observers are currently participating in a session, we furthermore use a counter $o\_count \in \mathbb{N}_0$ for the current count of observers in the session. The initial value of *o_count* is 0. We don't consider any actions of the trustworthy clients in the model because we consider this irrelevant for the security goals.

In XACML a policy is given by a set of rules and a rule-combining algorithm. Each rule is composed of a condition, an effect, and a target. The conditions (predicates on attributes of subject, resource, action) associated with a policy rule specify when the policy rule is applicable. If the condition returns *False*, the rule returns *NotApplicable*. If the condition returns *True*, the value of the effect element (*Permit* or *Deny*) is returned.

For better readability we use an abbreviated syntax in this paper and define the rules from our example now by

$$\mathit{rule}_x : \mathit{subject} \times \mathit{s\_state} \times \mathit{action} \times \mathit{o\_count} \rightarrow \{\mathit{permit}, \mathit{deny}, \mathit{not\_applicable}\}.$$

$$\mathit{rule}_1(s,a,z,c) = \begin{cases} \mathit{permit} \mid & s = \mathit{observer} \ \wedge \ a = \mathit{join} \ \wedge \ z = \mathit{pub} \\ \mathit{not\_applicable} \mid & \mathit{else} \end{cases}$$

$$\mathit{rule}_2(s,a,z,c) = \begin{cases} \mathit{permit} \mid & s = \mathit{observer} \ \wedge \ a = \mathit{leave} \\ \mathit{not\_applicable} \mid & \mathit{else} \end{cases}$$

$$\mathit{rule}_3(s,a,z,c) = \begin{cases} \mathit{permit} \mid & s = \mathit{manager} \ \wedge \ a = \mathit{close} \ \wedge \ c = 0 \\ \mathit{not\_applicable} \mid & \mathit{else} \end{cases}$$

$$\mathit{rule}_4(s,a,z,c) = \begin{cases} \mathit{permit} \mid & s = \mathit{manager} \ \wedge \ a = \mathit{open} \ \wedge \ z = \mathit{conf} \\ \mathit{not\_applicable} \mid & \mathit{else} \end{cases}$$

The rule-combining algorithm we use to derive the policy result from the given rules is the permit-overrides algorithm, if a single permit result is encountered, then the combined result is permit. So we define the policy for our example now by

$$\mathit{policy} : \mathit{subject} \times \mathit{action} \times \mathit{s\_state} \times \mathit{o\_count} \rightarrow \{\mathit{permit}, \mathit{deny}\}.$$

$$\mathit{policy}(s,a,z,c) = \begin{cases} \mathit{permit} \mid & \mathit{rule}_1(s,a,z,c) = \mathit{permit} \vee \\ & \mathit{rule}_2(s,a,z,c) = \mathit{permit} \vee \\ & \mathit{rule}_3(s,a,z,c) = \mathit{permit} \vee \\ & \mathit{rule}_4(s,a,z,c) = \mathit{permit} \\ \mathit{deny} \mid & \mathit{else} \end{cases}$$

Generally, security policies have to guarantee certain security properties of a system and moreover they must not prevent the system from working.

In our example we define the following security properties:

- the collaboration is in state $conf$ only if no observer is present (security), and
- always eventually state changes between *pub* and $conf$ are possible (liveness).

These properties are formally verified in Sect. 5.

## 4 Verification of System Properties

Our operational finite state model of the behaviour of the given collaboration scenario is based on *Asynchronous Product Automata (APA)*, a flexible operational specification concept for cooperating systems [5]. An APA consists of a family of so called *elementary automata* communicating by common components of their state (shared memory).

### 4.1 Formal Modelling Technique

We now introduce the formal modelling techniques used, and illustrate the usage by our collaboration example.

**Definition 1.** *An* Asynchronous Product Automaton *consists of*

- *a family of* state sets $Z_s, s \in \mathbb{S}$,
- *a family of* elementary automata $(\Phi_e, \Delta_e), e \in \mathbb{E}$ *and*
- *a* neighbourhood relation $N: \mathbb{E} \to \mathcal{P}(\mathbb{S})$

*$\mathbb{S}$ and $\mathbb{E}$ are index sets with the names of state components and of elementary automata and $\mathcal{P}(\mathbb{S})$ is the power set of $\mathbb{S}$.*

*For each elementary automaton $(\Phi_e, \Delta_e)$ with* Alphabet $\Phi_e$, *its* state transition relation *is $\Delta_e \subseteq \bigtimes_{s \in N(e)}(Z_s) \times \Phi_e \times \bigtimes_{s \in N(e)}(Z_s)$. For each element of $\Phi_e$ the state transition relation $\Delta_e$ defines state transitions that change only the state components in $N(e)$. An APA's (global)* states *are elements of $\bigtimes_{s \in \mathbb{S}}(Z_s)$. To avoid pathological cases it is generally assumed that $\mathbb{S} = \bigcup_{e \in \mathbb{E}}(N(e))$ and $N(e) \neq \emptyset$ for all $e \in \mathbb{E}$. Each APA has one* initial state *$q_0 = (q_{0s})_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$. In total, an APA $\mathbb{A}$ is defined by*

$$\mathbb{A} = ((Z_s)_{s \in \mathbb{S}}, (\Phi_e, \Delta_e)_{e \in \mathbb{E}}, N, s_0)$$

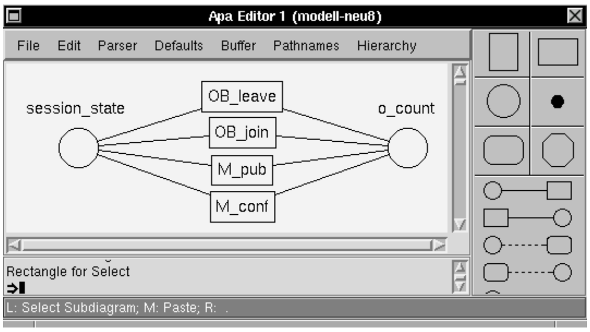
**Finite state model of the collaboration scenario.** The collaboration model described in Sect. 3 is specified for the proposed analysis method using the following *APA state components*:

$\mathbb{S} = \{s\_state, o\_count\}$ with $Z_{s\_state} = \{pub, conf\}$ and $Z_{o\_count} = \mathbb{N}_0$,
$q_0 = (q_{0s\_state}, q_{0o\_count}) = (pub, 0)$.

The set of *elementary automata* $\mathbb{E} = \{OB\_join, OB\_leave, M\_conf, M\_pub\}$ represents the possible actions that the subjects (manager and observers) can take. These specifications are represented in the data structures and initial configuration of the state components in the APA model. The lines in Fig. 1 between state components and elementary automata represent the neighbourhood relation.

From Fig. 1 we conclude that $N(e) = \mathbb{S}$ for each $e \in \mathbb{E}$.
For each $e \in \mathbb{E}$ we choose $\Phi_e = \{\#\}$. Therefore we can omit the middle component of the state transition relation $\Delta_e$.

Using the abbreviation $state = \{pub, conf\}$, it holds $\Delta_e \subset (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0)$ for each $e \in \mathbb{E}$.



Rectangle for Select

L: Select Subdiagram; M: Paste; R: .

*OB_join* - observer *join* collaboration,
*OB_leave* - observer *leave* collaboration,
*M_pub* - manager changes state to *pub*,
*M_conf* - manager changes state to *conf*

**Fig. 1.** Collaboration model

In detail:

$$\begin{aligned}
\Delta_{OB_{leave}} &= \{((x,y),(x,y-1)) \in (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0) \mid \\
&\quad y > 0 \wedge policy(observer, leave, x, y) = permit\} \\
\Delta_{OB_{join}} &= \{((x,y),(x,y+1)) \in (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0) \mid \\
&\quad y > maxOB \wedge policy(observer, join, x, y) = permit\} \\
\Delta_{M_{conf}} &= \{((x,y),(conf,y)) \in (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0) \mid \\
&\quad policy(manager, close, x, y) = permit\} \\
\Delta_{M_{pub}} &= \{((x,y),(pub,y)) \in (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0) \mid \\
&\quad policy(manager, open, x, y) = permit\}
\end{aligned}$$

Note that this APA is parameterised by $maxOB \in \mathbb{N}_0$.

**Definition 2.** *An elementary automaton* $(\Phi_e, \Delta_e)$ *is* activated *in a state* $q = (q_s)_{s\in\mathbb{S}} \in \times_{s\in\mathbb{S}}(Z_s)$ *as to an* interpretation $i \in \Phi_e$, *if there are* $(p_s)_{s\in N(e)} \in \times_{s\in N(e)}(Z_s)$ *with* $((q_s)_{s\in N(e)}, i, (p_s)_{s\in N(e)}) \in \Delta_e$. *An activated elementary automaton* $(\Phi_e, \Delta_e)$ *can execute a state transition and produce a* successor state $p = (p_s)_{s\in\mathbb{S}} \in \times_{s\in\mathbb{S}}(Z_s)$, *if* $q_r = p_r$ *for* $r \in \mathbb{S} \setminus N(e)$ *and* $(q_s)_{s\in N(e)}, i, (p_s)_{s\in N(e)} \in \Delta_e$. *The corresponding state transition is* $(q, (e, i), p)$.

For example $((conf, 0), (M\_pub, \#), (pub, 0))$ is a state transition of our example. As mentioned above, we omit # in the sequel.

**Definition 3.** *The behaviour of an APA is represented by all possible coherent sequences of state transitions starting with initial state* $q_0$. *The sequence* $(q_0, (e_1, i_1), q_1)$ $(q_1, (e_2, i_2), q_2)$ $(q_2, (e_3, i_3), q_3) \ldots$ $(q_{n-1}, (e_n, i_n), q_n)$ *with* $i_k \in \Phi_{e_k}$ *represents one possible sequence of actions of an APA.* $q_n$ *is called the* goal *of this action sequence.*

*State transitions* $(p, (e, i), q)$ *may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA:* $(p, (e, i), q)$ *is the edge leading from* $p$ *to* $q$ *and labelled by* $(e, i)$. *The subgraph reachable from the node* $q_0$ *is called the* reachability graph *of an APA.*

*Let* $\mathbb{Q}$ *denote the set of all states* $q \in \times_{s\in\mathbb{S}}(Z_s)$ *that are reachable from the initial state* $q_0$ *and let* $\Psi$ *denote the set of all state transitions with the first component in* $\mathbb{Q}$.

*The set $L \subset \Psi^*$ of all action sequences with initial state $q_0$ including the empty sequence $\epsilon$ denotes the* action language *of the corresponding APA. The action language is prefix closed. By definition $q_0$ is the goal of $\epsilon$.*

The *reachability graph* of the example depends on the parameter $maxOB \in \mathbb{N}_0$; its set of nodes is given by $\mathbb{Q}_{maxOB}$ and its set of edges is $\Psi_{maxOB}$.

It is $\mathbb{Q}_{maxOB} \subset \{pub, conf\} \times \mathbb{N}_0$ and $\Psi_{maxOB} \subset \mathbb{Q}_{maxOB} \times \mathbb{E} \times \mathbb{Q}_{maxOB}$. The reachability graph for $maxOB = 0$ is shown in Fig. 2. The reachability graph for $maxOB = 1$ is depicted by the solid lines in Fig. 3, whereas the dashed lines in the same figure show the reachability graph for $maxOB = 2$.

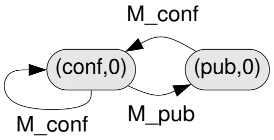


**Fig. 2.** Reachability graph for $maxOB = 0$

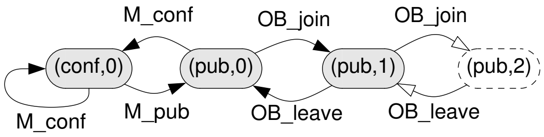


**Fig. 3.** Reachability graphs for $maxOB = 1$ (solid lines) and $maxOB = 2$ (dashed)

For example $((pub, 0), M\_conf, (conf, 0))((conf, 0), M\_pub, (pub, 0))$ is an element of the action language.

### 4.2 Abstraction Based Verification Concept

Now behaviour abstraction of an APA can be formalised by language homomorphisms, more precisely by alphabetic language homomorphisms $h : \Sigma^* \to \Sigma'^*$.

By these homomorphisms certain transitions are ignored and others are renamed, which may have the effect, that different transitions are identified with one another. A mapping $h : \Sigma^* \to \Sigma'^*$ is called a language homomorphism if $h(\epsilon) = \epsilon$ and $h(yz) = h(y)h(z)$ for each $y, z \in \Sigma^*$. It is called alphabetic, if $h(\Sigma) \subset \Sigma' \cup \{\epsilon\}$.

It is now the question, whether, by investigating an abstract behaviour, we may verify the correctness of the underlying concrete behaviour. Generally under abstraction the problem occurs, that an incorrect subbehaviour can be hidden by a correct one. We will answer this question positively, requiring a restriction to the permitted abstraction techniques [1].

As it is well known, system properties are divided into two types: safety (what happens is not wrong) and liveness properties (eventually something desired happens, e.g. availability) [14].

On account of liveness aspects system properties are formalised by $\omega$-languages (sets of infinite long words). So to investigate satisfaction of properties "infinite system behaviour" has to be considered. This is formalised by so called Eilenberg limits of action languages (more precisely: by Eilenberg limits of modified action languages where maximal words are continued by an unbounded repetition of a dummy action) [15].

The usual concept of linear satisfaction of properties (each infinite run of the system satisfies the property) is not suitable in this context because no fairness constraints are considered. We put a very abstract notion of fairness into the satisfaction relation for properties, which considers that independent of a finitely long computation of a system certain desired events may occur eventually. To formalise such "possibility properties", which are of interest when considering what we call cooperating systems, the notion of approximate satisfaction of properties is defined in [15].

**Definition 4.** *A system* approximately satisfies *a property if and only if each finite behaviour can be continued to an infinite behaviour, which satisfies the property.*

For safety properties linear satisfaction and approximate satisfaction are equivalent [15]. To deduce approximately satisfied properties of a specification from properties of its abstract behaviour an additional property of abstractions called simplicity of homomorphisms on an action language [16] is required. Simplicity of homomorphisms is a very technical condition concerning the possible continuations of finite behaviours.

For regular languages simplicity is decidable. In [16] a sufficient condition based on the strongly connected components of corresponding automata is given, which easily can be checked. Especially: If the automaton or reachability graph is strongly connected, then each homomorphism is simple.

The following theorem [15] shows that approximate satisfaction of properties and simplicity of homomorphisms exactly fit together for verifying cooperating systems.

**Theorem 1.** *Simple homomorphisms define exactly the class of such abstractions, for which holds that each property is approximately satisfied by the abstract behaviour if and only if the "corresponding" property is approximately satisfied by the concrete behaviour of the system.*

Formally, the "corresponding" property is expressed by the inverse image of the abstract property with respect to the homomorphism.

In the example of this paper the desired security properties are safety and liveness properties. Generally there are more complex security properties. In [17] and [18] it has been shown how authenticity, provability and confidentiality are also treated in terms of prefix closed languages and property preserving language homomorphisms.

### 4.3 Verification Tool

The *Simple Homomorphism (SH) verification tool* [5] is used to analyse the collaboration model for different concrete values of $maxOB$. It has been developed at the *Fraunhofer-Institute for Secure Information Technology.* The SH verification tool provides components for the complete cycle from formal specification to exhaustive validation as well as visualisation and inspection of computed reachability graphs and minimal automata. The applied specification method based on *Asynchronous Product Automata (APA)* is supported by this tool. The tool manages the components of the model, allows to select alternative parts of the specification and automatically *glues* together the selected components to generate a combined model of the APA specification. After an initial state is selected, the reachability graph is automatically computed by the SH verification tool.

The tool provides an editor to define homomorphisms on action languages, it computes corresponding minimal automata [19] for the homomorphic images and checks simplicity of the homomorphisms.

*Model checking.* If it is required to inspect some or all paths of the graph to check for the violation of a security property, as it is usually the case for liveness properties, then the tool's temporal logic component can be used. Temporal logic formulae can also be checked on the abstract behaviour (under a simple homomorphism). The method for checking approximate satisfaction of properties fits exactly to the built-in simple homomorphism check [5].

The SH verification tool successfully has been applied in several security projects such as Valikrypt (`http://www.bsi.bund.de/fachthem/valikrypt/`) and CASENET[1].

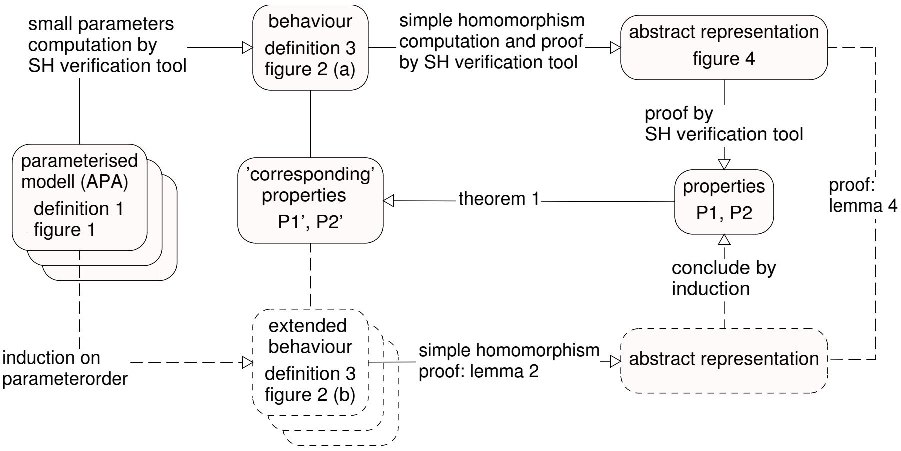
## 5 Verification of the Collaboration Scenario

An outline of our verification concept for parameterised models, exemplary realised for the collaboration scenario, is given in Fig. 4.

The abstraction based verification concept introduced in Sect. 4.2 and the tool support described in Sect. 4.3 cover the part marked by solid lines in Fig. 4 whereas we now prove the components marked by dashed lines.

Using the graphs of Fig. 2 and Fig. 3 as induction base we will now prove Lemma 1 below by induction on $maxOB$. We use the abbreviations
$T_{mjoin}$ for $((pub, maxOB), OB\_join, (pub, maxOB + 1))$ and
$T_{mleave}$ for $((pub, maxOB + 1), OB\_leave, (pub, maxOB))$.

[1] The EU project CASENET (`http://www.casenet-eu.org/`) has provided a tool-supported framework for the systematic specification, design and analysis of e-commerce and e-government transactions to produce protocols with proven security properties, and to assist in code generation for these protocols.



small parameters computation by SH verification tool
behaviour definition 3 figure 2 (a)
simple homomorphism computation and proof by SH verification tool
abstract representation figure 4
proof by SH verification tool
parameterised modell (APA) definition 1 figure 1
'corresponding' properties P1', P2'
theorem 1
properties P1, P2
proof: lemma 4
conclude by induction
extended behaviour definition 3 figure 2 (b)
induction on parameterorder
simple homomorphism proof: lemma 2
abstract representation

**Fig. 4.** Verification concept for parameterised APA

**Lemma 1.** *(a)* $\mathbb{Q}_{maxOB} = \{(pub, i)|0 \leq i \leq maxOB\} \cup \{(conf, 0)\}$
*(b)* $\Psi_{maxOB+1} = \Psi_{maxOB} \dot{\cup} \{T_{mjoin}, T_{mleave}\}$

*Proof.* Figure 3 shows the reachability graph with $maxOB = 1$. Together with Fig. 2, Fig. 3 proves the induction base.
*Induction step.*
By inspection of the 4 elementary automata we get: $\Psi_{maxOB} \subset \Psi_{maxOB+1}$. Starting from the nodes in $\mathbb{Q}_{maxOB}$ from $maxOB + 1$ only the additional transitions $T_{mjoin}$ and $T_{mleave}$ are possible.

□

It follows by induction:

**Lemma 2.** *For each* $maxOB \in \mathbb{N}_0$ *the corresponding reachability graph is finite and strongly connected.*

Let $L_{maxOB} \subset \Psi^*_{maxOB}$ denote the *action language*, then using Lemma 1(b) we can derive

**Lemma 3.** *(a)* $L_{maxOB} \subset L_{maxOB+1}$ *and*
*(b) for each* $u \in L_{maxOB+1}$*:* $h(u) \in L_{maxOB}$ *with the homomorphism*
$h : \Psi^*_{maxOB+1} \to \Psi^*_{maxOB}$
*with* $h(T_{mjoin}) = \varepsilon = h(T_{mleave})$ *and* $h(x) = x$ *for* $x \in \Psi_{maxOB}$
*(c) The goal of* $u$ *is identical to the goal of* $h(u)$ *or*
*the goal of* $u$ *is* $(pub, maxOB + 1)$ *and the goal of* $h(u)$ *is* $(pub, maxOB)$*.*

*Proof of Lemma 3 (b) by induction on the length of u.*
*Induction base.* Lemma 3(b) is true for $u = \varepsilon$. Note that by definition the goal of the empty transition sequence is equal to the initial state of the APA.

*Induction step.* Consider $ua \in L_{maxOB+1}$ with $a \in \Psi_{maxOB+1}$. From induction hypothesis there are 2 different cases:

*Case 1.* The goal of $u$ is equal to the goal of $h(u)$ and therefore an element of $\mathbb{Q}_{maxOB}$.
Therefore $a \in \Psi_{maxOB} \cup \{T_{mjoin}\}$.
For $a \in \Psi_{maxOB}$ holds: $h(ua) = h(u)h(a) = h(u)a$
Therefore from induction hypothesis $h(ua) \in L_{maxOB}$ and goals of $ua$ and $h(ua)$ are equal.
For $a = T_{mjoin}$ the goal of $u$ and therefore also the goal of $h(u)$ is $(pub, maxOB)$.
Now it holds that $h(ua) = h(u)h(a) = h(u)$.
From induction hypothesis we get that $h(ua) \in L_{maxOB}$ and goal of $ua$ is $(pub, maxOB+1)$ and goal of $h(ua)$ is $(pub, maxOB)$.

*Case 2.* The goal of $u$ is $(pub, maxOB+1)$ and goal of $h(u)$ is $(pub, maxOB)$.
Then: $a = T_{mleave}$
And so:
$h(ua) = h(u)h(a) = h(u) \in L_{maxOB}$ and $ua$ and also $h(ua)$ have the same goal, namely $(pub, maxOB)$. □

Now from Lemma 3 (a) we get $L_{maxOB} = h(L_{maxOB}) \subset h(L_{maxOB+1})$
and from 3 (b) we get $h(L_{maxOB+1}) \subset L_{maxOB}$
together $L_{maxOB} = h(L_{maxOB+1})$.
For each homomorphism $f : \Psi^*_{maxOB+1} \to \Sigma'^*$ with $f(T_{mjoin}) = \varepsilon = f(T_{mleave})$ it holds that: $f(L_{maxOB+1}) = f(h(L_{maxOB+1})) = f(L_{maxOB})$
and so:

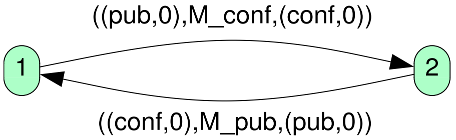
**Lemma 4.** *With the assumptions above holds:* $f(L_{maxOB+1}) = f(L_{maxOB})$

### 5.1 Proving Security and Liveness of the Collaboration Example

To consider our example's correctness we have to observe the state changes between *pub* and *conf*. So we define an appropriate homomorphism
$c : \Psi^*_{maxOB} \to \Psi^*_{maxOB}$ by
$c(((x_1, x_2), e, (y_1, y_2))) = ((x_1, x_2), e, (y_1, y_2))$ if $x_1 \neq y_1$ , and
$c(((x_1, x_2), e, (y_1, y_2))) = \varepsilon$ if $x_1 = y_1$ .

This homomorphism $c$ fulfils the condition of Lemma 4 and therefore we get $c(L_{maxOB+1}) = c(L_{maxOB})$.

This implies $c(L_{maxOB}) = c(L_0)$ for each $maxOB \in \mathbb{N}_0$.



Initial state is 1.
All states are final states.

**Fig. 5.** Minimal automaton of $c(L_0)$

It is easy to see, that the automaton of Fig. 5 is the minimal automaton of $c(L_0)$.

This automaton shows that the collaboration is in state $conf$ only if no observer is present (P1). Moreover always state changes between *pub* and $conf$ are possible (P2).

By Lemma 2 $c$ is simple on each $L_{maxOB}$ and therefore (Theorem 1) corresponding properties P1' and P2' hold for each concrete behaviour $L_{maxOB}$. In content P1' is the same as P1. P2' is the property that always eventually state changes between *pub* and $conf$ are possible. The difference between P2 and P2' is caused by actions of the concrete behaviour which are mapped to $\varepsilon$ by the homomorphism $c$. P1' and P2' are the desired properties of the collaboration as formulated in Sect. 3.

## 6 Conclusions and Future Work

Based on property preserving abstractions (simple homomorphisms) we combined our tool supported finite state methods with induction proofs to verify security and liveness properties of a parameterised system.

We have shown how abstractions serve as a framework for individual proofs of problem specific security properties. So our results are no contradictions to well known undecidability properties of general security models e.g. Harrison-Ruzzo-Ullman.

This paper focussed on properties which are independent of concrete parameter values. Considering parameterised abstract behaviours we will extend our method to verify parameter dependent properties. The induction proofs in this paper are "handmade". So it would be desirable to support such proofs by a theorem prover. For that purpose our system specifications based on parameterised APA have to be represented in a corresponding theorem prover.

**Acknowledgements.** We would like to thank Carsten Kunz, Carsten Rudolph and Björn Steinemann for cooperation on early versions of this work and many productive discussions on the subject.

## References

1. Ochsenschläger, P., Repp, J., Rieke, R.: Abstraction and composition – a verification method for co-operating systems. Journal of Experimental and Theoretical Artificial Intelligence **12** (2000) 447–459 Copyright: ©2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.
2. Guttman, J.D., Herzog, A.L., Ramsdell, J.D.: Information flow in operating systems: Eager formal methods. IFIP WG 1.7 Workshop on Issues in the Theory of Security (2003)
3. Guttman, J.D., Herzog, A.L.: Rigorous automated network security management. International Journal of Information Security **4**(1-2) (2005) 29–48

4. Rieke, R.: Modelling and Analysing Network Security Policies in a Given Vulnerability Setting. In: Critical Information Infrastructures Security, First International Workshop, CRITIS 2006, Samos Island, Greece. Volume 4347 of LNCS., Springer (2006) 67–78 © Springer.
5. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing, The International Journal of Formal Method **11** (1999) 1–24
6. Ip, C.N., Dill, D.L.: Verifying Systems with Replicated Components in Mur$\varphi$. Formal Methods in System Design **14**(3) (1999) 273–310
7. Derepas, F., Gastin, P.: Model checking systems of replicated processes with spin. In: SPIN '01: Proceedings of the 8th international SPIN workshop on Model checking of software, New York, NY, USA, Springer-Verlag New York, Inc. (2001) 235–251
8. Lakhnech, Y., Bensalem, S., Berezin, S., Owre, S.: Incremental verification by abstraction. In Margaria, T., Yi, W., eds.: TACAS. Volume 2031 of Lecture Notes in Computer Science., Springer (2001) 98–112
9. Basu, S., Ramakrishnan, C.R.: Compositional analysis for verification of parameterized systems. Theor. Comput. Sci. **354**(2) (2006) 211–229
10. Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice Hall (1989)
11. Bradfield, J., Stirling, C.: Modal logics and mu-calculi: an introduction (2001)
12. Uribe, T.E.: Combinations of model checking and theorem proving. In: FroCoS '00: Proceedings of the Third International Workshop on Frontiers of Combining Systems, London, UK, Springer-Verlag (2000) 151–170
13. Moses, T.: eXtensible Access Control Markup Language (XACML), Version 2.0. Technical report, OASIS Standard (2005)
14. Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters **21**(4) (1985) 181–185
15. Nitsche, U., Ochsenschläger, P.: Approximately satisfied properties of systems and simple language homomorphisms. Information Processing Letters **60** (1996) 201–206
16. Ochsenschläger, P.: Verification of cooperating systems by simple homomorphisms using the product net machine. In Desel, J., Oberweis, A., Reisig, W., eds.: Workshop: Algorithmen und Werkzeuge für Petrinetze, Humboldt Universität Berlin (1994) 48–53
17. Gürgens, S., Ochsenschläger, P., Rudolph, C.: On a formal framework for security properties. International Computer Standards & Interface Journal (CSI), Special issue on formal methods, techniques and tools for secure and reliable applications (2004)
18. Gürgens, S., Ochsenschläger, P., Rudolph, C.: Abstractions preserving parameter confidentiality. In: Computer Security – ESORICS 2005. (2005) 418–437 Copyright: ©2005, Springer Verlag.
19. Eilenberg, S.: Automata, Languages and Machines. Volume A. Academic Press, New York (1974)

# P5

# IDENTIFICATION OF SECURITY REQUIREMENTS IN SYSTEMS OF SYSTEMS BY FUNCTIONAL SECURITY ANALYSIS

| | |
|---|---|
| **Title** | Identification of Security Requirements in Systems of Systems by Functional Security Analysis |
| **Authors** | Andreas Fuchs and Roland Rieke |
| **Publication** | In Antonio Casimiro, Rogério de Lemos, and Cristina Gacek, editors, *Architecting Dependable Systems VII*, pages 74–96. |
| **ISBN/ISSN** | ISBN 978-3-642-17244-1 |
| **DOI** | `http://dx.doi.org/10.1007/978-3-642-17245-8_4` |
| **Status** | Published |
| **Publisher** | Springer Berlin Heidelberg |
| **Publication Type** | Book Chapter (LNCS, Vol. 6420) |
| **Copyright** | 2010, Springer |
| **Contribution of Roland Rieke** | Co-Author ranking equally; this is an invited book chapter that is an extended version of [Fuchs & Rieke, 2009] by the same authors (also ranking equally) which the author of this thesis previously presented at the Workshop on Architecting Dependable Systems (WADS 2009) in connection with the 2009 IEEE/IFIP Conference on Dependable Systems and Networks. |

Table 10: Fact Sheet Publication *P5*

Publication P5 [Fuchs & Rieke, 2010] addresses the following research question:

RQ3 *How can security requirements for cooperating systems be elicited systematically?*

This book chapter is an extended version of [Fuchs & Rieke, 2009]. It provides a model-based approach to systematically identify security requirements for cooperating systems. The proposed method comprises the tracing down of functional dependencies over system component boundaries right onto the origin of information as a functional flow graph. Based on this graph, comprehensive sets of formally defined authenticity requirements for the given security and dependability objectives are systematically deduced. The proposed method thereby avoids premature assumptions on the security architecture's structure as well as the means by which it is realised. The most common problem with security requirements is that they tend to be replaced with security-specific architectural constraints that may unnecessarily constrain the choice of the most appropriate security mechanisms [Firesmith, 2003]. Therefore, the proposed approach avoids to break down the overall security requirements to requirements for specific components or communication channels prematurely. So the requirements identified by this approach are independent of decisions not only on concrete security enforcement mechanisms to use, but also on the structure, such as hop-by-hop versus end-to-end security measures.

# Identification of Security Requirements in Systems of Systems by Functional Security Analysis

Andreas Fuchs and Roland Rieke

Fraunhofer Institute for Secure Information Technology (SIT)
Rheinstrasse 75, 64295 Darmstadt, Germany
`{andreas.fuchs,roland.rieke}@sit.fraunhofer.de`

**Abstract.** Cooperating systems typically base decisions on information from their own components as well as on input from other systems. Safety critical decisions based on cooperative reasoning however raise severe concerns to security issues. Here, we address the security requirements elicitation step in the security engineering process for such systems of systems. The method comprises the tracing down of functional dependencies over system component boundaries right onto the origin of information as a functional flow graph. Based on this graph, we systematically deduce comprehensive sets of formally defined authenticity requirements for the given security and dependability objectives. The proposed method thereby avoids premature assumptions on the security architecture's structure as well as the means by which it is realised. Furthermore, a tool-assisted approach that follows the presented methodology is described.

**Key words:** security requirements elicitation, systems of systems security engineering, security analysis for vehicular communication systems

## 1 Introduction

Architecting novel mobile systems of systems (SoS) poses new challenges to getting the dependability and specifically the security requirements right as early as possible in the system design process. Security engineering is one important aspect of dependability [1]. The security engineering process addresses issues such as how to identify and mitigate risks resulting from connectivity and how to integrate security into a target architecture [2]. Security requirements need to be explicit, precise, adequate, non-conflicting with other requirements and complete [13].

A typical application area for mobile SoS are vehicular communication systems in which vehicles and roadside units communicate in ad hoc manner to exchange information such as safety warnings and traffic information. As a cooperative approach, vehicular communication systems can be more effective in avoiding accidents and traffic congestion than current technologies where each vehicle tries to solve these problems individually. However, introducing dependence of possibly safety-critical decisions in a vehicle on information from other systems, such as other vehicles or roadside units, raises severe concerns to security issues. Security is an enabling technology in this emerging field because without security some applications within those SoS would not be possible at all. In some cases security is the main concern of the architecture [22].

The first step in the design of an architecture for a novel system of systems is the requirements engineering process. With respect to security requirements this process typically covers at least the following activities [17, 16, 15]

– the identification of the target of evaluation and the principal security goals and the elicitation of artifacts (e.g. use case and threat scenarios) as well as risk assessment
– the actual security requirements elicitation process
– a requirements categorisation and prioritisation, followed by requirements inspection

In this paper we address the security requirements elicitation step in this process. We present a model-based approach to systematically identify security requirements for system architectures to be designed for cooperative applications in a SoS context. Our contribution comprises the following distinctive features.

*Identification of a Consistent and Complete Set of Authenticity Requirements.* We base our method on the following general assumption about the overall security goal with respect to authenticity requirements:

> *For every safety-critical action in a system of systems, all information that is used in the reasoning process that leads to this action has to be authentic.*

To achieve this, we first derive a functional model of a system by identification of atomic actions and functional dependencies in a use case description. From this model we generate a dependency graph with the safety-critical function under consideration as root and the origins of decision relevant information as leaves. Based on this graph, we deduce a set of authenticity requirements that is comprehensive and defines the maximal set of authenticity requirements from the given functional dependencies.

*Security Mechanism Independence.* The most common problem with security requirements is, that they tend to be replaced with security-specific architectural constraints that may unnecessarily constrain the choice of the most appropriate security mechanisms [4].

In our approach we avoid to break down the overall security requirements to requirements for specific components or communication channels prematurely. So the requirements identified by this approach are independent of decisions not only on concrete security enforcement mechanisms to use, but also on the structure, such as hop-by-hop versus end-to-end security measures.

Throughout this paper we use the following terminology taken from [1]: A *system* is an entity that interacts with other entities, i.e., other systems. These other systems are the *environment* of the given system. A *system boundary* is the common frontier between the system and its environment. Such a system itself is composed of *components*, where each component is yet another system. Furthermore, in [1] the *dependence* of system A on system B represents the extent to which system A's dependability is affected by that of system B. Our work though focuses on purely functional aspects of dependence and omits quantitative reasoning. For the approach proposed, we describe the *function* of such a system by a *functional model* and treat the components as atomic and thus we do not make preliminary assumptions regarding their inner structure. Rather, the adaption to a concrete architecture is considered to be a task within a follow-up refinement and engineering process.

The subsequent paper is structured as follows. Section 2 gives an overview of the related work on security engineering and requirements identification methodologies. In Sect. 3 we introduce a scenario from the automotive domain that will serve as use case throughout the rest of this text. Section 4 introduces the proposed approach to requirements identification, exemplified by application on the given use case. Section 5 presents an tool-assisted methodology that follows this approach utilising the scenario. Finally, the paper ends with conclusions and an outlook in Sect. 6.

## 2 Related Work

The development of new security relevant systems that interact to build new SoS requires the integration of a security engineering process in the earliest stages of the development life-cycle. This is specifically important in the development of systems where security is the enabling technology that makes new applications possible. There are several common approaches that may be taken, depending on the system architect's background.

In order to design a secure of vehicular communication system, an architect with a background in Mobile Adhoc Networks (MANETs) would probably first define the data origin authentication [27] of the transmitted message. In a next step he may reason about the trustworthiness of the transmitting system. An architect with a background in Trusted Computing [7] would first require for the transmitting vehicle to attest for its behaviour [25]. Advanced experts may use the Trusted Platform Module (TPM) techniques of sealing, binding, key restrictions and TPM-CertifyKey to validate the trustworthiness and bind the transmitted data to this key [24]. A distributed software architect may first start to define the trust zones. This would imply that some computational means of composing slippery wheels with temperature and position happen in an untrusted domain. Results may be the timestamped signing of the sensor data and a composition of these data at the receiving vehicle.

This shall only illustrate a few different approaches that might be taken in a security engineering process for new SoS. Very different types of security requirements are the outcome. Some of these leave attack vectors open, such as the manipulation of the sending or receiving vehicle’s internal communication and computation.

Another conclusion that can be derived from these examples is related to premature assumptions about the implementation. Whilst in one case the vehicle is seen as a single computational unit that can be trusted, in another case it has to attest for its behaviour when sending out warnings. The trust zone based analysis of the same use cases however requires for a direct communication link and cryptography between the sensors and the receiving vehicle and the composition of data is moved to the receiver side. A direct result of falsely defined system boundaries typically are security requirements that are formulated against internal subsystems rather than the system at stake itself, To overcome these problems several methods for security requirements elicitation have been proposed.

A comprehensive concept for an overall security requirements engineering process is described in detail in [16]. The authors propose a 9 step approach called SQUARE (Security Quality Engineering Methodology). The elicitation of the security requirements is one important step in the SQUARE process. In [15] several concrete methods to carry out this step are compared. These methods are based on misuse cases (MC), soft systems methodology (SSM), quality function deployment (QFD), controlled requirements expression (CORE), issue-based information systems (IBIS), joint application development (JAD), feature-oriented domain analysis (FODA), critical discourse analysis (CDA) as well as accelerated requirements method (ARM). A comparative rating based on 9 different criteria is also given but none of these criteria measures the completeness of the security requirements elicited by the different methods.

A similar approach based on the integration of Common Criteria (ISO/IEC 15408) called SREP (Security Requirements Engineering Process) is described in [17]. However the concrete techniques that carry out the security requirements elicitation process are given only very broadly. A threat driven method is proposed but is not described in detail.

In [13] anti-goals derived from negated security goals are used to systematically construct threat trees by refinement of these anti-goals. Security requirements are then obtained as countermeasures. This method aims to produce more complete requirements than other methods based on misuse cases. The refinement steps in this method can be performed informally or formally.

In [4] different kinds of security requirements are identified and informal guidelines are listed that have proven useful when eliciting concrete security requirements. The author emphasises that there has to be a clear distinction between security requirements and security mechanisms.

In [9] it is proposed to use Jackson‘s problem diagrams to determine security requirements which are given as constraints on functional requirements. Though this approach presents a methodology to derive security requirements from security goals, it does not explain the actual refinements process, which leaves open, the degree of coverage of requirements, depending only on expert knowledge.

In [10–12] Hatebur et al. describe a security engineering process based on security problem frames and concretised security problem frames. The two kinds of frames constitute patterns for analysing security problems and associated solution approaches. They are arranged in a pattern system with formal preconditions and postconditions for the frames which makes dependencies between them explicit. A method to use this pattern system to analyse a given security problem and find solution approaches is described. The focus of [10] is on anonymity, while [11] focusses on confidential data transmission, and [12] addresses accountability by logging and the steps of the process.

In [14] actor dependency analysis is used to identify attackers and potential threats in order to identify security requirements. The so called $i^*$ approach facilitates the analysis of security requirements within the social context of relevant actors. In [6] a formal framework is presented for modelling and analysis of security and trust requirements at an organisational level. Both of these approaches target organisational relations among agents rather than functional dependence. Those approaches might be utilised complementary to the one presented in this paper, as the output of organisational relations analysis may be an input to our functional security analysis.

Though all of the approaches may lead to a sufficient level of security for the designed architecture, there is no obvious means by which they can be compared regarding the security requirements that they fulfil. The choice of the appropriate abstraction level and system boundaries constitutes a rather big challenge to SoS architecture design, especially with respect to SoS applications like the one presented here.

The method described in Sect. 4 in this paper is based on the work presented in [5], whereas the tool-assisted methodology that builds on this approach presented in Sect. 5 is a new contribution of this work. We are targeting here the identification of a consistent and complete set of authenticity requirements. For an analysis of privacy-related requirements with respect to vehicular communication systems please refer to [26].

## 3 Vehicular Communication Systems Scenario

The derivation of security requirements in general, especially the derivation of authenticity requirements represents an essential building block for system design. With an increase in the severity of safety-relevant systems' failures the demand increases for a systematic approach of requirements derivation with a maximum coverage. Also during the derivation of security requirements, no pre-assumptions should be made about possible implementations. We will further motivate this with respect to the requirements derivation process with an example from the field of vehicle-to-vehicle communications.

### 3.1 Example Use Cases

In order to illustrate our approach we use a scenario taken from the project EVITA (E-Safety Vehicle Intrusion Protected Applications) [23]. The scenario is based on an evaluation of security relevant use cases for vehicular communication systems in which vehicles and roadside units communicate in an ad hoc manner to exchange information such as safety warnings and traffic information. Optionally, local danger warning information can also be provided to in-vehicular safety concepts for further processing.

Our example system consists of vehicles $V_1, \ldots, V_n$. Each $V_i$ has its driver $D_i$ and is equipped with an Electronic Stability Protection (ESP) sensor $ESP_i$ and a Global Positioning System (GPS) sensor $GPS_i$. Within each vehicle's on-board network, the scenario involves a communication unit (CU) $CU_i$ for sending and receiving messages. Furthermore, a connection to a Human Machine Interface (HMI) $HMI_i$ is required for displaying the warning message, e.g. via audio signals or on a display. Furthermore, the example system includes a roadside unit (RSU) that can send cooperative awareness messages *cam*. For simplicity reasons we assume that the same information is provided by all roadside units in the system, so we can abstract from the individual entity. Our vehicle-to-vehicle scenario is based on the following use cases:

**Use case 1** A roadside unit broadcasts a cooperative awareness message.

**Use case 2** A vehicle's ESP sensor recognises that the ground is very slippery when accelerating in combination with a low temperature. In order to warn successive vehicles about a possibly icy road, the vehicle uses its communication unit to send out information about this danger including the GPS position data indicating where the danger was detected.

**Use case 3** A vehicle receives a cooperative awareness message, such as a warning about an icy road at a certain position, from a roadside unit or another vehicle. It compares the information to its own position and heading and signals the driver a warning if the dangerous area lies up front.

**Use case 4** A vehicle receives a cooperative awareness message. It compares the information to its own position and heading and retransmits the warning, given that the position of this occurrence is not too far away.

For local danger warning applications, at least two entities are involved, namely the vehicle receiving a critical warning message and the entity sending such a message. The entity that sends out the message can be another vehicle, a roadside unit or traffic light, or an infrastructure based server. The scenario uses the actions described in table 1.

## 4 Functional Security Analysis

The approach described in the following can be decomposed into three basic steps. The first one is the derivation of the functional model from the use case descriptions in terms of an action oriented system. In a second step the system at stake is defined and possible instantiations of the first functional model are elaborated. In a third and final step, the actual requirements are derived in a systematic way, resulting in a consistent and complete set of security requirements.

**Table 1.** Actions for the example system

| Action | Explanation |
|---|---|
| *send*(cam(pos)) | A roadside unit broadcasts a cooperative awareness message cam concerning a danger at position pos. |
| *sense*($ESP_i$, sW) | The ESP sensor of vehicle $V_i$ senses slippery wheels (sW). |
| *pos*($GPS_i$, pos) | The GPS sensor of vehicle $V_i$ computes its position. |
| *send*($CU_i$, cam(pos)) | The communication unit $CU_i$ of vehicle $V_i$ sends a cooperative awareness message cam concerning the assumed danger based on the slippery wheels measurement for position pos. |
| *rec*($CU_i$, cam(pos)) | The communication unit $CU_i$ of vehicle $V_i$ receives a cooperative awareness message cam for position pos from another vehicle or a roadside unit. |
| *fwd*($CU_i$, cam(pos)) | The communication unit $CU_i$ of vehicle $V_i$ forwards a cooperative awareness message cam for position pos. |
| *show*($HMI_i$, warn) | The human machine interface $HMI_i$ of Vehicle $V_i$ shows its driver a warning warn with respect to the relative position. |

### 4.1 Functional Model

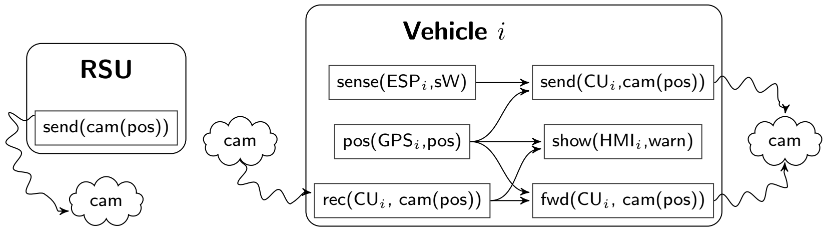
Information flow between systems and system components is highly complex, especially given that a system can evolve via the replacement of its components. Consequently, an important aspect of security evaluation is the analysis of the potential information flows. We use the analysis of the potential information flows to derive the dependencies for the functional model.

For the description of the functional model from the use cases an action-oriented approach is chosen. The approach is based on the work from [18]. For reasons of simplicity and readability the formal description of the model is omitted here and a graphical representation is used to illustrate the behaviour of the evaluation target.

A functional model can be derived from a use case description by identifying the atomic actions in the use case description. These actions are set into relation by defining the functional flow among them. This action oriented approach considers possible sequences of actions (control flow) and information flow (input/output) between interdependent actions.

In the case of highly distributed systems and especially a distributed system of distributed systems, it is very common that use cases do not cover a complete functional cycle throughout the whole system under investigation. Rather only certain components of the system are described regarding their behaviour. This must be kept in mind when deriving the functional model. In order to clarify this distinction, functional models that describe only parts of the overall system behaviour will be called *functional component model*.

Figures 1(a) and 1(b) show functional component models for a roadside unit and a vehicle respectively. These models are derived from the example use cases



(a) Roadside Unit

(b) Vehicle

**Fig. 1.** Functional component models

given in Sect. 3.1. The functional flow arrows outside of the vehicle's boundaries refer to functional flows between different instances of the component, whilst internal flow arrows refer to flows within the same instance of the component. For the given example, the external flows represent data transmission of one system to another, whilst the internal flows represent communication within a single system.

### 4.2 System of Systems Instances

Based on the functional component model, one may now start to reason about the overall system of systems which consists of a number of instances of the functional components. The synthesis of the internal flow between the actions within the component instances and the external flow between systems (in this case vehicles and roadside units) builds the global system of systems behaviour. In order to model instances of the global system of systems, all structurally different combinations of component instances shall be considered. Isomorphic combinations can be neglected. Finally, all possible instances may be regrouped and the system's boundary actions (denoting the actions that are triggered by or influence the system environment) have to be identified. These will be the basis for the security requirements definition in the next step.

In Fig. 2 an example for a possible SoS instance combining use cases 1 and 3 comprising a roadside unit and a vehicle is presented. In this SoS instance vehicle $V_w$ receives cooperative awareness message from a RSU.

### 4.3 Functional Security Requirements Identification

The set of possible instantiations of the functional component model is used in a next step to derive security requirements. First, the boundary actions of the system model instances are determined. Let the term *boundary action* refer to the actions that form the interaction of the internals of the system with the outside world. These are actions that are either triggered by occurrences outside of the system or actions that involve changes to the outside of the system.

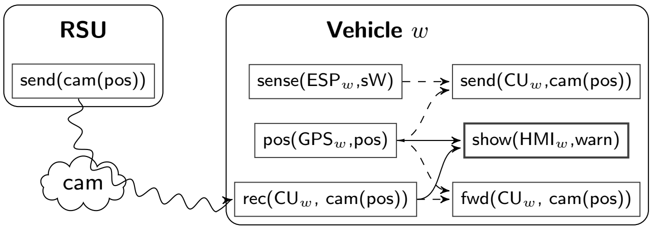


**Fig. 2.** Vehicle $w$ receives warning from RSU

With the boundary actions being identified, one may now follow the functional graph backwards. Beginning with the boundary actions by which the system takes influence on the outside, we may propagate backwards along the functional flow. These backwards references basically describe the functional dependencies of actions among each other. From the functional dependency graph we may now identify the end points - the boundary actions that trigger the system behaviour that depends on them. Between these and the corresponding starting points, the requirement exists that without such an action happening as input to the system, the corresponding output action must not happen as well. From this we formulate the security goal of the system at stake:

*Whenever a certain output action happens, the input actions that presumably led to it must actually have happened.*

*Example 1 (Boundary Actions and Dependencies).* In the SoS instance in Fig. 2 we are interested to identify the authenticity requirements for the boundary action $show(HMI_w, \text{warn})$. Following backwards along the functional flow we derive that the output action $show(HMI_w, \text{warn})$ is depending on the input actions $pos(GPS_w, \text{pos})$ of vehicle $w$ and $send(\text{cam}(\text{pos}))$ of the $RSU$.

These dependencies shall now be enriched by additional parameters. In particular, it shall be identified which is the entity that must be assured of the respective authenticity requirements. With these additional parameters set, we may utilise the following definition of authenticity from the formal framework of Fraunhofer SIT [8] to specify the identified requirements.

**Definition 1.** *$auth(a, b, P)$: Whenever an action $b$ happens, it must be authentic for an Agent $P$ that in any course of events that seem possible to him, a certain action $a$ has happened (for a formal definition see [8]).*

*Example 2 (Derive Requirements from Dependencies).* For the dependencies in Example 1 this leads to the following authenticity requirements with respect to the action $show(HMI_w, \text{warn})$:

- It must be authentic for the driver of vehicle $w$ that the relative position of the danger he/she is warned about is based on correct position information of his/her vehicle. Formally: $auth(pos(GPS_w, \text{pos}), show(HMI_w, \text{warn}), D_w)$

- It must be authentic for the driver of vehicle $w$ that the roadside unit issued the warning. Formally: $\mathit{auth}(\mathit{send}(\mathrm{cam}(\mathrm{pos})), \mathit{show}(\mathit{HMI}_w, \mathrm{warn}), D_w)$

It shall be noted that the requirements elicitation process in this case utilises positive formulations of how the system should behave, rather than preventing a certain malicious behaviour. Also it has to be stressed that this approach guarantees for the system / component architect to be free regarding the choice of concepts during the security engineering process.

This manual analysis may reveal that certain functional dependencies are presented only for performance reasons. This can be valuable input for the architects as well, and sometimes reveals premature decisions about mechanisms that were already done during the use case definition phase.

This approach cannot prevent the specification of circular dependencies among systems' actions but usually this is avoided for well-defined use cases. This actually originates from the fact that every action represents a progress in time. Accordingly an infinite loop among actions in the system would indicate that the system described will not terminate. The requirements derivation process will however highlight every functional dependency that is described within the use cases. Accordingly, when the use case description incorporates more than the sheer safety related functional description, additional requirements may arise. Therefore, the requirements have to be evaluated towards their meaning for the system's safety. Whilst one can be assured not to have missed any safety relevant requirement, this is a critical task because misjudging a requirement's relevance would induce security holes. Once an exhaustive list of security requirements is identified, a requirements categorisation and prioritisation process can evaluate them according to a maximum acceptable risk strategy.

### 4.4 Formalisation

Formally, the functional flow among actions can be interpreted as an ordering relation $\zeta_i$ on the set of actions $\Sigma_i$ in a certain system instance $i$. To derive the requirements the reflexive transitive closure $\zeta_i^*$ is constructed. In the following we assume that the functional flow graph is sequential and free of loops, as every action can only depend on past actions. Accordingly, the relation is anti-symmetric. $\zeta_i^*$ is a partial order on $\Sigma_i$, with the maximal elements $max_i$ corresponding to the outgoing boundary actions and the minimal elements $min_i$ corresponding to the incoming boundary actions. After restricting $\zeta_i^*$ to these elements $\chi_i = \{(x, y) \in \Sigma_i \times \Sigma_i \mid (x, y) \in \zeta_i^* \wedge x \in min_i \wedge y \in max_i\}$ this new relation represents the authenticity requirements for the corresponding system instance: *For all* $x, y \in \Sigma_i$ *with* $(x, y) \in \chi_i$ : $\mathit{auth}(x, y, \mathit{stakeholder}(y))$ is a requirement. Accordingly the union of all these requirements for the different instances poses the set of requirements for the whole system. This set can be reduced by eliminating duplicate requirements or by use of first-order predicates for a parameterised notation of similar requirements.

*Example 3 (Formal Derivation of Authenticity Requirements).* For the given system model instances, we may now identify the authenticity requirements for the action $show(HMI_w, \text{warn})$ using the actions and abbreviations defined in table 1. Graphically, this could be done by reversing the arrows and removing the dotted arrows and boxes.

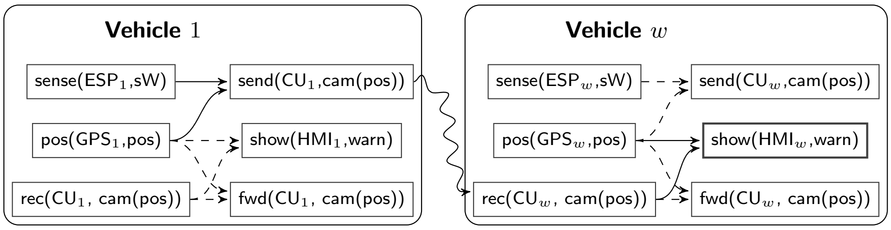


**Fig. 3.** Vehicle $w$ receives a warning from vehicle 1

Figure 3 shows an example for a possible SoS instance combining use cases 2 and 3 comprising two vehicles. In this SoS instance vehicle $V_w$ receives cooperative awareness message from vehicle $V_1$. Formally, for the SoS instance depicted in Fig. 3, we can analyse:

$$\begin{aligned}
\zeta_1 = &\{(sense(ESP_1, \text{sW}), send(CU_1, \text{cam}(\text{pos}))),\\
&(pos(GPS_1, \text{pos}), send(CU_1, \text{cam}(\text{pos}))),\\
&(send(CU_1, \text{cam}(\text{pos})), rec(CU_w, \text{cam}(\text{pos}))),\\
&(pos(GPS_w, \text{pos}), show(HMI_w, \text{warn})),\\
&(rec(CU_w, \text{cam}(\text{pos})), show(HMI_w, \text{warn}))\}\\
\zeta_1^* = &\zeta_1 \cup \{(x, x) \mid x \in \Sigma\} \cup \{\\
&(sense(ESP_1, \text{sW}), rec(CU_w, \text{cam}(\text{pos}))),\\
&(sense(ESP_1, \text{sW}), show(HMI_w, \text{warn})),\\
&(pos(GPS_1, \text{pos}), rec(CU_w, \text{cam}(\text{pos}))),\\
&(pos(GPS_1, \text{pos}), show(HMI_w, \text{warn})),\\
&(send(CU_1, \text{cam}(\text{pos})), show(HMI_w, \text{warn}))\}\\
\chi_1 = &\{(sense(ESP_1, \text{sW}), show(HMI_w, \text{warn})),\\
&(pos(GPS_1, \text{pos}), show(HMI_w, \text{warn})),\\
&(pos(GPS_w, \text{pos}), show(HMI_w, \text{warn}))\}
\end{aligned}$$

For further analysis we consider a possible SoS instance combining use cases 2, 3 and 4 comprising three vehicles as shown in Fig. 4. In this SoS instance vehicle $V_2$ forwards warnings from vehicle $V_1$ to vehicle $V_w$.

An analysis of the SoS instance with 3 vehicles as depicted in Fig. 4 will result in:

$$\chi_2 = \chi_1 \cup \{(pos(GPS_2, \text{pos}), show(HMI_w, \text{warn}))\}$$

In the given SoS model the forwarding of a message is restricted by a *position based forwarding policy* with respect to the distance from the danger that is being

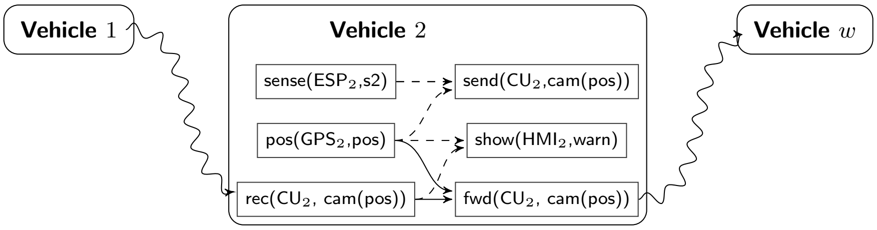


**Fig. 4.** Vehicle 2 forwards warnings (vehicles 1, 2 and $w$ are instances from Fig. 1)

warned about and the time of issue of the danger sensing. We could therefore assume a maximal number of system instances involved general enough to cover all these cases, e.g. by utilising a description in a parameterised way. An analysis for an SoS instance with $i$ vehicles will result in:

$$\chi_i = \chi_{i-1} \cup \{(pos(GPS_i, \text{pos}), show(HMI_w, \text{warn}))\}$$

The first three elements in each $\chi_i$ will obviously always be the same in all instances of the example. The rest of the elements can be expressed in terms of first-order predicates. This leads to the following authenticity requirements for all possible system instances for the action $show(HMI_w, \text{warn})$:

$$auth(pos(GPS_w, \text{pos}), show(HMI_w, \text{warn}), D_w) \tag{1}$$
$$auth(pos(GPS_1, \text{pos}), show(HMI_w, \text{warn}), D_w) \tag{2}$$
$$auth(sense(ESP_1, \text{sW}), show(HMI_w, \text{warn}), D_w) \tag{3}$$
$$\forall x \in V_{forward} : auth(pos(GPS_x, \text{pos}), show(HMI_w, \text{warn}), D_w) \tag{4}$$

$V_{forward}$ denotes the set of vehicles per system instance, that forward the warning message.

As mentioned above, the resulting requirements have to be evaluated regarding their meaning for the functional safety of the system. For the first three requirements the argumentation is very straight forward regarding why they have to be fulfilled:

1. It must be authentic for the driver that the relative position of the danger he/she is warned about is based on correct position information of his/her vehicle.
2. It must be authentic for the driver that the position of the danger he/she is warned about is based on correct position information of the vehicle issuing the warning.
3. It must be authentic for the driver that the danger he/she is warned about is based on correct sensor data.

The last requirement (4) however must be further evaluated. Studying the use case, we see that this functional dependency originates from the position based forwarding policy. This policy is introduced for performance reasons, such that bandwidth is saved by not flooding the whole network. Braking this requirement would therefore result either in a smaller or in a larger broadcasting area. As bad as those cases may be, they cannot cause the warning of a driver that should not be warned. Therefore we do not consider requirement (4) to be a safety related authenticity requirement. It can be considered a requirement regarding availability by preventing the denial of a service or unintended consumption of bandwidth.

In practice, the method described here has been applied in the project EVITA [23] to derive authenticity requirements for the development of a new automotive on-board architecture utilising vehicle-to-vehicle and vehicle-to-infrastructure communication. A total of 29 authenticity requirements have been elicited by means of a system model comprising 38 component boundary actions with 16 system boundary actions comprising 9 maximal and 7 minimal elements.

## 5 Tool-assisted Requirements Identification

The method for deriving authenticity requirements as described in the previous section relies on manual identification and processing only. In this section we will give an example on how to use the capabilities of existing tools, such as the SH verification tool [20] in order to facilitate the process especially for larger models.

As the previous section explained, the basis for the systematic identification of authenticity requirements for a given system is the relations between maxima and minima of the partial order of functional dependence. In this approach we first identified the direct relations of adjacent actions, then built the reflexive transitive closure and finally extracted those relations from this set that exist between maxima and minima of this partial order.

The tool-assisted approach will proceed in reverse order. First we will identify the maxima and minima of the partial order – without deriving the actual partial order – and then we will identify combinations of maxima and minima that are related by functional dependence. This approach will be illustrated with a simple example first, to provide the general idea and then with a more complex example, in order to demonstrate the application of abstraction techniques to cover the analysis of non-trivial systems.

### 5.1 Formal Modelling Technique

In order to analyse the system behaviour with tool support, an appropriate formal representation has to be chosen. In our approach, we choose an operational finite state model of the behaviour of the given vehicular communication scenario that is based on *Asynchronous Product Automata (APA)*, a flexible operational specification concept for cooperating systems [20]. An APA consists of a family of so called *elementary automata* communicating by common components of their state (shared memory). We now introduce the formal modelling techniques used, and illustrate the usage by our collaboration example.

**Definition 2 (Asynchronous Product Automaton (APA)).**
*An* Asynchronous Product Automaton *consists of*

- *a family of* state sets $Z_s, s \in \mathbb{S}$,
- *a family of* elementary automata $(\Phi_t, \Delta_t), t \in \mathbb{T}$ *and*
- *a* neighbourhood relation $N : \mathbb{T} \to \mathfrak{P}(\mathbb{S})$.

$\mathbb{S}$ *and* $\mathbb{T}$ *are index sets with the names of state components and of elementary automata and* $\mathfrak{P}(\mathbb{S})$ *is the power set of* $\mathbb{S}$.
*For each elementary automaton* $(\Phi_t, \Delta_t)$ *with* Alphabet $\Phi_t$, *its* state transition relation *is*

$$\Delta_t \subseteq \bigtimes_{s \in N(t)}(Z_s) \times \Phi_t \times \bigtimes_{s \in N(t)}(Z_s).$$

*For each element of* $\Phi_t$ *the state transition relation* $\Delta_t$ *defines state transitions that change only the state components in* $N(t)$. *An APA's (global)* states *are elements of* $\bigtimes_{s \in \mathbb{S}}(Z_s)$. *To avoid pathological cases it is generally assumed that* $N(t) \neq \emptyset$ *for all* $t \in \mathbb{T}$.
*Each APA has one* initial state $q_0 = (q_{0s})_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$.
*In total, an APA* $\mathbb{A}$ *is defined by*

$$\mathbb{A} = ((Z_s)_{s \in \mathbb{S}}, (\Phi_t, \Delta_t)_{t \in \mathbb{T}}, N, q_0).$$

*An elementary automaton* $(\Phi_t, \Delta_t)$ *is* activated *in a state* $p = (p_s)_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$ *as to an* interpretation $i \in \Phi_t$, *if there are* $(q_s)_{s \in N(t)} \in \bigtimes_{s \in N(t)}(Z_s)$ *with* $((p_s)_{s \in N(t)}, i, (q_s)_{s \in N(t)}) \in \Delta_t$.
*An activated elementary automaton* $(\Phi_t, \Delta_t)$ *can execute a state transition and produce a* successor state

$$q = (q_r)_{r \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s), \textit{ if}$$

$$q_r = p_r \textit{ for } r \in \mathbb{S} \setminus N(t) \textit{ and } ((p_s)_{s \in N(t)}, i, (q_s)_{s \in N(t)}) \in \Delta_t.$$

*The corresponding state transition is* $(p, (t, i), q)$.

For the following analysis by model checking and abstraction we use a reduced version of the functional component model of a vehicle that corresponds to the functional model illustrated in Fig. 1(b) but does not contain the $forward$ action.

*Example 4 (Finite State Model of the Collaboration Components).* The vehicle component model described in Sect. 4.1 is specified for the proposed analysis method using the following *APA state components* for each of the vehicles:

$$\begin{aligned}
\mathbb{S}_i = \{&esp_i, gps_i, hmi_i, bus_i, net\}, \text{ with}\\
Z_{esp_i} &= \mathfrak{P}(\{\text{sW}\}),\\
Z_{gps_i} &= \mathfrak{P}(\{pos1, pos2, pos3, pos4\}),\\
Z_{hmi_i} &= \mathfrak{P}(\{\text{warn}\}),\\
Z_{bus_i} &= \mathfrak{P}(Z_{esp} \cup Z_{gps} \cup Z_{hmi}) \text{ and}\\
Z_{net} &= \mathfrak{P}(\{\text{cam}\} \times \{V_1, V_2, V_3, V_4\} \times Z_{gps}).
\end{aligned}$$

The *inputs* to the vehicle model are represented by the state components $esp_i$ and $gps_i$. $esp_i$ represents input measurements taken by the ESP sensor. A pending data set here will trigger the *sense* action for slippery wheels. $gps_i$ represents the derivation of GPS position information. Pending data here will trigger the *pos* action for retrieving the current position of the vehicle.

The *outputs* of the vehicle model are represented by the state component $hmi_i$ that represents the HMI interface's display, showing (warning) information to the driver. The *show* action will push information to this medium.

Internally the vehicle component has an additional state component $bus_i$ representing its internal communication bus. It is filled with information from the *rec*, *sense* and *pos* action and read by the *send* and *forward* action.

Finally, *net* is a shared state component between all the vehicles that represents the wireless communication medium. A pending message here will trigger the *rec* action of the component. The actions *send* and *forward* will push a message into this medium.

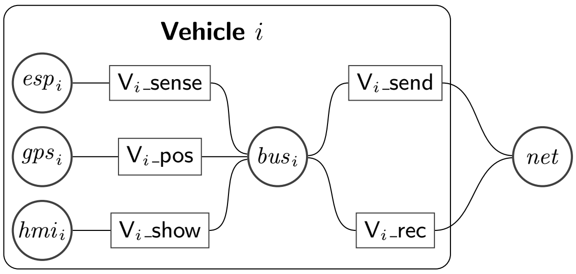


**Fig. 5.** APA model of a vehicle

The *elementary automata* $\mathbb{T}_i = \{V_i\_pos, V_i\_sense, V_i\_rec, V_i\_send, V_i\_show\}$ represent the possible actions that the systems can take. These specifications are represented in the data structures and initial configuration of the state components in the APA model. Elementary automata and state components of the APA model of a vehicle are depicted in Fig. 5. The lines in Fig. 5 between state components and elementary automata represent the neighbourhood relation.

The state transition relation for the APA model of a vehicle is given by:

$$\begin{aligned}
\Delta_{V_i\_sense} =& \{((esp_i, bus_i), (esp), (esp_i \backslash \{esp\}, bus_i \cup \{esp\}))\\
& \in (Z_{esp_i} \times Z_{bus_i}) \times ESP \times (Z_{esp_i} \times Z_{bus_i}) \mid esp \in esp_i\}\\
\Delta_{V_i\_pos} =& \{((gps_i, bus_i), (gps), (gps_i \backslash \{gps\}, bus_i \cup \{gps\})\\
& \in (Z_{gps_i} \times Z_{bus_i}) \times GPS \times (Z_{gps_i} \times Z_{bus_i}) \mid gps \in gps_i\}
\end{aligned}$$

$$\begin{aligned}
\mathit{\Delta}_{V_i\_send} =& \{((bus_i, net), (esp, gps, msg), (bus_i \setminus \{esp, gps\}, net \cup \{msg\})) \\
& \in (Z_{bus_i} \times Z_{net}) \times (ESP \times GPS \times NET) \times (Z_{bus_i} \times Z_{net}) \mid \\
& esp \in bus_i \land gps \in bus_i \land msg = (\mathrm{cam}, gps)\} \\
\mathit{\Delta}_{V_i\_rec} =& \{((net, bus_i), (msg, gps, \mathrm{warn}), (net \setminus \{msg\}, bus_i \setminus \{gps\} \cup \{\mathrm{warn}\})) \\
& \in (Z_{net} \times Z_{bus}) \times (NET \times GPS \times HMI) \times (Z_{net} \times Z_{bus_i}) \mid \\
& msg \in net \land gps \in bus_i \land distance(msg, gps) < \mathrm{range}\} \\
\mathit{\Delta}_{V_i\_show} =& \{((bus_i, hmi_i), (\mathrm{warn}), (bus_i \setminus \{\mathrm{warn}\}, hmi \cup \{\mathrm{warn}\})) \\
& \in (Z_{bus_i} \times Z_{hmi_i}) \times HMI \times (Z_{bus_i} \times Z_{hmi_i}) \mid \mathrm{warn} \in bus_i\}
\end{aligned}$$

The model is parameterised by $i$ except for the shared state component *net*.

### 5.2 Formal Representation of System of Systems Instances

The SoS instance that we investigate first includes two vehicle components that are assumed to be within the wireless transmission range similar to the example given in Fig. 3. In this SoS instance vehicle $V_2$ receives cooperative awareness message from vehicle $V_1$. Therefore the *net* components are mapped together, such that outputs of each one of the vehicles are input for the other vehicle. The rest of the inputs (Sensors and GPSs) as well as outputs (displays) are not internal parts of the system but filled and read by the systems environment. It should be noted that timing behaviour is not included in the model, because we solely want to retrieve functional dependencies. As we want to instantiate $V_1$ to perform use Case 2 and $V_2$ to perform use Case 3, we set

- $V_1$'s sensor input to a measurement of slippery wheels sW,
- $V_1$'s GPS input to some position pos1 that is within warning range of $V_2$ and
- $V_2$'s GPS input to some position pos2 that is within warning range of $V_1$.

*Example 5 (Finite State Model of an SoS Instance with 2 Vehicles).*
The state components for this instance are

$$\mathbb{S} = \{esp_1, pos_1, bus_1, hmi_1, esp_2, pos_2, bus_2, hmi_2, net\}$$

and the set of elementary automata is

$$\begin{aligned}
\mathbb{T} = \{& V_1\_sense, V_1\_pos, V_1\_send, V_1\_rec, V_1\_show, \\
& V_2\_sense, V_2\_pos, V_2\_send, V_2\_rec, V_2\_show\}.
\end{aligned}$$

The neighbourhood relation $N(t)$ can be read directly from the graphical illustration in Fig. 6. The initial state for our simulation is defined as:

$$\begin{aligned}
q_0 =& (q_{0\_esp_1}, q_{0\_gps_1}, q_{0\_bus_1}, q_{0\_hmi_1}, q_{0\_esp_2}, q_{0\_gps_2}, q_{0\_bus_2}, q_{0\_hmi_2}, q_{0\_net}) \\
=& (\{\mathrm{sW}\}, \{\mathrm{pos1}\}, \emptyset, \emptyset, \emptyset, \{\mathrm{pos2}\}, \emptyset, \emptyset, \emptyset).
\end{aligned}$$

For example $(q_0, (V_1\_sense, \mathrm{sW}), (\emptyset, \{\mathrm{pos1}\}, \{\mathrm{sW}\}, \emptyset, \emptyset, \{\mathrm{pos2}\}, \emptyset, \emptyset, \emptyset))$ is a state transition of this SoS instance.

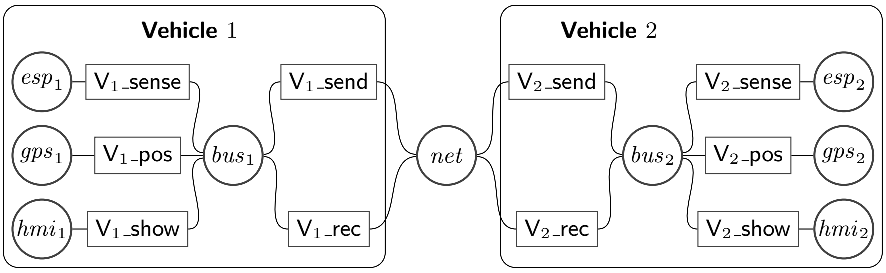


**Fig. 6.** APA model of a SoS instance with 2 vehicles

### 5.3 Computation of System of Systems Behaviour

Formally, the behaviour of our operational APA model of the vehicular communication system is described by a reachability graph. In the literature this is sometimes also referred to as labelled transition system (LTS).

**Definition 3 (Reachability graph).**
*The behaviour of an APA is represented by all possible coherent sequences of state transitions starting with initial state $q_0$. The sequence $(q_0, (t_1, i_1), q_1)$ $(q_1, (t_2, i_2), q_2)$ $\ldots (q_{n-1}, (t_n, i_n), q_n)$ with $i_k \in \Phi_{t_k}$ represents one possible sequence of actions of an APA.*

*State transitions $(p, (t, i), q)$ may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA: $(p, (t, i), q)$ is the edge leading from $p$ to $q$ and labelled by $(t, i)$. The subgraph reachable from the node $q_0$ is called the* reachability graph *of an APA.*

We used the *Simple Homomorphism (SH) verification tool* [20] to analyse the functional component model for different concrete instantiations of the model. The tool has been developed at the *Fraunhofer-Institute for Secure Information Technology*. The applied specification method based on Asynchronous Product Automata is supported by this tool. The tool manages the components of the model, allows to select alternative parts of the specification and automatically *glues* together the selected components to generate a combined model of the APA specification. It provides components for the complete cycle from formal specification to exhaustive validation as well as visualisation and inspection of computed reachability graphs and minimal automata. The tool provides an editor to define homomorphisms on action languages, it computes corresponding minimal automata [3] for the homomorphic images and checks simplicity of the homomorphisms. If it is required to inspect some or all paths of the graph to check for the violation of a security property, as it is usually the case for liveness properties, then the tool's temporal logic component can be used. Temporal logic formulae can also be checked on the abstract behaviour (under a simple homomorphism). The method for checking approximate satisfaction of properties fits exactly to the built-in simple homomorphism check [20].

**Computation of SoS Instance's Behaviour.** Starting with the analysis, we define a representation of the component behaviour in preamble language of the SH verification tool according to the use cases. Then for a first simple example, we instantiated it twice – with a warning vehicle $V_1$ and a vehicle that receives the warning $V_2$, similar to Fig. 6. After an initial state is selected, the reachability graph is automatically computed by the SH verification tool. Fig. 7 shows the reachability graph resulting from the analysis of the model instance in Fig. 6. Please note that the tool prints the state $q_0$ as $M$-1.

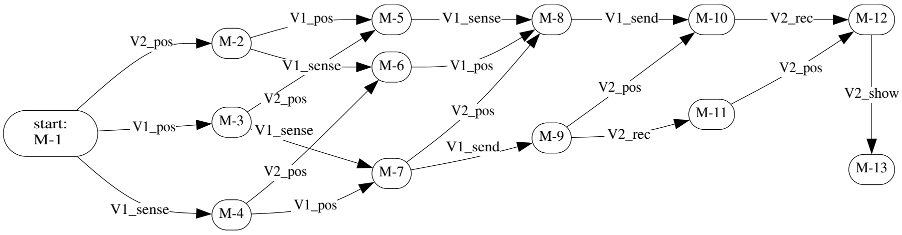


**Fig. 7.** Reachability graph of SoS instance with two vehicles in the SH verification tool

### 5.4 Evaluating the Functional Dependence Relation

Starting from the model of the system components and their instantiations the reachability analysis provides a graph with serialised traces of actions in the system. In order to identify the *minima* of such a system, we look at the initial state *M-1* of the reachability graph. Every action that leaves the initial state on any of the traces is obviously a minimum, because it does not functionally depend on any other action to have occurred before. In order to identify the *maxima* we investigate those actions leading to the dead state from any trace. These actions do not trigger any further action after they have been performed.

*Example 6 (The SH verification tool's result for Example 5).*

```
 The minima of this analysis:          The corresponding maxima:
M–1
 V1_sense  M–4                         M–12  V2_show
 V1_pos  M–3                           M–13+
 V2_pos  M–2                           +++  dead  +++
```

Since we now have identified the maxima and minima of the partial order of functionally dependent actions, we must evaluate which of these maxima have a functional dependence relation. For this simple example, it can easily be seen from the reachability graph, that the maximum only occurs after all the minima have occurred in each of the traces, i.e. the maximum depends on all the identified minima. Accordingly, the simple example has the following set of requirements: $auth(V_1\_sense, V_2\_show, D_2)$, $auth(V_1\_pos, V_2\_show, D_2)$, $auth(V_2\_pos,\ V_2\_show, D_2)$.

### 5.5 Abstraction Based Verification Concept

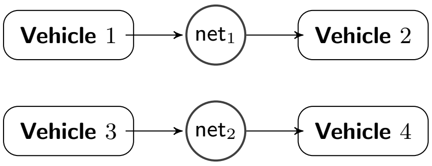


**Fig. 8.** Model for SoS instance with four vehicles

In order to further demonstrate our approach for a more complex scenario, a second example of a SoS instance that includes four vehicles – two pairs of two vehicles, each pair within communication range but out of range from the other pair, performing the same scenario each ($V_1$ warns $V_2$ and $V_3$ warns $V_4$) – can be seen in Fig. 8 with the corresponding reachability graph in Fig. 9.

The minima of this analysis:

M–1
V1_sense M–7
V3_sense M–6
V1_pos M–5
V2_pos M–4
V3_pos M–3
V4_pos M–2

The corresponding maxima:

M–168 V2_show
M–167 V4_show
M–169+
+++ dead +++

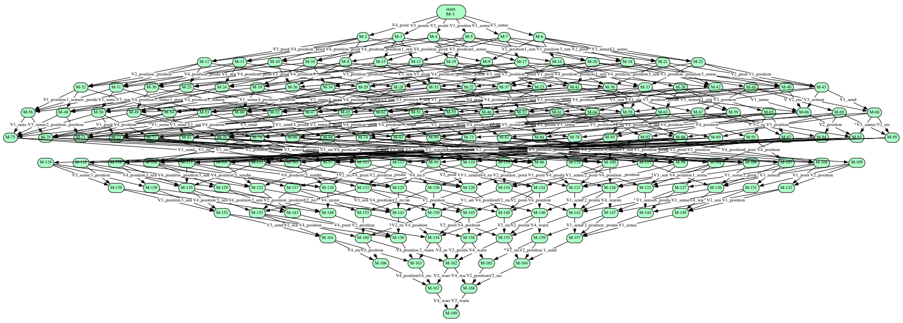


**Fig. 9.** Reachability graph of SoS instance with four vehicles in the SH verification tool

Obviously, the reachability graph in Fig. 9 that is generated from the complex scenario cannot be evaluated directly. However the technique of abstraction can help us to identify if a given maximum functionally depends on a given minimum.

Behaviour abstraction of an APA can be formalised by language homomorphisms, more precisely by alphabetic language homomorphisms $h : \Sigma^* \rightarrow \Sigma'^*$. By these homomorphisms certain transitions are ignored and others are renamed, which may have the effect, that different transitions are identified with one another. A mapping $h : \Sigma^* \rightarrow \Sigma'^*$ is called a language homomorphism if $h(\varepsilon) = \varepsilon$ and $h(yz) = h(y)h(z)$ for each $y, z \in \Sigma^*$. It is called alphabetic, if $h(\Sigma) \subset \Sigma' \cup \{\varepsilon\}$.

In order to analyse dependencies for each pair of maximum and minimum in the graph in Fig. 9, we can now define alphabetic language homomorphisms that will map every action except the given pairs of maximum and minimum to $\varepsilon$. The computed abstract representations then provide a visualisation focussing on the two actions, helping us to see directly, if the given maximum can occur independent of the given minimum or if it depends on the minimum's prior occurrence.

*Example 7.* The minimal automaton computed from the reachability graph under the homomorphism that preserves $V_1$*_sense* and $V_2$*_show* is depicted in Fig. 10. This graph indicates a functional dependence relation between the given maximum and minimum.

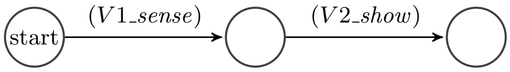


**Fig. 10.** Minimal automaton with maximum and minimum

The homomorphism preserving $V_1$*_sense* and $V_4$*_show* will result in the graph depicted in Fig. 11 that indicates the given maximum and minimum not to have a functional dependence relation.

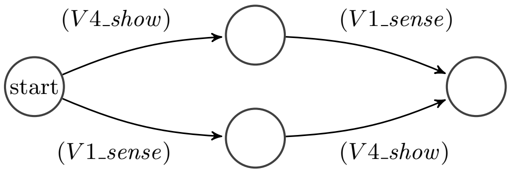


**Fig. 11.** Minimal automaton with independent maximum and minimum

Following this approach, testing each of the maxima with each of the minima for functional dependence, the complex scenario has the following set of requirements (with the stakeholder of $V_4$ to be driver $D_4$ of course):

$auth(V_1\_sense, V_2\_show, D_2)$, $auth(V_1\_pos, V_2\_show, D_2)$,
$auth(V_2\_pos, V_2\_show, D_2)$, $auth(V_3\_sense, V_4\_show, D_4)$,
$auth(V_3\_pos, V_4\_show, D_4)$, $auth(V_4\_pos, V_4\_show, D_4)$.

## 6 Conclusion

The presented approach for deriving safety-critical authenticity requirements in SoS solves several issues compared to existing approaches. It incorporates a clear scheme that will ensure a consistent and complete set of security requirements. Also it is based directly on the functional analysis, ensuring the safety of the system at stake. The systematic approach that incorporates formal semantics leads directly to the formal validation of security, as it is required by certain evaluation assurance levels of Common Criteria (ISO/IEC 15408). Furthermore the difficulties of designing SoS are specifically targeted.

Starting from this set of very high-level requirements, the security engineering process may proceed. This will include decisions regarding the mechanisms to be included. Accordingly the requirements have to be refined to more concrete requirements in this process. The design and refinement process may reveal further requirements regarding the internals of the system that have to be addressed as well.

Future work may include the derivation of confidentiality requirements in a similar way as was presented here. Though this will require for different security goals, as confidentiality is not related to safety in a similar way, but rather to privacy. Non-Repudiation may also be a target that should be approached in cooperation with lawyers in order to find the relevant security goals. Furthermore, the refinement throughout the design process should be evaluated regarding possibility of formalising it in schemes with respect to the security requirements refinement process.

For the tool-assisted method in Sect. 5, traditional model checking techniques allow a verification of SoS behaviour only for systems with very few components. We are developing an abstraction based approach to extend our current tool supported verification techniques to such families of systems that are usually parameterised by a number of replicated identical components. In [19] we demonstrated our technique by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario. In [21] we defined uniform parameterisations of phase based cooperations in terms of formal language theory. For such systems of cooperations a kind of self-similarity is formalised. Based on deterministic computations in shuffle automata a sufficient condition for self-similarity is given. Under certain regularity restrictions this condition can be verified by a semi-algorithm. For verification purposes, so called uniformly parameterised safety properties are defined. Such properties can be used to express privacy policies as well as security and dependability requirements. It is shown, how the parameterised problem of verifying such a property is reduced by self-similarity to a finite state problem.

**Acknowledgments.** Andreas Fuchs developed the work presented here in the context of the project EVITA (ID 224275) being co-funded by the European Commission within the Seventh Framework Programme. Roland Rieke developed the work presented here in the context of the projects Alliance Digital Product Flow (ADiWa) (ID 01IA08006F) and VOGUE (ID 01IS09032A) which are both funded by the German Federal Ministry of Education and Research.

## References

1. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Sec. Comput. 1(1), 11–33 (2004)
2. Bodeau, D.J.: System-of-Systems Security Engineering. In: In Proc. of the 10th Annual Computer Security Applications Conference, Orlando, Florida. pp. 228–235. IEEE Computer Society (1994)
3. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, New York (1974)
4. Firesmith, D.: Engineering security requirements. Journal of Object Technology 2(1), 53–68 (2003)
5. Fuchs, A., Rieke, R.: Identification of authenticity requirements in systems of systems by functional security analysis. In: Workshop on Architecting Dependable Systems (WADS 2009), in Proceedings of the 2009 IEEE/IFIP Conference on Dependable Systems and Networks, Supplementary Volume (2009), `http://sit.sit.fraunhofer.de/smv/publications/`
6. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Requirements engineering meets trust management: Model, methodology, and reasoning. In: In Proc. of iTrust 04, LNCS 2995. pp. 176–190. Springer-Verlag (2004)
7. Group, T.C.: TCG TPM Specification 1.2 revision 103. www.trustedcomputing.org (2006)
8. Gürgens, S., Ochsenschläger, P., Rudolph, C.: Authenticity and provability - a formal framework. In: Infrastructure Security Conference InfraSec 2002. Lecture Notes in Computer Science, vol. 2437, pp. 227–245. Springer Verlag (2002)
9. Haley, C.B., Laney, R.C., Moffett, J.D., Nuseibeh, B.: Security requirements engineering: A framework for representation and analysis. IEEE Trans. Software Eng. 34(1), 133–153 (2008)
10. Hatebur, D., Heisel, M., Schmidt, H.: A security engineering process based on patterns. In: Proceedings of the International Workshop on Secure Systems Methodologies using Patterns (SPatterns), DEXA 2007. pp. 734–738. IEEE Computer Society (2007), `http://www.ieee.org/`
11. Hatebur, D., Heisel, M., Schmidt, H.: A pattern system for security requirements engineering. In: Proceedings of the International Conference on Availability, Reliability and Security (AReS). pp. 356–365. IEEE (2007), `http://www.ieee.org/`
12. Hatebur, D., Heisel, M., Schmidt, H.: Analysis and component-based realization of security requirements. In: Proceedings of the International Conference on Availability, Reliability and Security (AReS). pp. 195–203. IEEE Computer Society (2008), `http://www.ieee.org/`
13. van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: ICSE '04: Proceedings of the 26th International Conference on Software Engineering. pp. 148–157. IEEE Computer Society, Washington, DC, USA (2004)

14. Liu, L., Yu, E., Mylopoulos, J.: Analyzing security requirements as relationships among strategic actors. In: 2nd Symposium on Requirements Engineering for Information Security (SREIS'02) (2002)
15. Mead, N.R.: How To Compare the Security Quality Requirements Engineering (SQUARE) Method with Other Methods . Tech. Rep. CMU/SEI-2007-TN-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2007)
16. Mead, N.R., Hough, E.D.: Security requirements engineering for software systems: Case studies in support of software engineering education. In: CSEET '06: Proceedings of the 19th Conference on Software Engineering Education & Training. pp. 149–158. IEEE Computer Society, Washington, DC, USA (2006)
17. Mellado, D., Fernández-Medina, E., Piattini, M.: A common criteria based security requirements engineering process for the development of secure information systems. Comput. Stand. Interfaces 29(2), 244–253 (2007)
18. Ochsenschläger, P., Repp, J., Rieke, R.: Abstraction and composition – a verification method for co-operating systems. Journal of Experimental and Theoretical Artificial Intelligence 12, 447–459 (June 2000), http://sit.sit.fraunhofer.de/smv/publications/, copyright: ©2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.
19. Ochsenschläger, P., Rieke, R.: Abstraction based verification of a parameterised policy controlled system. In: International Conference ”Mathematical Methods, Models and Architectures for Computer Networks Security” (MMM-ACNS-7). CCIS, vol. 1. Springer (September 2007), http://sit.sit.fraunhofer.de/smv/publications/, Springer
20. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing, The International Journal of Formal Method 11, 1–24 (1999)
21. Ochsenschläger, P., Rieke, R.: Uniform parameterisation of phase based cooperations. Tech. Rep. SIT-TR-2010/1, Fraunhofer SIT (2010), http://sit.sit.fraunhofer.de/smv/publications/
22. Papadimitratos, P., Buttyan, L., Hubaux, J.P., Kargl, F., Kung, A., Raya, M.: Architecture for Secure and Private Vehicular Communications. In: IEEE International Conference on ITS Telecommunications (ITST). pp. 1–6. Sophia Antipolis, France (June 2007)
23. Ruddle, A., Ward, D., Weyl, B., Idrees, S., Roudier, Y., Friedewald, M., Leimbach, T., Fuchs, A., Grgens, S., Henniger, O., Rieke, R., Ritscher, M., Broberg, H., Apvrille, L., Pacalet, R., Pedroza, G.: Security requirements for automotive onboard networks based on dark-side scenarios. EVITA Deliverable D2.3, EVITA project (2009), http://evita-project.org/deliverables.html
24. Sadeghi, A.R., Stüble, C.: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: NSPW '04: Proceedings of the 2004 workshop on New security paradigms. pp. 67–77. ACM, New York, NY, USA (2004)
25. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th USENIX Security Symposium. USENIX Association (2004)
26. Schaub, F., Ma, Z., Kargl, F.: Privacy requirements in vehicular communication systems. In: IEEE International Conference on Privacy, Security, Risk, and Trust (PASSAT 2009), Symposium on Secure Computing (SecureCom09). Vancouver, Canada (08/2009 2009), http://doi.ieeecomputersociety.org/10.1109/CSE.2009.135
27. Shirey, R.: Internet Security Glossary, Version 2. RFC 4949 (Informational) (Aug 2007), http://www.ietf.org/rfc/rfc4949.txt

# P6

# A TRUSTED INFORMATION AGENT FOR SECURITY INFORMATION AND EVENT MANAGEMENT

| | |
|---|---|
| **Title** | A Trusted Information Agent for Security Information and Event Management |
| **Authors** | Luigi Coppolino, Michael Jäger, Nicolai Kuntze and Roland Rieke |
| **Publication** | In *ICONS 2012, The Seventh International Conference on Systems, February 29 - March 5, 2012 - Saint Gilles, Reunion Island*, pages 6–12. |
| **ISBN/ISSN** | ISBN 978-1-61208-184-7 |
| **URL** | `http://www.thinkmind.org/index.php?view=article&articleid=icons_2012_1_20_20062`. |
| **Status** | Published |
| **Publisher** | IARIA |
| **Publication Type** | ThinkMind(TM) Digital Library |
| **Copyright** | 2012, IARIA |
| **Contribution of Roland Rieke** | Main Author; specific contribution is the elicitation and analysis of the security requirements for the critical infrastructure. |

Table 11: Fact Sheet Publication *P6*

Publication P6 [Coppolino, Jäger, Kuntze & Rieke, 2012] addresses the following research question:

RQ3 *How can security requirements for cooperating systems be elicited systematically?*

This paper demonstrates on an example of a critical infrastructure - a hydroelectric power plant - how security requirements for such SoS can be derived by application of the requirements elicitation method described in P5. The elicited requirements provide implications for the design of the security architecture which - in this case - leads to the application of trusted computing technology.

# A Trusted Information Agent for Security Information and Event Management

Luigi Coppolino
*Epsilon S.r.l.,*
*Naples, Italy*
*luigi.coppolino@epsilonline.com*

Michael Jäger
*Technische Hochschule Mittelhessen*
*Giessen, Germany*
*michael.jaeger@mni.thm.de*

Nicolai Kuntze and Roland Rieke
*Fraunhofer Institute for*
*Secure Information Technology*
*Darmstadt, Germany*
*{nicolai.kuntze,roland.rieke}@sit.fraunhofer.de*

***Abstract*—This paper addresses security information management in untrusted environments. A security information and event management system collects and examines security related events and provides a unifying view of the monitored system's security status. The sensors, which provide the event data, are typically placed in a non-protected environment at the boarder of the managed system. They are exposed to various kinds of attacks. Compromised sensors may lead to misjudgement on the system's state with possibly serious consequences. The particular security requirements arising from these problems are discussed for large scale critical infrastructures. The main contribution of this paper is a concept that provides trusted event reporting. Critical event sources are holistically protected such that authenticity of the security related events is guaranteed. This enables better assessment of the managed system's reliability and trustworthiness. As a proof of this concept, the paper presents an exemplary realisation of a trustworthy event source.**

***Keywords*-reliability aspects of security information and event management systems; trusted event reporting; trusted android application; critical infrastructure protection.**

## I. Introduction

Security information and event management (SIEM) systems provide important security services. They collect and analyse data from different sources, such as sensors, firewalls, routers or servers, and provide decision support based on anticipatory impact analysis. This enables adequate response to attacks as well as impact mitigation by adaptive configuration of countermeasures. The project MASSIF [1], a large-scale integrating project co-funded by the European Commission, addresses these challenges with respect to four industrial domains: (i) the management of the Olympic Games information technology (IT) infrastructure [2]; (ii) a mobile phone based money transfer service, facing high-level threats such as money laundering; (iii) managed IT outsource services for large distributed enterprises; and (iv) an IT system supporting a critical infrastructure (dam) [3].

Common to these use cases is the requirement to prove that a measured value has been acquired at a certain time and within a specified "valid" operation environment. Authenticity of such measures can only be assured together with authentication of the used device itself, it's configuration, and the software running at the time of the event.

In geographically dispersed infrastructures, various equipment, including the critical sources of event data, is often placed in non-protected environments. Therefore, attackers are able to access and manipulate this equipment with relative ease[4].

**Proposition 1.** *When physical access to the sensoring devices cannot be inhibited, an effective security solution must address detection of manipulations.*

Manipulated equipment can be used to hide critical conditions, generate false alerts, and in general cause misjudgement on system's state. Wrong assumptions about a system's state in turn can lead to false decisions with severe impact on the overall system.

**Proposition 2.** *Whenever a certain control decision is made, the input information that presumably led to it must be authentic.*

As a consequence, the system has to assure that all safety critical actions using sensor data must only use authentic sensor data. The question, which measurements and system control decisions are critical to the overall system behaviour, cannot be answerded independently of the concrete system and application context determined.

**Proposition 3.** *A risk assessment of the deployed monitoring capabilities is necessary.*

*Contribution:* By means of a representative example, namely a hydroelectric power plant in a dam, we analyse security threats for critical infrastructures and justify the relevance of the postulated propositions for adequate security requirements. Further, the paper presents both, a concept and a prototypical implementation for trustworthy event reporting. Digital signatures obviously can provide authenticity and integrity of recorded data [5]. However, a signature gives no information on the status of the measurement device at the time of measurement. Our solution, the *trusted information agent* (TIA), is based on trusted computing technology [6] and integrates industry approaches to the attestation of event reporter states. This approach provides a certain degree of trustworthiness and non-repudiation for the collected events, which can be used as a basis for risk assessment according to Proposition 3.

The paper is structured as follows. Section II gives an overview of the related work. In Section III we introduce the exemplary application scenario. We then elicit a number of specific security requirements from the application scenario and justify the propositions for our concept in Section IV. Based on these requirements, we address a solution for our propositions and describe the concept and a prototypical implementation of a trusted information agent in Section V. Finally, the paper ends with conclusions and an outlook in Section VI.

## II. Related Work

The paper addresses the integration of Trusted Computing concepts into SIEM systems for critical infrastructures based on examples from a hydroelectric power plant in a dam.

Security information and event management technology provides log management and compliance reporting as well as real-time monitoring and incident management for security events from networks, systems, and applications. Current SIEM systems' functionalities are discussed in [7]. SIEM systems manage security events but are not concerned with the trustworthiness of the event sources. Security requirements analysis and an authenticity concept for event sources is, however, the main topic of this paper. The specification of the application level security requirements is based on the formal framework developed by Fraunhofer SIT [8]. In this framework, systems are specified in terms of sequences of actions and security properties are constraints on these sequences. Applying the methods of this framework, we derive security requirements for the event sources in the dam scenario.

Dam monitoring applications with *automated data acquisition systems* (ADAS) are discussed in [9], [10]. Usually, an ADAS is organised as a *supervisory control and data acquisition* (SCADA) system with a hierarchical organisation. Details on SCADA systems organisation can be found in [11], [12]. In the majority of cases, SCADA systems have very little protection against the escalating cyber threats.

Compared to traditional IT systems, securing SCADA systems poses unique challenges. In order to understand those challenges and the potential danger, [4] provides a taxonomy of possible cyber attacks including cyber-induced cyber-physical attacks on SCADA systems.

Trusted Computing technology standards provide methods for reliably verifying a system's integrity and identifying anomalous and/or unwanted characteristics [6]. An approach for the generation of secure evidence records was presented in [13]. This approach, which is the basis for our proof-of-concept implementation, makes use of established hardware-based security mechanisms for special data recording devices. Our communication protocols extend the Trusted Network Connect (TNC) [14] protocol suite. We use the open source implementation of IF-MAP presented in [15].

## III. Application Scenario

Our analysis of security threats for critical infrastructures is based on examples from a hydroelectric power plant in a dam. The dam scenario is typical for critical infrastructures in many respects. On the one hand, it is a layered system with intra- and cross-layer dependencies, and, on the other hand, there are various other sources of complexity; several distinct functionalities influence controlling and monitoring activities. Moreover, different components, mechanisms, and operative devices are involved, each one with different requirements in terms of produced data and computational loads.

A dam might be devised for a multitude of purposes and its features are strictly related to the aims it is built for, e.g., food water supplying, hydroelectric power generation, irrigation, water sports, wildlife habitat granting, flow diversion, or navigation. Since a dam is a complex infrastructure, a huge number of parameters must be monitored in order to guarantee safety and security. Which parameters are actually monitored, depends on the dam's structure and design (earthfill, embankment or rockfill, gravity, concrete arch, buttress), the purpose (storage, diversion, detention, overflow), and the function (hydroelectric power generation, water supply, irrigation).

Table I
Dam instrumentation sensors

| Sensor | Parameter or physical event |
|---|---|
| Water level sensor (*WLS*) | Current water level (*wl*) |
| Inclinometer/Tiltmeter (*TM*) | Earth or wall inclination or tilt (*tm*) |
| Crackmeter (*CM*) | Wall/rock crack enlargement (*cm*) |
| Jointmeter (*JM*) | Joint shrinkage (*jm*) |
| Piezometer | Seepage or water pressure |
| Pressure cell | Concrete or embankment pressure |
| Turbidimeter | Fluid turbidity |
| Thermometer | Temperature |

Table I lists some of the most commonly employed sensors together with a brief explanation of their usage. The heterogeneity of currently used devices is a relevant challenge in the dam process control: they range from old industrial control systems, designed and deployed over the last 20 years and requiring extensive manual intervention by human operators, to more recently developed systems, conceived for automatic operations (SCADA). Indeed, the trend of development is toward increasingly automated dam control systems. While automation leads to more efficient systems and also prevents operating errors; on the downside, it poses a limit to human control in situations, where an operator would possibly foresee and manually prevent incidents.

Modern automated systems support remote management and also provide for centralised control of multiple infrastructures. As an example, the Terni hydroelectric complex, located about 150 Km in the north of Rome, is composed by 16 hydroelectric power plants, three reservoirs (Salto, Turano and Corbara), and one pumping plant, all of them supervised by a single remote command post located at Villa Valle.

As a severe disadvantage, increased automation and remote

Table II
SECURITY RELATED SCENARIOS AND THE RESPECTIVE MONITORING

| Monitored Event | Impact | Detection |
|---|---|---|
| Changes in the flow levels of the seepage channels | Seepages always affect dams (whatever their structure and design are). Seepage channels are monitored to evaluate the seepage intensity. A sudden change in flow levels could show that the structure is subject to internal erosion or to piping phenomena. This event can be the cause of dam cracks and failures | By inserting into the channel a weir with a known section the depth of water (monitored by using a water level sensor) behind the weir can be converted to a rate of flow. |
| Gates opening | Intake gates are opened to release water on a regular basis for water supply, hydroelectricity generation, etc. Moreover spillways gates(aka overflow channels) release water (during flood period) so that the water does not overtop and damage or even destroy the dam. Gates opening must be operated under controlled conditions since it may result in: i) Flooding of the underlying areas; ii) Increased rate of flow in the downstream that can ultimately result in a catastrophic flooding of down-river areas. | A tiltmeter (angle position sensor) can be applied to the gate to measure its position angle. |
| Changes in the turbine/infrastructure vibration levels | Increased vibrations of the infrastructure or the turbines in a hydro-powerplant can anticipate a failure of the structure. Possible reasons for such event include: i) earthquakes (Fukushima, Japan, a dam failure resulted in a village washed away ); ii) unwanted solicitations to the turbines (Sayano-Shushenskaya, Siberia, 75 dead due to a failure of the turbines in a hydro-powerplant). | Vibration sensors can be installed over structures or turbines to measure the stress level they are receiving. |
| Water levels overtake the alert thresholds | Spillway are used to release water when the reservoir water level reaches alert thresholds. If this does not happen the water overtops the dam resulting in possible damage to the crest of the dam (Taum Sauk hydroelectric power station). | This event can be used to detect unexpected discharges. Water level can also be correlated to other parameters to detect anomalous behaviour (e.g., not revealed gate opening). |

control raise a new class of security-induced safety issues, i.e., the possibility that cyber attacks against the IT layer of the dam ultimately result in damage to people and environment.

Dam monitoring aims towards identifying anomalous behaviour related to the infrastructure. Table II summarises a list of possible scenarios illustrating the necessity of monitoring specific parameters.

## IV. SECURITY REQUIREMENTS ANALYSIS

We use a model-based approach to systematically identify security requirements for the dam application scenario. Specifically, *authenticity* can be seen as the assurance that a particular action has occurred in the past. For a formal specification of the application-level authenticity requirements, we use Definition 1, which is taken from [8].

**Definition 1.** $auth(a,b,P)$: *Whenever an action b happens, it must be authentic for an Agent P that in any course of events that seem possible to him, a certain action a has happened.*

In [8] a *security modelling framework* (SeMF) for the formal specification of security properties was presented. Requirements are defined by specific constraints regarding sequences of actions than can or can not occur in a system's behaviour. Actions in SeMF represent an abstract view on actions of the real system, which models the *interdependencies* between actions and ignores their functionality. An action is specified in a parameterised format, consisting of the action's name, the acting agent and a variable set of parameters:

$$actionName(actingAgent, parameter1, parameter2, \dots)$$

Table III lists the dam scenario actions used for our security requirements analysis.

Table III
DAM ACTIONS

| Action | Description |
|---|---|
| $sense(WLS, wl)$ | Measurement of the water level. |
| $sense(TM, tm)$ | Measurement of the tilt. |
| $sense(CM, cm)$ | Measurement of the crack enlargement. |
| $sense(JM, jm)$ | Measurement of the joint shrinkage. |
| $sense(PP, power)$ | Measurement of voltage and current in the power grid. The power plant $PP$ sends commands $ppc$ to the dam control station depending on these measurements. |
| $sense(SDC, wdc)$ | Measurement of the water discharge on the penstock gates $PG$. |
| $sense(PG, open)$ | Reporting of the state of the penstock gates. |
| $display(DCS, X)$ | Display $X$ at the dam control station, with $X \in \{wl, tm, cm, jm, ppc, wdc, open\}$. |
| $activate(Admin, cmd)$ | Decision of the administrator, which command shall be triggered. |
| $exec(PG, cmd)$ | Command to be executed by penstock gates. |

We now analyse some possible misuse cases, which have been reported in the scenario deliverable [16] of the MASSIF project.

*Water level sensor compromise:* The attacker takes control of the water level sensors and uses them to send spoofed measurements to the dam control station (*DCS*). This hides the real status of the reservoir to the dam administrator (*Admin*). In this way, the dam can be overflown without alarms being raised by the monitoring system.

From this, we get the requirement that the water level measures have to be authentic for the administrator when they are displayed at the dam control station. More formally, we get the authenticity requirement:

$$auth(sense(WLS,wl),display(DCS,wl),Admin) \quad (1)$$

*Tiltmeter compromise:* The attacker takes control of the tiltmeter sensors and uses them to send false measurements to the dam control station, thus hiding the real status of the tilt of the dam's walls to the dam administrator. An excessive tilt may lead to the wall's failure. The respective authenticity requirement is:

$$auth(sense(TM,tm),display(DCS,tm),Admin) \quad (2)$$

*Crackmeter / jointmeter compromise:* The attacker has access to one of the crackmeters or jointmeters deployed across the dam's walls and takes control of it. So the attacker can weaken the joint or increase the size of the crack at the wall's weak point without any alarm being raised at the monitoring station, which leads to the following authenticity requirements:

$$auth(sense(CM,cm),display(DCS,cm),Admin) \quad (3)$$
$$auth(sense(JM,jm),display(DCS,jm),Admin) \quad (4)$$

These examples show that some elementary security requirements can be derived directly from misuse cases. In general, however, information flows between systems and components are highly complex, especially when organisational processes need to be considered. Hence, not all security problems are discoverable easily. In order to achieve the desired security goals, security requirements need to be derived systematically.

An important aspect of a systematic security evaluation is the analysis of potential information flows. A method to elicit authenticity requirements by analysis of functional dependencies is described in [17]. From the use case descriptions, atomic actions are derived and set into relation by defining the functional flow among them. The action-oriented approach considers possible sequences of actions (control flow) and information flow (input/output) between interdependent actions. Actions of interest are specifically the *boundary actions*, which represent the interaction of the system's internals with the outside world. From a functional dependency graph, the boundary actions can be identified. We now give an example of security information flows by a use case of the dam scenario [16].

*On demand electric production:* The Dam Control Station feeds an hydroelectric turbine, connected to the dam by means of penstocks, for producing electric power on demand. The turbine and hydroelectric power production depends on the water discharge in the penstocks. By analysing the parameters of the command received by the dam control station, we can infer that the safety critical actions are the opening and closing actions of the penstock gates (*PG*).

An identification of functional dependencies reveals that the dam control activity makes use of the (i) current water level, (ii) the state of the gates joined to the hydroelectric power plant, (iii) the gates openness, and, (iv) the discharge through the penstocks. Figure 1 shows the dependency graph of this use case. The decision of the administrator, which command shall be triggered, depends on the displayed measurements. The dashed line indicates that there is no direct functional dependency.

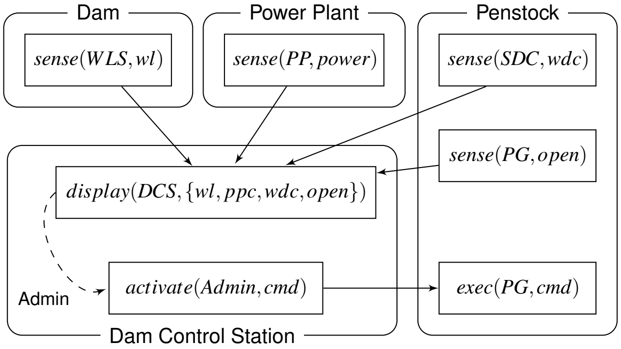


Figure 1. Functional dependencies: *On demand electric production*

An analysis of the dependencies depicted in Figure 1 leads to the following conclusion: The control display values are derived from the measurements of *wl*, *power*, *wdc*, and *open*. From this, we conclude that, in addition to the water level *wl* (1), the measurements of *power*, *wdc*, and *open* have to be authentic. More formally:

$$auth(sense(PP,power),display(DCS,ppc),Admin) \quad (5)$$
$$auth(sense(SDC,wdc),display(DCS,wdc),Admin) \quad (6)$$
$$auth(sense(PG,open),display(DCS,open),Admin) \quad (7)$$

Furthermore, the activation of the penstock command by the administrator has to be authentic for the penstock gate when executing it.

$$auth(activate(Admin,cmd),exec(PG,cmd),PG) \quad (8)$$

So the authenticity requirements for the use case described in Figure 1 are given by: (1) and (5)–(8).

In summary, the analysis of the use case and misuse cases of this critical infrastructure scenario shows that the overall function of the system requires authenticity of measurement values for several sensors, namely (1) – (7). In that sense, the dam scenario is a prime example for the relevance of the requirements postulated in Proposition 1 and 2. It is evident that further types of security requirements are needed in order to cover important liveness properties such as *availability* of necessary information at a certain place and time. In some cases also *confidentiality* of certain information may be required. These requirements are important but not in the scope of the work presented here.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-184-7

## V. Trusted Information Agent

The usefulness of monitoring large systems clearly depends on the observer's level of confidence in the correctness of the available monitoring data. In order to achieve that confidence, network security measures and provisions against technical faults are not enough. As stated above, unrevealed manipulation of monitoring equipment can lead to serious consequences. In order to improve the coverage of this type of requirements in a SIEM framework, we now describe a concept and a prototypical implementation of a trusted information agent (TIA).

### A. Trust Anchor and Architecture

As shown in Section IV, protection of the identity of the device for measurement collection is necessary. Furthermore, the lack of control on the physical access to the sensor node induces strong requirements on the protection level.

By a suitable combination of hardware- and software-level protection techniques any manipulations of a sensor have to be revealed. In addition to the node-level protection, network security measures are needed in order to achieve specification-conformant behaviour of the sensor network, e.g., secure communication channels that protect data against tampering. This paper is not intended to discuss network security, neither protection of hardware components. We rather concentrate on the important problem of clandestine manipulations of the sensor software.

A commonly used technique to reveal manipulation of a software component is software measurement: Each component is considered as a byte sequence and thus can be measured by computing a hash value, which is subsequently compared to the component's reference value. The component is authentic, if and only if both values are identical. Obviously, such measurements make no sense if the measuring component or the reference values are manipulated themselves. A common solution is to establish a chain of trust: In a layered architecture, each layer is responsible for computing the checksums of the components in the next upper layer. At the very bottom of this chain a dedicated security hardware chip takes the role of the trust anchor or “root of trust”.

Trusted Computing [6] offers such a hardware root of trust providing certain security functionalities, which can be used to reveal malicious manipulations of the sensors in the field. Trusted Computing technology standards provide methods for reliably checking a system's integrity and identifying anomalous and/or unwanted characteristics. A trusted system in this sense is build on top of a Trusted Platform Module (TPM) as specified by the Trusted Computing Group (TCG). A TPM is hardened against physical attacks and equipped with several cryptographic capabilities like strong encryption and digital signatures. TPMs have been proven to be much less susceptible to attacks than corresponding software-only solutions.

The key concept of Trusted Computing is the extension of trust from the TPM to further system components. This concept is commonly used to ensure that a system is and remains in a predictable and trustworthy state and thus produces authentic results. As described above, each layer of the chain checks the integrity of the next upper layer's programs, libraries, etc. On a PC, for example, the TPM has to check the BIOS before giving the control of the boot-process to it. The BIOS then has to verify the operating system kernel, which in turn is responsible for the measurement of the next level. Actually, a reliable and practically useful implementation for PCs and systems of similar or higher complexity is not yet feasible. Sensoring and measuring devices, however, typically have a considerably more primitive architecture than PCs and are well-suited for this kind of integrity check concept. Even for modern sensor-equipped smartphones, able to act as event detectors, but having the same magnitude of computing power that PCs had a few years ago, an implementation of the presented concept is possible. A prototypical implementation is presented in more detail now.

### B. Proof of Concept: Base Measure Aquisition

Figure 2 depicts the architecture of the TIA.

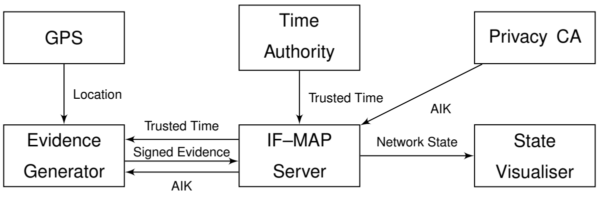


Figure 2. TIA architecture

The main component of the TIA is the *evidence generator* (EG), which collects base measures and provides the measurement functions used to produce derived measures. Furthermore, the EG supports the processing of measures from external sensors, e.g., location data from a GPS module. The EG is expected to operate in unprotected environments with low physical protection and externally accessible interfaces such as wireless networks and USB access for maintenance. A necessary precondition to guarantee authenticity of the measures, is a trustworthy state of the measurement device. To meet this requirement, the EG is equipped with a TPM as trust anchor and implements a chain of trust [18]. As explained above, revelation of software manipulations is based on the comparison between the software checksums and the corresponding reference values. This comparison may be done locally within the node (self-attestation) or by a remote verifier component (remote attestation) [6].

The EG submits the collected measures digitally signed to an IF-MAP [14] server, which acts as an event information broker. During initialisation, the EG obtains two credentials

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-184-7

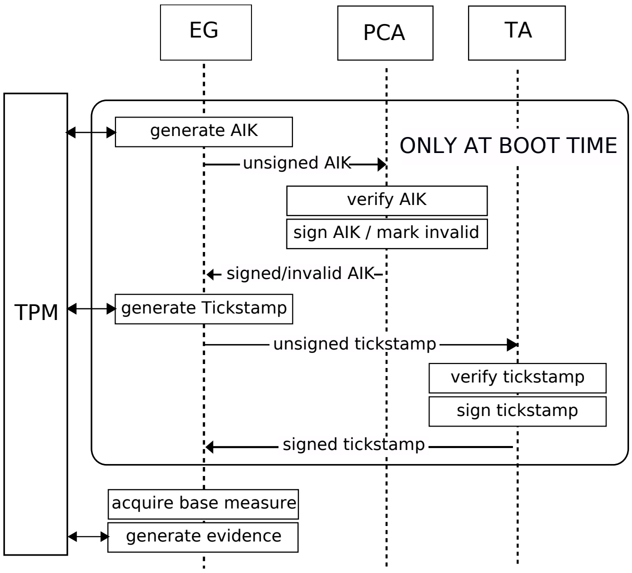


Figure 3. Process model

from trusted third-party services for signature purposes. Figure 3 depicts the boot-time interaction between the EG and those services, and the role of the TPM in this interaction.

An Attestation Identity Key (AIK) is used to sign measurement results in a manner that allows verification by a remote party. The Privacy Certification Authority (PCA) issues a credential for the TPM-generated AIK. The certified AIK is, henceforth, used as an identity for this platform. According to TCG standards, AIKs cannot only be used to attest origin and authenticity of a trust measurement, but also, to authenticate other keys and data generated by the TPM. However, the AIK functionality of a TPM is designed primarily to support remote attestation by signing the checksums of the EG's software components, while signing arbitrary data is, in fact, not directly available as a TPM operation. We have shown elsewhere, how to circumvent this limitation [19]. Hence, we are able to use TPM-signatures for arbitrary data from the EG's sensors.

Any TPM is equipped with an accurate timer. Each event signature includes the current timer value. However, the TPM timer is a relative counter, not associated to an absolute time. A *time authority* (TA) issues a certificate about the correspondence between a TPM timestamp (tickstamp) and the absolute time. The combination of tickstamp and TA-certificate can be used as a trusted timestamp. Alternatively, another trusted time source, such as GPS, could have been used.

Putting it all together, a measurement record includes arbitrary sensor data, a TA-certified time stamp, and a hash value of the EG's software components. The record itself is signed by the TA-certified AIK.

Figure 4 shows a prototype EG, which has been implemented based on the Android smartphone platform. This platform has been selected for various reasons. Modern smartphones are equipped with a variety of sensors such as GPS, gyro sensor, electronic compass, proximity sensor, accelerometer, barometer, and ambient light sensor. Furthermore, photos, video and sound can be regarded and processed as event data. Moreover, Android is well-suited as a software platform for future embedded devices.

The TPM-anchored chain of trust is extended to the linux system and linux application layers by using the Integrity Measurement Architecture (IMA), which is integrated into any stock linux kernel as a kernel module. The Android application layer is based on libraries and the Dalvik Virtual Machine (VM). While the linux kernel layer can check the Android system libraries and the VM, Android applications run on top of the VM and are invisible to the kernel. Thus, we built a modified VM, which extends the chain of trust to the Android application level by computing the applications' checksums. A timestamp-based variant of remote attestation provided by the TPM is used for the verification of the node authenticity. All communication is based on the Trusted Network Connect (TNC) [14] protocol suite, which offers advanced security features, such as dedicated access control mechanisms for TPM-equipped nodes.

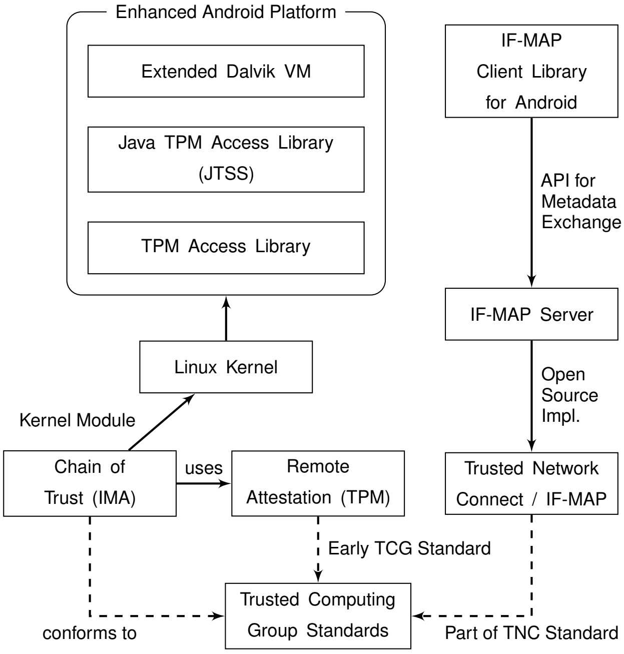


Figure 4. Technical building blocks

## VI. Conclusion and Future Work

In geographically dispersed infrastructures the critical sources of event data are often placed in non-protected environments. Attackers can thus easily manipulate these sensors and thereby hide critical conditions, generate false alerts, and in general cause misjudgement on system's state. By exemplary analysis of a typical application scenario we have demonstrated that this can lead to false decisions with severe impact on the overall system. In order to prevent such threats, we presented a concept for holistically protected critical event sources by assuring a trustworthy state of the measurement devices. This enables better assessment of the managed system's reliability and trustworthiness.

As a proof of this concept, the paper presented an exemplary realisation of a trusted information agent based on trusted computing technology. Planned next steps include a detailed analysis on the impact on scalability and bandwidth of different schemes to generate evidence using this architecture. Especially, the correlation of independent events may allow for improvements but also requires trustworthy schemes to cryptographically link various events to one evidence record. Also, the hardware-based security functionalities can be improved with respect to scalability and performance. Further, suggestions to improve standards for future hardware security modules, are planned.

## Acknowledgement

Luigi Coppolino and Roland Rieke developed the work presented here in the context of the project MASSIF (ID 257475) being co-funded by the European Commission within FP7. Nicolai Kuntze developed the work presented here in the context of the project ESUKOM (ID 01BY1052) which is funded by the German Federal Ministry of Education and Research.

## References

[1] "Project MASSIF website," 2012. [Online]. Available: http://www.massif-project.eu/

[2] E. Prieto, R. Diaz, L. Romano, R. Rieke, and M. Achemlal, "MASSIF: A promising solution to enhance olympic games IT security," in *International Conference on Global Security, Safety and Sustainability (ICGS3 2011)*, 2011.

[3] L. Coppolino, S. D'Antonio, V. Formicola, and L. Romano, "Integration of a System for Critical Infrastructure Protection with the OSSIM SIEM Platform: A dam case study," in *SAFECOMP*, ser. Lecture Notes in Computer Science, F. Flammini, S. Bologna, and V. Vittorini, Eds., vol. 6894. Springer, 2011, pp. 199–212.

[4] B. Zhu, A. Joseph, and S. Sastry, "Taxonomy of Cyber Attacks on SCADA Systems," in *Proceedings of CPSCom 2011: The 4th IEEE International Conference on Cyber, Physical and Social Computing, Dalian, China*, 2011.

[5] J. Choi, I. Shin, J. Seo, and C. Lee, "An efficient message authentication for non-repudiation of the smart metering service," *Computers, Networks, Systems and Industrial Engineering, ACIS/JNU International Conference on*, vol. 0, pp. 331–333, 2011.

[6] C. Mitchell, *Trusted Computing*. Iet, 2005.

[7] M. Nicolett and K. M. Kavanagh, "Magic Quadrant for Security Information and Event Management," Gartner Reasearch, May 2010.

[8] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "Authenticity and provability - a formal framework," in *Infrastructure Security Conference InfraSec 2002*, ser. LNCS, vol. 2437. Springer, 2002, pp. 227–245.

[9] M. Parekh, K. Stone, and J. Delborne, "Coordinating intelligent and continuous performance monitoring with dam and levee safety management policy," in *Association of State Dam Safety Officials,Proceedings of Dam Safety Conference 2010*, 2010.

[10] B. K. Myers, G. C. Dutson, and T. Sherman, "Utilizing Automated Monitoring for the Franzen Reservoir Dam Safety Program," in *25th USSD Annual Meeting and Conference Proceedings (2005)*.

[11] L. Coppolino, S. D'Antonio, and L. Romano, "Dependability and resilience of computer networks (scada cybersecurity)," in *CRITICAL INFRASTRUCTURE SECURITY: Assessment, Prevention, Detection, Response*. WIT press, in press.

[12] L. Coppolino, S. D'Antonio, L. Romano, and G. Spagnuolo, "An intrusion detection system for critical information infrastructures using wireless sensor network technologies," in *Critical Infrastructure (CRIS), 2010 5th International Conference on*, sept. 2010, pp. 1 –8.

[13] J. Richter, N. Kuntze, and C. Rudolph, "Security Digital Evidence," in *2010 Fifth International Workshop on Systematic Approaches to Digital Forensic Engineering*. IEEE, 2010, pp. 119–130.

[14] T. C. Group, "TCG Trusted Network Connect – TNC IF-MAP Binding for SOAP Version 2.0," www.trustedcomputing.org, 2010.

[15] J. v. H. I. Bente, J. Vieweg, "Towards Trustworthy Networks with Open Source Software," in *Horizons in Computer Science Volume 3*. Nova Science Publishers Inc., T. S. Clary (Eds.), 2011.

[16] M. Llanes, E. Prieto, R. Diaz, , L. Coppolino, A. Sergio, R. Cristaldi, M. Achemlal, S. Gharout, C. Gaber, A. Hutchison, and K. Dennie, "Scenario requirements (public version)," MASSIF Project, Tech. Rep. Deliverable D2.1.1, 2011.

[17] A. Fuchs and R. Rieke, "Identification of Security Requirements in Systems of Systems by Functional Security Analysis," in *Architecting Dependable Systems VII*, ser. LNCS. Springer, 2010, vol. 6420, pp. 74–96.

[18] N. Kuntze and C. Rudolph, "Secure digital chains of evidence," in *Sixth International Workshop on Systematic Approaches to Digital Forensic Engeneering*, 2011.

[19] N. Kuntze, D. Mähler, and A. U. Schmidt, "Employing trusted computing for the forward pricing of pseudonyms in reputation systems," in *Axmedis 2006, Proceedings of the 2nd International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution, Volume for Workshops, Industrial, and Application Sessions*, 2006.

# P7

# SECURITY PROPERTIES OF SELF-SIMILAR UNIFORMLY PARAMETERISED SYSTEMS OF COOPERATIONS

| | |
|---|---|
| **Title** | Security Properties of Self-similar Uniformly Parameterised Systems of Cooperations |
| **Authors** | Peter Ochsenschläger and Roland Rieke |
| **Publication** | In *Proceedings of the 19th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 640–645. |
| **ISBN/ISSN** | ISSN 1066-6192 |
| **DOI** | `http://dx.doi.org/10.1109/PDP.2011.57` |
| **Status** | Published |
| **Publisher** | IEEE Computer Society, Los Alamitos, CA, USA |
| **Publication Type** | Conference Proceedings |
| **Copyright** | 2011, IEEE |
| **Contribution of Roland Rieke** | Co-Author with significant contribution, editor, and presenter at the special session on "Security in Networked and Distributed Systems" at the 19th Euromicro Conference on Parallel, Distributed and Network-Based Processing.<br>Specific contributions are: (1) analysis of the examples given in the paper in the SHVT; (2) experiments with bigger examples to assure that the abstraction concept scales. |

Table 12: Fact Sheet Publication *P7*

Publication P7 [Ochsenschläger & Rieke, 2011] addresses the following research question:

RQ4 *Which design principles facilitate verifiability of security properties of scalable systems?*

In this paper uniform parameterisations of cooperations are defined in terms of formal language theory, such that each pair of partners cooperates in the same manner, and that the mechanism (schedule) to determine how one partner may be involved in several cooperations, is the same for each partner. Generalising each pair of partners cooperating in the same manner, for such systems of cooperations a kind of self-similarity is formalised. From an abstracting point of view, where only actions of some selected partners are considered, the complex system of all partners behaves like the smaller subsystem of the selected partners. For verification purposes, so called uniformly parameterised safety properties are defined. Such properties can be used to express privacy policies as well as security and dependability requirements. It is shown, how the parameterised problem of verifying such a property is reduced by self-similarity to a finite state problem.

# Security Properties of Self-similar Uniformly Parameterised Systems of Cooperations

Peter Ochsenschläger and Roland Rieke
*Fraunhofer Institute for Secure Information Technology, SIT*
*Darmstadt, Germany*
*Email: peter-ochsenschlaeger@t-online.de, roland.rieke@sit.fraunhofer.de*

***Abstract*—Uniform parameterisations of cooperations are defined in terms of formal language theory, such that each pair of partners cooperates in the same manner, and that the mechanism (schedule) to determine how one partner may be involved in several cooperations, is the same for each partner. Generalising each pair of partners cooperating in the same manner, for such systems of cooperations a kind of self-similarity is formalised. From an abstracting point of view, where only actions of some selected partners are considered, the complex system of all partners behaves like the smaller subsystem of the selected partners. For verification purposes, so called uniformly parameterised safety properties are defined. Such properties can be used to express privacy policies as well as security and dependability requirements. It is shown, how the parameterised problem of verifying such a property is reduced by self-similarity to a finite state problem.**

***Keywords*-cooperations as prefix closed languages; abstractions of system behaviour; self-similarity in systems of cooperations; privacy policies; uniformly parameterised safety properties;**

## I. Introduction

As an example for cooperations let us consider an e-commerce protocol, that determines how two cooperation partners have to perform a certain kind of financial transactions. As such a protocol should work for several partners in the same manner, it is parameterised by the partners and the parameterisation should be uniform w.r.t. the partners. It is quite evident that similar requirements have to be fulfilled in any highly scalable system or system of systems such as cloud computing platforms or vehicular communication systems in which vehicles and roadside units communicate in ad hoc manner to exchange traffic information [1].

In this paper (Sect. III) we formalise uniform parameterisations of two-sided cooperations in terms of formal language theory, such that each pair of partners cooperates in the same manner, and that the mechanism (schedule) to determine how one partner may be involved in several cooperations, is the same for each partner. Generalising each pair of partners cooperating in the same manner, the following kind of self-similarity is desirable for such systems of cooperations: From an abstracting point of view, where only actions of some selected partners are considered, the complex system of all partners behaves like the smaller subsystem of the selected partners (Sect. IV).

For verification purposes it is of interest to know, which kind of dynamic system properties are “compatible” with self-similarity. Therefore in Sect. V so called uniformly parameterised safety properties are defined. An example shows how such properties can be used to express privacy policies. Subsequently, it is shown how the parameterised problem of verifying such a property is reduced by self-similarity to a finite state problem under certain regularity restrictions. Liveness aspects of self-similar systems will be subject of a forthcoming paper (see Sect. VI).

## II. Related Work

*Verification approaches for parameterised systems.:* An extension to the *Mur$\varphi$* verifier to verify systems with replicated identical components through a new data type called RepetitiveID is presented in [2]. During the verification Mur$\varphi$ checks if the result can be generalised for a family of systems. The soundness of the abstraction algorithm is guaranteed by the restrictions on the use of repetitiveIDs. A typical application area of this tool are cache coherence protocols. The aim of [3] is an abstraction method through symmetry which works also when using variables holding references to other processes which is not possible in Mur$\varphi$. An implementation of this approach for the *SPIN* model-checker (http://spinroot.com/) is described. In [4] a methodology for constructing abstractions and refining them by analysing counter-examples is presented. The method combines abstraction, model-checking and deductive verification and in particular, allows to use the set of reachable states of the abstract system in a deductive proof even when the abstract model does not satisfy the specification and when it simulates the concrete system with respect to a weaker simulation notion than Milner’s. The tool *InVeSt* supports this approach and makes use of *PVS* (http://pvs.csl.sri.com/) and *SMV* (http://www.cs.cmu.edu/~modelcheck/smv.html). This approach however does not consider liveness properties. In [5] a technique for automatic verification of parameterised systems based on process algebra *CCS* [6] and the logic modal *mu-calculus* [7] is presented. This technique views processes as property transformers and is based on computing the limit of a sequence of mu-calculus formula generated by these transformers. In [8] we developed an *abstraction based approach* to extend our tool supported verification techniques to be able to verify families of parameterised systems, independent of the exact number of replicated components. The above-mentioned approaches demonstrate, that finite state methods combined with deductive methods can be applied to analyse parameterised systems. The approaches differ in varying amounts of user intervention and their range of application. A survey of approaches to combine model checking and theorem proving methods is given in [9].

*Iterated shuffle products.:* In [10] it is shown that our definition of uniformly parameterised cooperations is strongly related to iterated shuffle products [11], if the cooperations are "structured into phases". For such systems of cooperations a sufficient condition for the kind of self-similarity which we use here is given. Under certain regularity restrictions this condition can be verified by a semi-algorithm. The main concept for such a condition are shuffle automata [12] (multicounter automata [13]) whose computations, if they are deterministic, unambiguously describe how a cooperation partner is involved in several phases.

The main contribution of this paper is to show how the parameterised problem of verifying a uniformly parameterised safety property can be solved by means of the self-similarity results of [10] and finite state methods.

## III. Parameterised Cooperations

The behaviour $L$ of a discrete system can be formally described by the set of its possible sequences of actions. Therefore $L \subset \Sigma^*$ holds where $\Sigma$ is the set of all actions of the system, and $\Sigma^*$ (free monoid over $\Sigma$) is the set of all finite sequences of elements of $\Sigma$ (words), including the empty sequence denoted by $\varepsilon$. Subsets of $\Sigma^*$ are called formal languages. Words can be composed: if $u$ and $v$ are words, then $uv$ is also a word. This operation is called the *concatenation*; especially $\varepsilon u = u\varepsilon = u$. A word $u$ is called a *prefix* of a word $v$ if there is a word $x$ such that $v = ux$. The set of all prefixes of a word $u$ is denoted by $\mathrm{pre}(u)$; $\varepsilon \in \mathrm{pre}(u)$ holds for every word $u$.

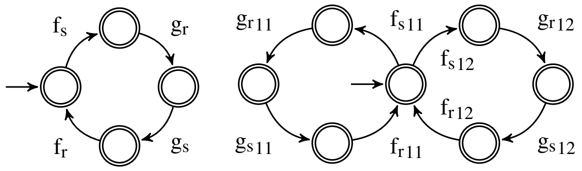
Formal languages which describe system behaviour have the characteristic that $\mathrm{pre}(u) \subset L$ holds for every word $u \in L$. Such languages are called *prefix closed*. System behaviour is thus described by prefix closed formal languages.

Different formal models of the same system are partially ordered with respect to different levels of abstraction. Formally, abstractions are described by so called alphabetic language homomorphisms. These are mappings $h^* : \Sigma^* \longrightarrow \Sigma'^*$ with $h^*(xy) = h^*(x)h^*(y)$ , $h^*(\varepsilon) = \varepsilon$ and $h^*(\Sigma) \subset \Sigma' \cup \{\varepsilon\}$. So they are uniquely defined by corresponding mappings $h : \Sigma \longrightarrow \Sigma' \cup \{\varepsilon\}$. In the following we denote both the mapping $h$ and the homomorphism $h^*$ by $h$. Inverse homomorphism are denoted by $h^{-1}$. Let $L$ be a language over the alphabet $\Sigma'$. Then $h^{-1}(L)$ is the set of words $w \in \Sigma^*$ such that $h(w) \in L$. In this paper we consider a lot of alphabetic language homomorphisms. So for simplicity we tacitly assume that a mapping between free monoids is an alphabetic language homomorphism if nothing contrary is stated.

To describe a two-sided cooperation, let $\Sigma = \Phi \mathbin{\dot\cup} \Gamma$ where $\Phi$ is the set of actions of cooperation partner $F$ and $\Gamma$ is the set of actions of cooperation partner $G$. Now a prefix closed language $L \subset (\Phi \mathbin{\dot\cup} \Gamma)^*$ formally defines a two-sided cooperation.

**Example 1.** *Let* $\Phi = \{\mathrm{f_s}, \mathrm{f_r}\}$ *and* $\Gamma = \{\mathrm{g_s}, \mathrm{g_r}\}$ *and hence* $\Sigma = \{\mathrm{f_s}, \mathrm{f_r}, \mathrm{g_s}, \mathrm{g_r}\}$. *An example for a cooperation* $L \subset \Sigma^*$ *is now given by the automaton in Fig. 1(a). It describes a simple handshake between F and G.*

Please note that in the following we will denote initial states by a short incoming arrow and final states by double circles. In this automaton all states are final states, since $L$ is prefix closed.



(a) 1-1-cooperation $L$ (b) 1-2-cooperation $\mathscr{L}_{\{1\}\{1,2\}}$

Figure 1. Automata for a simple parameterised cooperation

For parameter sets $I$, $K$ and $(i,k) \in I \times K$ let $\Sigma_{ik}$ denote pairwise disjoint copies of $\Sigma$. The elements of $\Sigma_{ik}$ are denoted by $a_{ik}$ and $\Sigma_{IK} := \dot{\bigcup}_{(i,k)\in I\times K} \Sigma_{ik}$. The index $ik$ describes the bijection $a \leftrightarrow a_{ik}$ for $a \in \Sigma$ and $a_{ik} \in \Sigma_{ik}$. Now $\mathscr{L}_{IK} \subset \Sigma_{IK}^*$ (prefix-closed) describes a *parameterised system*. To avoid pathological cases we generally assume parameter and index sets to be non empty.

For a cooperation between one partner of type $F$ with two partners of type $G$ in Example 1 let $\Phi_{\{1\}\{1,2\}} = \{\mathrm{f_{s11}}, \mathrm{f_{r11}}, \mathrm{f_{s12}}, \mathrm{f_{r12}}\}$, $\Gamma_{\{1\}\{1,2\}} = \{\mathrm{g_{s11}}, \mathrm{g_{r11}}, \mathrm{g_{s12}}, \mathrm{g_{r12}}\}$ and $\Sigma_{\{1\}\{1,2\}} = \Phi_{\{1\}\{1,2\}} \mathbin{\dot\cup} \Gamma_{\{1\}\{1,2\}}$. A 1-2-cooperation, where each pair of partners cooperates restricted by $L$ and each partner has to finish the handshake it just is involved in before entering a new one, is now given (by reachability analysis) by the automaton in Fig. 1(b) for $\mathscr{L}_{\{1\}\{1,2\}}$. Fig. 2 in contrast depicts an automaton for a 2-1-cooperation $\mathscr{L}_{\{1,2\}\{1\}}$ with the same overall number of partners involved but two of type $F$ and one partner of type $G$. A 3-3-cooperation with the same simple behaviour of partners already requires an automaton with 916 states and 3168 state transitions.

For $(i,k) \in I \times K$, let $\pi_{ik}^{IK} : \Sigma_{IK}^* \to \Sigma^*$ with

$$\pi_{ik}^{IK}(a_{rs}) = \begin{cases} a \mid & a_{rs} \in \Sigma_{ik} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Sigma_{ik} \end{cases} .$$

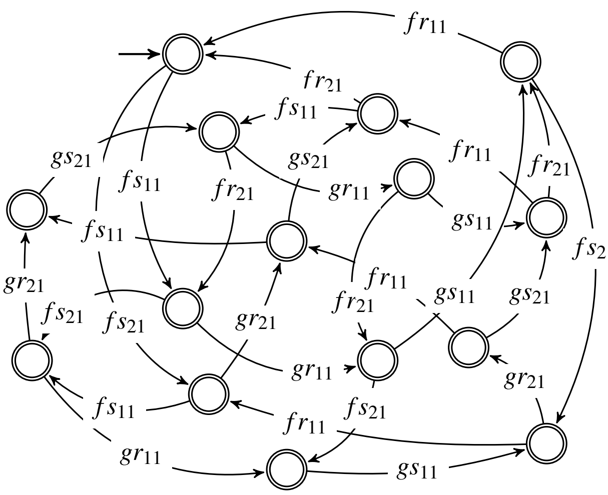


Figure 2. Automaton for the 2-1-cooperation $\mathscr{L}_{\{1,2\}\{1\}}$

For *uniformly parameterised systems* $\mathscr{L}_{IK}$ we generally want to have

$$\mathscr{L}_{IK} \subset \bigcap_{(i,k)\in I\times K} ((\pi_{ik}^{IK})^{-1}(L))$$

because from an abstracting point of view, where only the actions of a specific $\Sigma_{ik}$ are considered, the complex system $\mathscr{L}_{IK}$ is restricted by $L$.

In addition to this inclusion $\mathscr{L}_{IK}$ is defined by *local schedules* that determine how each "version of a partner" can participate in "different cooperations". More precisely, let $SF \subset \Phi^*$, $SG \subset \Gamma^*$ be prefix closed. For $(i,k) \in I \times K$, let $\varphi_i^{IK} : \Sigma_{IK}^* \to \Phi^*$ and $\gamma_k^{IK} : \Sigma_{IK}^* \to \Gamma^*$ with

$$\varphi_i^{IK}(a_{rs}) = \begin{cases} a \mid & a_{rs} \in \Phi_{\{i\}K} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Phi_{\{i\}K} \end{cases} \text{ and}$$

$$\gamma_k^{IK}(a_{rs}) = \begin{cases} a \mid & a_{rs} \in \Gamma_{I\{k\}} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Gamma_{I\{k\}} \end{cases},$$

where $\Phi_{IK}$ and $\Gamma_{IK}$ are defined correspondingly to $\Sigma_{IK}$.

**Definition 1** (Uniformly parameterised cooperation $\mathscr{L}_{IK}$). *Let $I$, $K$ be finite parameter sets, then*

$$\mathscr{L}_{IK} := \bigcap_{(i,k)\in I\times K} (\pi_{ik}^{IK})^{-1}(L) \cap$$
$$\cap \bigcap_{i\in I} (\varphi_i^{IK})^{-1}(SF) \cap \bigcap_{k\in K} (\gamma_k^{IK})^{-1}(SG)$$

By this definition

$$\mathscr{L}_{\{1\}\{1\}} = (\pi_{11}^{\{1\}\{1\}})^{-1}(L) \cap$$
$$\cap\ (\varphi_1^{\{1\}\{1\}})^{-1}(SF) \cap (\gamma_1^{\{1\}\{1\}})^{-1}(SG).$$

As we want $\mathscr{L}_{\{1\}\{1\}}$ being isomorphic to $L$ by the isomorphism $\pi_{11}^{\{1\}\{1\}} : \Sigma_{\{1\}\{1\}}^* \to \Sigma^*$ we additionally need

$$(\pi_{11}^{\{1\}\{1\}})^{-1}(L) \subset (\varphi_1^{\{1\}\{1\}})^{-1}(SF) \text{ and}$$
$$(\pi_{11}^{\{1\}\{1\}})^{-1}(L) \subset (\gamma_1^{\{1\}\{1\}})^{-1}(SG).$$

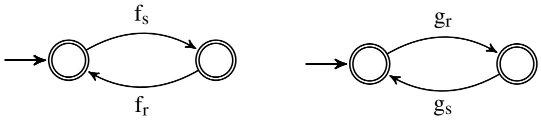
This is equivalent to $\pi_\Phi(L) \subset SF$ and $\pi_\Gamma(L) \subset SG$, where $\pi_\Phi : \Sigma^* \to \Phi^*$ and $\pi_\Gamma : \Sigma^* \to \Gamma^*$ are defined by

$$\pi_\Phi(a) = \begin{cases} a \mid & a \in \Phi \\ \varepsilon \mid & a \in \Gamma \end{cases} \text{ and } \pi_\Gamma(a) = \begin{cases} a \mid & a \in \Gamma \\ \varepsilon \mid & a \in \Phi \end{cases}.$$

So we complete Def. 1 by the additional conditions

$$\pi_\Phi(L) \subset SF \text{ and } \pi_\Gamma(L) \subset SG.$$

Schedules $SF$ and $SG$ that fit to the cooperations given in Example 1 are depicted in Figs. 3(a) and 3(b). Here we have $\pi_\Phi(L) = SF$ and $\pi_\Gamma(L) = SG$.



(a) Schedule $SF$ (b) Schedule $SG$

Figure 3. Automata $\mathbb{SF}$ and $\mathbb{SG}$ for the schedules $SF$ and $SG$

The system $\mathscr{L}_{IK}$ of cooperations is a typical example of a *complex system*. It consists of several identical components (copies of the two-sided cooperation $L$), which "interact" in a uniform manner (described by the schedules $SF$ and $SG$ and by the homomorphisms $\varphi_i^{IK}$ and $\gamma_k^{IK}$).

**Remark 1.** *It is easy to see that $\mathscr{L}_{IK}$ is isomorphic to $\mathscr{L}_{I'K'}$ if $I$ is isomorphic to $I'$ and $K$ is isomorphic to $K'$. More precisely, let $\iota_{I'}^{I} : I \to I'$ and $\iota_{K'}^{K} : K \to K'$ be bijections and let $\iota_{I'K'}^{IK} : \Sigma_{IK}^* \to \Sigma_{I'K'}^*$ be defined by*

$$\iota_{I'K'}^{IK}(a_{ik}) := a_{\iota_{I'}^{I}(i)\iota_{K'}^{K}(k)} \textit{ for } a_{ik} \in \Sigma_{IK}.$$

*Then $\iota_{I'K'}^{IK}$ is a isomorphism and $\iota_{I'K'}^{IK}(\mathscr{L}_{IK}) = \mathscr{L}_{I'K'}$. The set of all these isomorphisms $\iota_{I'K'}^{IK}$ defined by corresponding bijections $\iota_{I'}^{I}$ and $\iota_{K'}^{K}$ is denoted by $\mathscr{I}_{I'K'}^{IK}$.*

## IV. Self-Similarity

By *self-similary* we want to formalise that for $I' \subset I$ and $K' \subset K$ from an abstracting point of view, where only the actions of $\Sigma_{I'K'}$ are considered, the complex system $\mathscr{L}_{IK}$ behaves like the smaller subsystem $\mathscr{L}_{I'K'}$. Therefore we now consider special abstractions on $\mathscr{L}_{IK}$.

**Definition 2** (Projection abstraction). *For $I' \subset I$ and $K' \subset K$ let $\Pi_{I'K'}^{IK} : \Sigma_{IK}^* \to \Sigma_{I'K'}^*$ with*

$$\Pi_{I'K'}^{IK}(a_{rs}) = \begin{cases} a_{rs} \mid & a_{rs} \in \Sigma_{I'K'} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Sigma_{I'K'}. \end{cases}$$

It is easy to see [10]:

**Theorem 1.** *$\mathscr{L}_{IK} \supset \mathscr{L}_{I'K'}$ for $I' \times K' \subset I \times K$, and therefore*

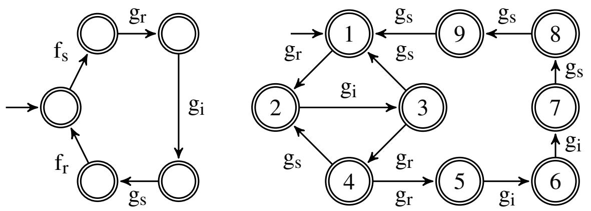
$$\Pi_{I'K'}^{IK}(\mathscr{L}_{IK}) \supset \Pi_{I'K'}^{IK}(\mathscr{L}_{I'K'}) = \mathscr{L}_{I'K'}.$$

The reverse inclusions

$$\Pi^{IK}_{I'K'}(\mathscr{L}_{IK}) \subset \mathscr{L}_{I'K'} \text{ for all } I' \times K' \subset I \times K \qquad (1)$$

do not hold in general, which is shown by the following example.

**Example 2.** *For a counterexample let us examine the 1-1-cooperation given by the automaton in Fig. 4(a). Let the schedule SF again be given by the automaton* $\mathbb{SF}$ *in Fig. 3(a) and the schedule SG be given by the automaton* $\mathbb{SG}$ *in Fig. 4(b).*



(a) 1-1-cooperation $L$ (b) Schedule $SG$

Figure 4. Automata for the counterexample

*In the automaton* $\mathbb{SG}$ *immediately after entering a second handshake (state 4) G may enter a third handshake but immediately after entering the first handshake (state 2) G may not enter a second handshake. We now get for example*

$$\mathrm{f}_{\mathrm{s}11}\mathrm{f}_{\mathrm{s}21}\mathrm{f}_{\mathrm{s}31}\mathrm{g}_{\mathrm{r}11}\mathrm{g}_{\mathrm{i}11}\mathrm{g}_{\mathrm{r}21}\mathrm{g}_{\mathrm{r}31} \in \mathscr{L}_{\{1,2,3\}\{1\}}.$$

*Hence*

$$\mathrm{f}_{\mathrm{s}21}\mathrm{f}_{\mathrm{s}31}\mathrm{g}_{\mathrm{r}21}\mathrm{g}_{\mathrm{r}31} \in \Pi^{\{1,2,3\}\{1\}}_{\{2,3\}\{1\}}(\mathscr{L}_{\{1,2,3\}\{1\}}), \textit{ but}$$

$$\mathrm{f}_{\mathrm{s}21}\mathrm{f}_{\mathrm{s}31}\mathrm{g}_{\mathrm{r}21}\mathrm{g}_{\mathrm{r}31} \notin \mathscr{L}_{\{2,3\}\{1\}}.$$

In the general case we do not know the decidability status of (1), but for many parameterised systems (1) holds, and therefore

$$\Pi^{IK}_{I'K'}(\mathscr{L}_{IK}) = \mathscr{L}_{I'K'},$$

which is a generalisation of $\pi^{IK}_{ik}(\mathscr{L}_{IK}) = L$.

**Definition 3** (Self-similarity)**.**
*A uniformly parameterised cooperation* $\mathscr{L}_{IK}$ *is called* self-similar *iff*

$$\Pi^{IK}_{I'K'}(\mathscr{L}_{IK}) = \mathscr{L}_{I'K'} \textit{ for each } I' \times K' \subset I \times K.$$

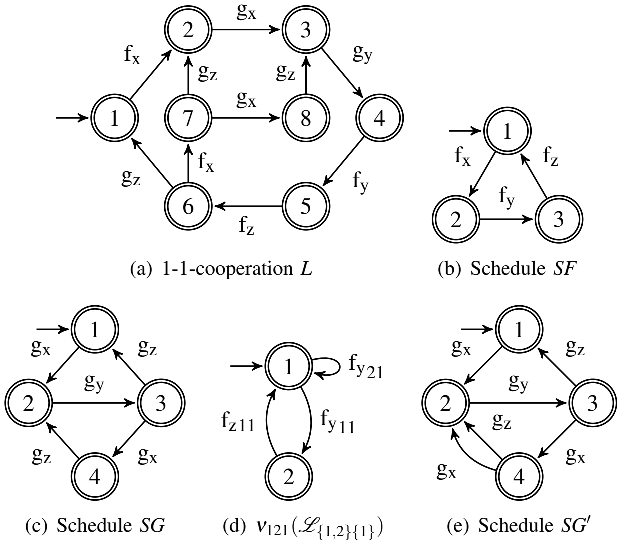
So we are looking for conditions, which imply (1). Fig. 4(b) is typical in the sense that it may serve as an idea to get a sufficient condition for self-similarity. It requires (a) two separate conditions, one for each schedule, (b) structuring schedules into phases, which may be shuffled in a restricted manner, (c) formalising "*how a cooperation partner is involved in several phases*", (d) the more phases a cooperation partner is involved in, the less possibilities of acting in each phase he has. In [10] a sufficient condition for self-similarity is given, which is based on deterministic computations in shuffle automata. Under certain regularity restrictions this condition can be verified by a semi-algorithm.

## V. Uniformly Parameterised Security Properties

We will now give an example that demonstrates the significance of self-similarity for verification purposes and then present a generic verification scheme for uniformly parameterised security properties.

**Example 3.** *We consider a system of servers, each of them managing a resource, and clients, which want to use these resources. We assume that as a means to enforce a given* privacy policy *a server has to manage its resource in such a way that no client may access this resource during it is in use by another client (*privacy requirement*). This may be required to ensure* anonymity *in such a way that clients and their actions on a resource cannot be linked by an observer.*

We formalise this system at an abstract level, where a client may perform the actions $\mathrm{f}_\mathrm{x}$ (send a request), $\mathrm{f}_\mathrm{y}$ (receive a permission) and $\mathrm{f}_\mathrm{z}$ (send a free-message), and a server may perform the corresponding actions $\mathrm{g}_\mathrm{x}$ (receive a request), $\mathrm{g}_\mathrm{y}$ (send a permission) and $\mathrm{g}_\mathrm{z}$ (receive a free-message). The possible sequences of actions of a client resp. of a server are given by the automaton $\mathbb{SF}$ resp. $\mathbb{SG}$. The automaton $\mathbb{L}$ describes the 1-1-cooperation of one client and one server (see Fig. 5). These automata define the client-server system $\mathscr{L}_{IK}$.



(a) 1-1-cooperation $L$ (b) Schedule $SF$

(c) Schedule $SG$ (d) $\nu_{121}(\mathscr{L}_{\{1,2\}\{1\}})$ (e) Schedule $SG'$

Figure 5. Automata $\mathbb{L}$, $\mathbb{SF}$, $\mathbb{SG}$, $\nu_{121}(\mathscr{L}_{\{1,2\}\{1\}})$ and $\mathbb{SG}'$ for Example 3

Considering $\mathrm{f}_\mathrm{y}$ as the begin-action and $\mathrm{f}_\mathrm{z}$ as the end-action w.r.t. accessing a resource, the privacy requirement can be formalised by (2).

Let $i,i' \in I$, $i \neq i'$, $k \in K$ and $\mu^{IK}_{ii'k} : \Sigma^*_{IK} \to \{\mathrm{f}_{\mathrm{y}_{ik}}, \mathrm{f}_{\mathrm{z}ik}, \mathrm{f}_{\mathrm{y}_{i'k}}\}^*$ with

$$\mu^{IK}_{ii'k}(a_{rs}) := \left\{ \begin{array}{r|l} a_{rs} & a_{rs} \in \{\mathrm{f}_{\mathrm{y}_{ik}}, \mathrm{f}_{\mathrm{z}ik}, \mathrm{f}_{\mathrm{y}_{i'k}}\} \\ \varepsilon & a_{rs} \in \Sigma_{IK} \setminus \{\mathrm{f}_{\mathrm{y}_{ik}}, \mathrm{f}_{\mathrm{z}ik}, \mathrm{f}_{\mathrm{y}_{i'k}}\}. \end{array} \right.$$

For each $i,i' \in I$, $i \neq i'$ and $k \in K$

$$\mu^{IK}_{ii'k}(\mathscr{L}_{IK}) \cap \Sigma^*_{\{i,i'\}\{k\}} \mathrm{f}_{\mathrm{y}_{ik}} \mathrm{f}_{\mathrm{y}_{i'k}} = \emptyset. \tag{2}$$

For $i,i' \in I$, $i \neq i'$, $k \in K$ let $\nu_{ii'k} : \Sigma^*_{\{i,i'\}\{k\}} \to \{\mathrm{f}_{\mathrm{y}_{ik}}, \mathrm{f}_{\mathrm{z}ik}, \mathrm{f}_{\mathrm{y}_{i'k}}\}^*$ be defined by

$$\nu_{ii'k}(a_{rs}) := \left\{ \begin{array}{r|l} a_{rs} & a_{rs} \in \{\mathrm{f}_{\mathrm{y}_{ik}}, \mathrm{f}_{\mathrm{z}ik}, \mathrm{f}_{\mathrm{y}_{i'k}}\} \\ \varepsilon & a_{rs} \in \Sigma_{\{i,i'\}\{k\}} \setminus \{\mathrm{f}_{\mathrm{y}_{ik}}, \mathrm{f}_{\mathrm{z}ik}, \mathrm{f}_{\mathrm{y}_{i'k}}\}, \end{array} \right.$$

then

$$\mu^{IK}_{ii'k} = \nu_{ii'k} \circ \Pi^{IK}_{\{i,i'\}\{k\}}.$$

Hence,

$$\mu^{IK}_{ii'k}(\mathscr{L}_{IK}) = \nu_{ii'k}(\mathscr{L}_{\{i,i'\}\{k\}}) \text{ if } \mathscr{L}_{IK} \text{ is self-similar.}$$

Let $\iota_{ii'k} : \Sigma^*_{\{i,i'\}\{k\}} \to \Sigma^*_{\{1,2\}\{1\}}$ be the isomorphism defined by

$$\iota_{ii'k}(a_{ik}) := \left\{ \begin{array}{r|l} a_{11} & a_{ik} \in \Sigma_{ik} \\ a_{21} & a_{i'k} \in \Sigma_{i'k}, \end{array} \right.$$

then by Remark 1

$$\iota_{ii'k}(\mathscr{L}_{\{i,i'\}\{k\}}) = \mathscr{L}_{\{1,2\}\{1\}},$$

since $\nu_{ii'k} = \iota^{-1}_{ii'k} \circ \nu_{121} \circ \iota_{ii'k}$, $\mathscr{L}_{IK}$ fulfills the privacy requirement (2) for each index set $I$ and $K$ iff

$$\nu_{121}(\mathscr{L}_{\{1,2\}\{1\}}) \cap \Sigma^*_{\{1,2\}\{1\}} \mathrm{f}_{\mathrm{y}_{11}} \mathrm{f}_{\mathrm{y}_{21}} = \emptyset. \tag{3}$$

This can be verified by checking the finite automaton of $\mathscr{L}_{\{1,2\}\{1\}}$. The automaton of $\mathscr{L}_{\{1,2\}\{1\}}$ consists of 28 states. The minimal automaton of $\nu_{121}(\mathscr{L}_{\{1,2\}\{1\}})$ is shown in Fig. 5(d) which implies (3). Self-similarity of $\mathscr{L}_{IK}$ can be shown using the methods given in [10]. So $\mathscr{L}_{IK}$ fulfills the privacy requirement.

On the contrary, $\mathscr{L}'_{IK}$ defined by $SF$, $SG'$ and $L$ of Fig. 5 is not self-similar because of

$$\mathrm{f}_{\mathrm{x}_{11}} \mathrm{f}_{\mathrm{x}_{21}} \mathrm{f}_{\mathrm{x}_{31}} \mathrm{g}_{\mathrm{x}_{11}} \mathrm{g}_{\mathrm{y}_{11}} \mathrm{g}_{\mathrm{x}_{21}} \mathrm{g}_{\mathrm{x}_{31}} \mathrm{g}_{\mathrm{y}_{21}} \mathrm{f}_{\mathrm{y}_{11}} \mathrm{f}_{\mathrm{y}_{21}} \in \mathscr{L}'_{\{1,2,3\}\{1\}},$$

$$\mathrm{f}_{\mathrm{x}_{11}} \mathrm{f}_{\mathrm{x}_{21}} \mathrm{g}_{\mathrm{x}_{11}} \mathrm{g}_{\mathrm{y}_{11}} \mathrm{g}_{\mathrm{x}_{21}} \mathrm{g}_{\mathrm{y}_{21}} \mathrm{f}_{\mathrm{y}_{11}} \mathrm{f}_{\mathrm{y}_{21}} \in \Pi^{\{1,2,3\}\{1\}}_{\{1,2\}\{1\}}(\mathscr{L}'_{\{1,2,3\}\{1\}})$$

$$\text{but } \mathrm{f}_{\mathrm{x}_{11}} \mathrm{f}_{\mathrm{x}_{21}} \mathrm{g}_{\mathrm{x}_{11}} \mathrm{g}_{\mathrm{y}_{11}} \mathrm{g}_{\mathrm{x}_{21}} \mathrm{g}_{\mathrm{y}_{21}} \mathrm{f}_{\mathrm{y}_{11}} \mathrm{f}_{\mathrm{y}_{21}} \notin \mathscr{L}'_{\{1,2\}\{1\}}.$$

The same action sequence shows that $\mathscr{L}'_{IK}$ does not fulfill the privacy requirement.

The privacy requirement of the example is a typical safety property [14]. These properties describe that “nothing forbidden happens”. They can be formalised by a set $\mathscr{F}$ of forbidden action sequences. So a system $\mathscr{L}_{IK} \subset \Sigma^*_{IK}$ satisfies a safety property $\mathscr{F}_{IK} \subset \Sigma^*_{IK}$ iff $\mathscr{L}_{IK} \cap \mathscr{F}_{IK} = \emptyset$. More precisely, these are safety properties which can be expressed without “dummy actions” to descibe deadlocks.

In our example the privacy requirement is formalised by

$$\mathscr{F}_{IK} = \bigcup_{i,i' \in I, i \neq i', k \in K} (\mu^{IK}_{ii'k})^{-1}(\Sigma^*_{\{i,i'\}\{k\}} \mathrm{f}_{\mathrm{y}_{ik}} \mathrm{f}_{\mathrm{y}_{i'k}}) =$$

$$= \bigcup_{i,i' \in I, i \neq i', k \in K} (\Pi^{IK}_{\{i,i'\}\{k\}})^{-1}(\iota^{-1}_{ii'k}[\nu^{-1}_{121}(\Sigma^*_{\{1,2\}\{1\}} \mathrm{f}_{\mathrm{y}_{11}} \mathrm{f}_{\mathrm{y}_{21}})])$$

because of

$$\mu^{IK}_{ii'k} = \iota^{-1}_{ii'k} \circ \nu_{121} \circ \iota_{ii'k} \circ \Pi^{IK}_{\{i,i'\}\{k\}} \text{ and}$$

$$\iota_{ii'k}(\Sigma^*_{\{i,i'\}\{k\}} \mathrm{f}_{\mathrm{y}_{ik}} \mathrm{f}_{\mathrm{y}_{i'k}}) = \Sigma^*_{\{1,2\}\{1\}} \mathrm{f}_{\mathrm{y}_{11}} \mathrm{f}_{\mathrm{y}_{21}}.$$

As

$$\nu^{-1}_{121}(\Sigma^*_{\{1,2\}\{1\}} \mathrm{f}_{\mathrm{y}_{11}} \mathrm{f}_{\mathrm{y}_{21}}) \subset \Sigma^*_{\{1,2\}\{1\}} \text{ and } \iota^{-1}_{ii'k} \in \mathscr{I}^{\{1,2\}\{1\}}_{\{i,i'\}\{k\}},$$

we now generally consider safety properties formalised by

$$\mathscr{F}^{\mathring{F}}_{IK} = \bigcup_{I' \subset I, K' \subset K, \iota^{\mathring{I}\mathring{K}}_{I'K'} \in \mathscr{I}^{\mathring{I}\mathring{K}}_{I'K'}} (\Pi^{IK}_{I'K'})^{-1}(\iota^{\mathring{I}\mathring{K}}_{I'K'}(\mathring{F})) \text{ and}$$

generated by $\mathring{F} \subset \Sigma^*_{\mathring{I}\mathring{K}}$.

For the privacy requirement above

$$\mathring{F} = \nu^{-1}_{121}(\Sigma^*_{\{1,2\}\{1\}} \mathrm{f}_{\mathrm{y}_{11}} \mathrm{f}_{\mathrm{y}_{21}}),\ \mathring{I} = \{1,2\},\ \mathring{K} = \{1\}.$$

By this definition

$$\mathscr{F}^{\mathring{F}}_{IK} = \emptyset \text{ for } |I| < |\mathring{I}| \text{ or } |K| < |\mathring{K}|, \tag{4}$$

as in that case $\mathscr{I}^{\mathring{I}\mathring{K}}_{I'K'} = \emptyset$ for each $I' \subset I$ and $K' \subset K$.

Now by the same argument as in our privacy example, we get

**Theorem 2.** *Self-similarity of $\mathscr{L}_{IK}$ implies that for a fixed $\mathring{F} \subset \Sigma^*_{\mathring{I}\mathring{K}}$ holds*

$$\mathscr{L}_{IK} \cap \mathscr{F}^{\mathring{F}}_{IK} = \emptyset \textit{ for each index sets } I \textit{ and } K$$

$$\textit{iff for the fixed index sets } \mathring{I} \textit{ and } \mathring{K} \quad \mathscr{L}_{\mathring{I}\mathring{K}} \cap \mathring{F} = \emptyset.$$

If $\mathscr{L}_{\mathring{I}\mathring{K}}$ and $\mathring{F}$ are regular subsets of $\Sigma^*_{\mathring{I}\mathring{K}}$ this can be checked by finite state methods [15]. On account of (4) it makes sense to consider safety properties defined by

$$\mathscr{F}_{IK} := \bigcup_{t \in T} \mathscr{F}^{\mathring{F}_t}_{IK} \text{ with finite } T \text{ and } \mathring{F}_t \subset \Sigma^*_{\mathring{I}_t\mathring{K}_t} \tag{5}$$

for each $t \in T$.

**Definition 4** (Uniformly parameterised safety properties)**.** *Safety properties of the form* (5) *we call* uniformly parameterised.

**Corollary 1.** *For self-similar $\mathscr{L}_{IK}$ the parameterised problem of verifying a uniformly parameterised safety property is reduced to finite many fixed finite state problems if the corresponding $\mathscr{L}_{\mathring{I}_t\mathring{K}_t}$ and $\mathring{F}_t$ are regular languages.*

Example 3 can also be seen as a dependability example when we assume that the managed resource is a rail track, the clients are trains and the policy to ensure integrity and safety of the system demands that only one train is allowed to use the rail track at a time. We can also derive an authentication example when we assume the client to be an ATM with $\mathrm{f_x}$ (authenticate credit card), $\mathrm{f_y}$ (draw-out cash) and $\mathrm{f_z}$ (eject credit card) and the server actions respectively manage a bank account. Please note that a stronger property "no further authentication before the card is ejected" does not hold for that specification.

## VI. Conclusions and Future Work

The main result of this paper is to demonstrate the significance of self-similarity for verification of security properties. We assume that uniformly parameterised structures like the uniformly parameterised cooperations we have defined and used here are likely to appear in any highly scalable system or system of systems such as cloud computing platforms or vehicular communication systems.

It is well known that dynamic system properties such as dependability and security properties are divided into safety and liveness properties [14]. Safety properties can be formalised by formal languages as demonstrated in section V. We have shown here in particular that for self-similar $\mathscr{L}_{IK}$ the parameterised problem of verifying a uniformly parameterised safety property can be reduced to finite many fixed finite state problems under certain regularity restrictions. For abstractions defined by alphabetic language homomorphisms it is easy to see that an abstract system satisfies a safety property as considered in Sect. V iff the concrete system satisfies a corresponding safety property. So our notion of self-similarity is compatible with uniformly parameterised safety properties.

Concerning liveness properties ("Eventually something desired happens.") such a relation between abstract and concrete systems does not hold in general. In [16] a property of homomorphisms is given that implies a similar relation between liveness properties of an abstract and a concrete system w.r.t. a modified satisfaction relation ("Eventually something desired is possible."). Based on that framework we will investigate liveness aspects of uniformly parameterised cooperations as well as safety properties related to deadlocks in a forthcoming paper. Another topic of interest is the generalisation of this method to n-sided cooperations.

## Acknowledgement

Roland Rieke developed the work presented here in the context of the project MASSIF (ID 257475) being co-funded by the European Commission within FP7.

## References

[1] A. Fuchs and R. Rieke, "Identification of Authenticity Requirements in Systems of Systems by Functional Security Analysis," in *Workshop on Architecting Dependable Systems (WADS 2009), in Proceedings of the 2009 IEEE/IFIP Conference on Dependable Systems and Networks, Supplementary Volume*, 2009.

[2] C. N. Ip and D. L. Dill, "Verifying Systems with Replicated Components in Mur$\varphi$," *Formal Methods in System Design*, vol. 14, no. 3, pp. 273–310, 1999.

[3] F. Derepas and P. Gastin, "Model checking systems of replicated processes with SPIN," in *Proceedings of the 8th International SPIN Workshop on Model Checking Software (SPIN'01)*, ser. Lecture Notes in Computer Science, M. B. Dwyer, Ed., vol. 2057. Toronto, Canada: Springer, May 2001, pp. 235–251.

[4] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre, "Incremental Verification by Abstraction." in *TACAS*, ser. Lecture Notes in Computer Science, T. Margaria and W. Yi, Eds., vol. 2031. Springer, 2001, pp. 98–112.

[5] S. Basu and C. R. Ramakrishnan, "Compositional analysis for verification of parameterized systems," *Theor. Comput. Sci.*, vol. 354, no. 2, pp. 211–229, 2006.

[6] R. Milner, *Communication and Concurrency*, ser. International Series in Computer Science. NY: Prentice Hall, 1989.

[7] J. Bradfield and C. Stirling, "Modal logics and mu-calculi: an introduction," 2001. [Online]. Available: citeseer.ist.psu.edu/bradfield01modal.html

[8] P. Ochsenschläger and R. Rieke, "Abstraction Based Verification of a Parameterised Policy Controlled System," in *International Conference "Mathematical Methods, Models and Architectures for Computer Networks Security" (MMM-ACNS-7)*, ser. CCIS, vol. 1. Springer, September 2007.

[9] T. E. Uribe, "Combinations of Model Checking and Theorem Proving," in *FroCoS '00: Proceedings of the Third International Workshop on Frontiers of Combining Systems*. London, UK: Springer-Verlag, 2000, pp. 151–170.

[10] P. Ochsenschläger and R. Rieke, "Uniform Parameterisation of Phase Based Cooperations," Fraunhofer SIT, Tech. Rep. SIT-TR-2010/1, 2010. [Online]. Available: http://sit.sit.fraunhofer.de/smv/publications

[11] M. Jantzen, "Extending Regular Expressions with Iterated Shuffle," *Theor. Comput. Sci.*, vol. 38, pp. 223–247, 1985.

[12] J. Jedrzejowicz and A. Szepietowski, "Shuffle languages are in P," *Theor. Comput. Sci.*, vol. 250, no. 1-2, pp. 31–53, 2001.

[13] H. Björklund and M. Bojanczyk, "Shuffle Expressions and Words with Nested Data," in *Mathematical Foundations of Computer Science 2007*, 2007, pp. 750–761.

[14] B. Alpern and F. B. Schneider, "Defining liveness," *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, October 1985.

[15] J. Sakarovitch, *Elements of Automata Theory*. Cambridge University Press, 2009.

[16] U. Nitsche and P. Ochsenschläger, "Approximately satisfied properties of systems and simple language homomorphisms," *Information Processing Letters*, vol. 60, pp. 201–206, 1996.

# P8

# RELIABILITY ASPECTS OF UNIFORMLY PARAMETERISED COOPERATIONS

| | |
|---|---|
| **Title** | Reliability Aspects of Uniformly Parameterised Cooperations |
| **Authors** | Peter Ochsenschläger and Roland Rieke |
| **Publication** | In *ICONS 2012, The Seventh International Conference on Systems, February 29 - March 5, 2012 - Saint Gilles, Reunion Island*, pages 25–34. |
| **ISBN/ISSN** | ISBN 978-1-61208-184-7 |
| **URL** | `http://www.thinkmind.org/index.php?view=article&articleid=icons_2012_2_10_20024` |
| **Status** | Published |
| **Publisher** | IARIA |
| **Publication Type** | ThinkMind(TM) Digital Library |
| **Copyright** | 2012, IARIA |
| **Contribution of Roland Rieke** | Co-Author with significant contribution, editor, and presenter at the Seventh International Conference on Systems (ICONS 2012). Specific contributions are: (1) analysis of the examples given in the paper in the SHVT. Related contribution: Roland Rieke also contributed to a related paper "Security Requirements for Uniformly Parameterised Cooperations" [Ochsenschläger & Rieke, 2012b] that he presented at the special session on security in networked and distributed systems at the 20th Euromicro Conference on Parallel, Distributed and Network-Based Processing. |

Table 13: Fact Sheet Publication *P8*

Publication P8 [Ochsenschläger & Rieke, 2012a] addresses the following research question:

RQ4 *Which design principles facilitate verifiability of security properties of scalable systems?*

In this paper reliability aspects of systems, which are characterised by the composition of a set of identical components are examined.

These components interact in a uniform manner, described by the schedules of the partners. Such kind of interaction is typical for scalable complex systems with cloud or grid structure. In addition to the safety properties of such *uniformly parameterised cooperations* which have been analysed in P7 reliability of such systems in a possibilistic sense is considered. This is formalised by always-eventually properties, a special class of liveness properties using a modified satisfaction relation, which expresses possibilities. As a main result, a finite state verification framework for uniformly parameterised reliability properties is given. The keys to this framework are structuring cooperations into phases and defining closed behaviours of systems. In order to verify reliability properties of such uniformly parameterised cooperations, finite state semi-algorithms that are independent of the concrete parameter setting are used.

# Reliability Aspects of Uniformly Parameterised Cooperations

Peter Ochsenschläger and Roland Rieke
*Fraunhofer Institute for Secure Information Technology, SIT*
*Darmstadt, Germany*
*Email: peter-ochsenschlaeger@t-online.de, roland.rieke@sit.fraunhofer.de*

***Abstract*—In this paper, we examine reliability aspects of systems, which are characterised by the composition of a set of identical components. These components interact in a uniform manner, described by the schedules of the partners. Such kind of interaction is typical for scalable complex systems with cloud or grid structure. We call these systems "uniformly parameterised cooperations". We consider reliability of such systems in a possibilistic sense. This is formalised by always-eventually properties, a special class of liveness properties using a modified satisfaction relation, which expresses possibilities. As a main result, a finite state verification framework for uniformly parameterised reliability properties is given. The keys to this framework are structuring cooperations into phases and defining closed behaviours of systems. In order to verify reliability properties of such uniformly parameterised cooperations, we use finite state semi-algorithms that are independent of the concrete parameter setting.**

***Keywords*-reliability aspects of scalable complex systems; liveness properties; uniformly parameterised reliability properties; finite state verification; possibilistic reliability.**

## I. Introduction

The transition from systems composed of many isolated, small-scale elements to large-scale, distributed and massively interconnected systems is a key challenge of modern information and communications technologies. These systems need to be dependable, which means they need to remain secure, robust and efficient [1]. Examples for highly scalable systems comprise (i) grid computing architectures; and (ii) cloud computing platforms. In grid computing, large scale allocation issues relying on centralised controls present challenges that threaten to overwhelm existing centralised management approaches [1]. Cloud computing introduced the concept, to make software available as a service. This concept can only be successful, if certain obstacles such as reliability issues are solved [2]. In order to be able to model functional requirements of dependable systems best satisfying both fault-tolerance and security attributes, three distinct classes of (system specification) properties need to be considered, namely *safety*, *liveness*, and *information flow* [3]. Concrete reliability problems related to liveness properties range from replica selection to consistency of cloud storage (which allows multiple clients to access stored data concurrently in a consistent fashion) [4]. Most existing replica selection schemes rely on either central coordination (which has reliability, security, and scalability limitations) or distributed heuristics (which may lead to instability) [4]. Another important issue is, that clients of cloud services do not operate continuously, so clients should not depend on other clients for liveness of their operations [5].

In this paper, we consider systems that interact in a way that is typical for scalable complex systems. These systems, which we call *uniformly parameterised cooperations*, are characterised by (i) the composition of a set of identical components (copies of a two-sided cooperation); and (ii) the fact that these components interact in a uniform manner (described by the schedules of the partners). As an example of such uniformly parameterised systems of cooperations, e-commerce protocols can be considered. In these protocols, the two cooperation partners have to perform a certain kind of financial transactions. Such a protocol should work for several partners in the same manner, and the mechanism (schedule) to determine how one partner may be involved in several cooperations is the same for each partner. So, the cooperation is parameterised by the partners and the parameterisation should be uniform with respect to the partners.

Reliability is an important concept related to dependability, which ensures *continuity of correct service* [6]. In this paper, we consider reliability in a possibilistic sense, which means that correct services can be provided according to a certain pattern of behaviour again and again. These possibilities of providing correct services are expressed by a special class of liveness properties using a modified satisfaction relation. We call these properties *always-eventually properties*.

As a main result of the work presented, a finite state verification framework for *uniformly parameterised reliability properties* is given. The keys to this framework are structuring cooperations into phases and defining closed behaviours of systems. In this framework, *completion of phases strategies* and corresponding *success conditions* can be formalised [7], which produce finite state semi-algorithms that are independent of the concrete parameter setting. These algorithms are used to verify reliability properties of uniformly parameterised cooperations under certain regularity restrictions.

The paper is structured as follows. Section II gives an overview of the related work. In Section III, uniform parameterisations of two-sided cooperations in terms of formal language theory is formalised. Section IV introduces the concept of uniformly parameterised reliability properties. The concept of structuring cooperations into phases given in Section V enables completion of phases strategies, which are described in Section VI. Consistent with this, corresponding success conditions can be formalised [2], which produce finite state semi-algorithms to verify reliability properties of uniformly parameterised cooperations. Finally, the paper ends with conclusions and an outlook in Section VII.

## II. Related Work

*System properties:* A formal definition of safety and liveness properties is proposed in [8]. In [9], we defined a satisfaction relation, called *approximate satisfaction*, which expresses a possibilistic view on liveness and is equivalent to the satisfaction relation in [8] for safety properties. In this paper, we extended this concept (cf. Section IV) and defined *uniformly parameterised reliability properties*, which fit to the parameterised structure of the systems, which we consider here. Besides these safety and liveness properties so called "hyperproperties" [10] are of interest because they give formalisations for non-interference and non-inference.

*Verification approaches for parameterised systems:* An extension to the *Mur$\varphi$* verifier to verify systems with replicated identical components through a new data type called RepetitiveID is presented in [11]. A typical application area of this tool are cache coherence protocols. The aim of [12] is an abstraction method through symmetry, which works also when using variables holding references to other processes. This is not possible in Mur$\varphi$. In [13], a methodology for constructing abstractions and refining them by analysing counter-examples is presented. The method combines abstraction, model-checking and deductive verification. However, this approach does not consider liveness properties. In [14], a technique for automatic verification of parameterised systems based on process algebra *CCS* [15] and the logic modal *mu-calculus* [16] is presented. This technique views processes as property transformers and is based on computing the limit of a sequence of mu-calculus formula generated by these transformers. The above-mentioned approaches demonstrate, that finite state methods combined with deductive methods can be applied to analyse parameterised systems. The approaches differ in varying amounts of user intervention and their range of application. A survey of approaches to combine model checking and theorem proving methods is given in [17].

*Iterated shuffle products:* In [18], it is shown that our definition of uniformly parameterised cooperations is strongly related to iterated shuffle products [19], if the cooperations are "structured into phases". The main concept for such a condition are shuffle automata [20] (multicounter automata [21]) whose computations, if they are deterministic, unambiguously describe how a cooperation partner is involved in several phases.

In [22], we have shown in particular that for self-similar parameterised systems $\mathscr{L}_{IK}$ the parameterised problem of verifying a *uniformly parameterised safety property* can be reduced to finite many fixed finite state problems.

Complementary to this, in the present paper, we define a uniformly parameterised reliability property based on this concept. The main result is a finite state verification framework for such *uniformly parameterised reliability properties*.

## III. Parameterised Cooperations

The behaviour $L$ of a discrete system can be formally described by the set of its possible sequences of actions. Therefore $L \subset \Sigma^*$ holds where $\Sigma$ is the set of all actions of the system, and $\Sigma^*$ (free monoid over $\Sigma$) is the set of all finite sequences of elements of $\Sigma$ (words), including the empty sequence denoted by $\varepsilon$. $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$. Subsets of $\Sigma^*$ are called formal languages [23]. Words can be composed: if $u$ and $v$ are words, then $uv$ is also a word. This operation is called the *concatenation*; especially $\varepsilon u = u\varepsilon = u$. Concatenation of formal languages $U, V \subset \Sigma^*$ are defined by $UV := \{uv \in \Sigma^* | u \in U \text{ and } v \in V\}$. A word $u$ is called a *prefix* of a word $v$ if there is a word $x$ such that $v = ux$. The set of all prefixes of a word $u$ is denoted by $\text{pre}(u)$; $\varepsilon \in \text{pre}(u)$ holds for every word $u$. The set of possible continuations of a word $u \in L$ is formalised by the *left quotient* $u^{-1}(L) := \{x \in \Sigma^* | ux \in L\}$.

Infinite words over $\Sigma$ are called $\omega$-words [24]. The set of all infinite words over $\Sigma$ is denoted $\Sigma^\omega$. An $\omega$-language $L$ over $\Sigma$ is a subset of $\Sigma^\omega$. For $u \in \Sigma^*$ and $v \in \Sigma^\omega$ the *left concatenation* $uv \in \Sigma^\omega$ is defined. It is also defined for $U \subset \Sigma^*$ and $V \subset \Sigma^\omega$ by $UV := \{uv \in \Sigma^\omega | u \in U \text{ and } v \in V\}$.

For an $\omega$-word $w$ the prefix set is given by the formal language $\text{pre}(w)$, which contains every finite prefix of $w$. The prefix set of an $\omega$-language $L \subset \Sigma^\omega$ is accordingly given by $\text{pre}(L) = \{u \in \Sigma^* |$ it exist $v \in \Sigma^\omega$ with $uv \in L\}$. For $M \subset \Sigma^*$ the *$\omega$-power* $M^\omega \subset \Sigma^\omega$ is the set of all "infinite concatenations" of arbitrary elements of $M$. More formal definitions of theses $\omega$-notions are given in the appendix.

Formal languages, which describe system behaviour, have the characteristic that $\text{pre}(u) \subset L$ holds for every word $u \in L$. Such languages are called *prefix closed*. System behaviour is thus described by prefix closed formal languages.

Different formal models of the same system are partially ordered with respect to different levels of abstraction. Formally, abstractions are described by so called alphabetic language homomorphisms. These are mappings $h^* : \Sigma^* \longrightarrow \Sigma'^*$ with $h^*(xy) = h^*(x)h^*(y)$ , $h^*(\varepsilon) = \varepsilon$ and $h^*(\Sigma) \subset \Sigma' \cup \{\varepsilon\}$. So, they are uniquely defined by corresponding mappings $h : \Sigma \longrightarrow \Sigma' \cup \{\varepsilon\}$. In the following, we denote both the mapping $h$ and the homomorphism $h^*$ by $h$. Inverse homomorphism are denoted by $h^{-1}$. Let $L$ be a language over the alphabet $\Sigma'$. Then $h^{-1}(L)$ is the set of words $w \in \Sigma^*$ such that $h(w) \in L$. In this paper, we consider a lot of alphabetic language homomorphisms. So, for simplicity, we tacitly assume that a mapping between free monoids is an alphabetic language homomorphism if nothing contrary is stated.

To describe a two-sided cooperation, let $\Sigma = \Phi \cup \Gamma$ where $\Phi$ is the set of actions of cooperation partner $F$ and $\Gamma$ is the set of actions of cooperation partner $G$ and $\Phi \cap \Gamma = \emptyset$. Now a prefix closed language $L \subset (\Phi \cup \Gamma)^*$ formally defines a two-sided cooperation.

**Example 1.** *Let* $\Phi = \{\mathrm{f_s}, \mathrm{f_r}\}$ *and* $\Gamma = \{\mathrm{g_r}, \mathrm{g_i}, \mathrm{g_s}\}$ *and hence* $\Sigma = \{\mathrm{f_s}, \mathrm{f_r}, \mathrm{g_r}, \mathrm{g_i}, \mathrm{g_s}\}$. *An example for a cooperation* $L \subset \Sigma^*$ *is now given by the automaton in Figure 1. It describes a simple handshake between F (client) and G (server), where a client may perform the actions* $\mathrm{f_s}$ *(send a request),* $\mathrm{f_r}$ *(receive a result) and a server may perform the corresponding actions* $\mathrm{g_r}$ *(receive a request),* $\mathrm{g_i}$ *(internal action to compute the result) and* $\mathrm{g_s}$ *(send the result).*

In the following, we will denote initial states by a short incoming arrow and final states by double circles. In this automaton, all states are final states, since $L$ is prefix closed.

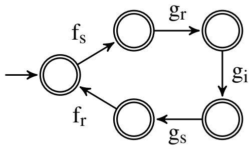


Figure 1. Automaton for 1-1-cooperation $L$

For parameter sets $I$, $K$ and $(i,k) \in I \times K$ let $\Sigma_{ik}$ denote pairwise disjoint copies of $\Sigma$. The elements of $\Sigma_{ik}$ are denoted by $a_{ik}$ and $\Sigma_{IK} := \bigcup_{(i,k)\in I\times K} \Sigma_{ik}$. The index $ik$ describes the bijection $a \leftrightarrow a_{ik}$ for $a \in \Sigma$ and $a_{ik} \in \Sigma_{ik}$. Now $\mathscr{L}_{IK} \subset \Sigma_{IK}^*$ (prefix-closed) describes a *parameterised system*. To avoid pathological cases, we generally assume parameter and index sets to be non empty.

For a cooperation between one partner of type $F$ with two partners of type $G$ in Example 1 let

$$\begin{aligned}
&\Phi_{\{1\}\{1,2\}} = \{\mathrm{f_{s11}}, \mathrm{f_{r11}}, \mathrm{f_{s12}}, \mathrm{f_{r12}}\},\\
&\Gamma_{\{1\}\{1,2\}} = \{\mathrm{g_{r11}}, \mathrm{g_{i11}}, \mathrm{g_{s11}}, \mathrm{g_{r12}}, \mathrm{g_{i12}}, \mathrm{g_{s12}}\} \text{ and}\\
&\Sigma_{\{1\}\{1,2\}} = \Phi_{\{1\}\{1,2\}} \cup \Gamma_{\{1\}\{1,2\}}.
\end{aligned}$$

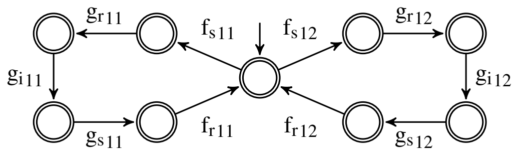


Figure 2. Automaton for 1-2-cooperation $\mathscr{L}_{\{1\}\{1,2\}}$

A 1-2-cooperation, where each pair of partners cooperates restricted by $L$ and each partner has to finish the handshake it just is involved in before entering a new one, is now given (by reachability analysis) by the automaton in Figure 2 for $\mathscr{L}_{\{1\}\{1,2\}}$. It shows that one after another client 1 runs a handshake either with server 1 or with server 2. Figure 3 in contrast depicts an automaton for a 2-1-cooperation $\mathscr{L}_{\{1,2\}\{1\}}$ with the same overall number of partners involved but two of type $F$ and one partner of type $G$. Figure 3 is more complex than Figure 2 because client 1 and client 2 may start a handshake independently of each other, but server 1 handles these handshakes one after another. A 5-3-cooperation with the same simple behaviour of partners already requires 194.677 states and 1.031.835 state transitions (computed by the SH verification tool [25]).

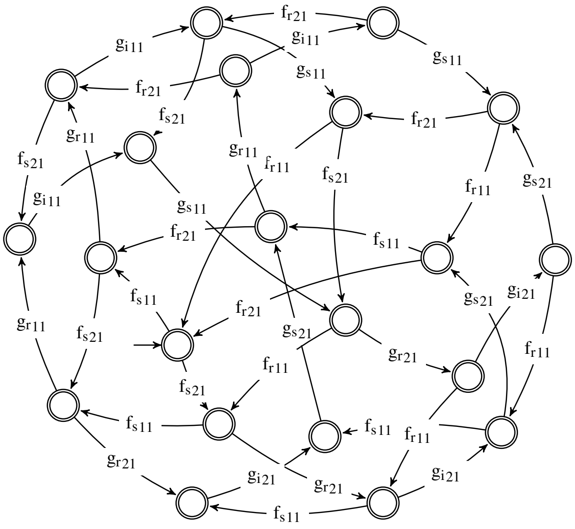


Figure 3. Automaton for the 2-1-cooperation $\mathscr{L}_{\{1,2\}\{1\}}$

For $(i,k) \in I \times K$, let $\pi_{ik}^{IK} : \Sigma_{IK}^* \to \Sigma^*$ with

$$\pi_{ik}^{IK}(a_{rs}) = \begin{cases} a \mid & a_{rs} \in \Sigma_{ik} \\ \varepsilon \mid & a_{rs} \in \Sigma_{IK} \setminus \Sigma_{ik} \end{cases} .$$

For *uniformly parameterised systems* $\mathscr{L}_{IK}$ we generally want to have

$$\mathscr{L}_{IK} \subset \bigcap_{(i,k)\in I\times K} ((\pi_{ik}^{IK})^{-1}(L))$$

because from an abstraction point of view, where only the actions of a specific $\Sigma_{ik}$ are considered, the complex system $\mathscr{L}_{IK}$ is restricted by $L$.

In addition to this inclusion, $\mathscr{L}_{IK}$ is defined by *local schedules* that determine how each "version of a partner" can participate in different cooperations. More precisely, let $SF \subset \Phi^*$, $SG \subset \Gamma^*$ be prefix closed. For $(i,k) \in I \times$

$K$, let $\varphi_i^{IK}:\Sigma_{IK}^*\to\Phi^*$ and $\gamma_k^{IK}:\Sigma_{IK}^*\to\Gamma^*$ with

$$\varphi_i^{IK}(a_{rs})=\begin{cases} a \mid & a_{rs}\in\Phi_{\{i\}K} \\ \varepsilon \mid & a_{rs}\in\Sigma_{IK}\setminus\Phi_{\{i\}K}\end{cases} \quad\text{and}$$

$$\gamma_k^{IK}(a_{rs})=\begin{cases} a \mid & a_{rs}\in\Gamma_{I\{k\}} \\ \varepsilon \mid & a_{rs}\in\Sigma_{IK}\setminus\Gamma_{I\{k\}}\end{cases},$$

where $\Phi_{IK}$ and $\Gamma_{IK}$ are defined correspondingly to $\Sigma_{IK}$.

**Definition 1** (uniformly parameterised cooperation)**.** *Let $I$, $K$ be finite parameter sets, then*

$$\mathscr{L}_{IK}:=\bigcap_{(i,k)\in I\times K}(\pi_{ik}^{IK})^{-1}(L)$$
$$\cap\bigcap_{i\in I}(\varphi_i^{IK})^{-1}(SF)\cap\bigcap_{k\in K}(\gamma_k^{IK})^{-1}(SG)$$

*denotes a* uniformly parameterised cooperation.

By this definition,

$$\mathscr{L}_{\{1\}\{1\}}=(\pi_{11}^{\{1\}\{1\}})^{-1}(L)$$
$$\cap\,(\varphi_1^{\{1\}\{1\}})^{-1}(SF)\cap(\gamma_1^{\{1\}\{1\}})^{-1}(SG).$$

Because we want $\mathscr{L}_{\{1\}\{1\}}$ being isomorphic to $L$ by the isomorphism $\pi_{11}^{\{1\}\{1\}}:\Sigma_{\{1\}\{1\}}^*\to\Sigma^*$, we additionally need

$$(\pi_{11}^{\{1\}\{1\}})^{-1}(L)\subset(\varphi_1^{\{1\}\{1\}})^{-1}(SF)\text{ and}$$
$$(\pi_{11}^{\{1\}\{1\}})^{-1}(L)\subset(\gamma_1^{\{1\}\{1\}})^{-1}(SG).$$

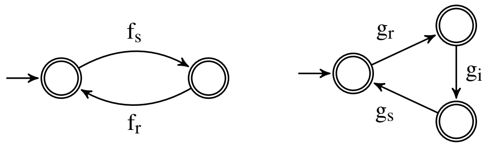
This is equivalent to $\pi_\Phi(L)\subset SF$ and $\pi_\Gamma(L)\subset SG$, where $\pi_\Phi:\Sigma^*\to\Phi^*$ and $\pi_\Gamma:\Sigma^*\to\Gamma^*$ are defined by

$$\pi_\Phi(a)=\begin{cases} a \mid & a\in\Phi \\ \varepsilon \mid & a\in\Gamma\end{cases} \text{ and } \pi_\Gamma(a)=\begin{cases} a \mid & a\in\Gamma \\ \varepsilon \mid & a\in\Phi\end{cases}.$$

So, we complete Def. 1 by the additional conditions

$$\pi_\Phi(L)\subset SF \text{ and } \pi_\Gamma(L)\subset SG.$$

Schedules $SF$ and $SG$ that fit to the cooperations given in Example 1 are depicted in Figs. 4(a) and 4(b). Here, we have $\pi_\Phi(L)=SF$ and $\pi_\Gamma(L)=SG$.



(a) Schedule $SF$ (b) Schedule $SG$

Figure 4. Automata $\mathbb{SF}$ and $\mathbb{SG}$ for the schedules $SF$ and $SG$

The system $\mathscr{L}_{IK}$ of cooperations is a typical example of a *complex system*. It consists of several identical components (copies of the two-sided cooperation $L$), which interact in a uniform manner (described by the schedules $SF$ and $SG$ and by the homomorphisms $\varphi_i^{IK}$ and $\gamma_k^{IK}$).

**Remark 1.** *It is easy to see that $\mathscr{L}_{IK}$ is isomorphic to $\mathscr{L}_{I'K'}$ if $I$ is isomorphic to $I'$ and $K$ is isomorphic to $K'$. More precisely, let $\iota_{I'}^I:I\to I'$ and $\iota_{K'}^K:K\to K'$ be bijections and let $\iota_{I'K'}^{IK}:\Sigma_{IK}^*\to\Sigma_{I'K'}^*$ be defined by*

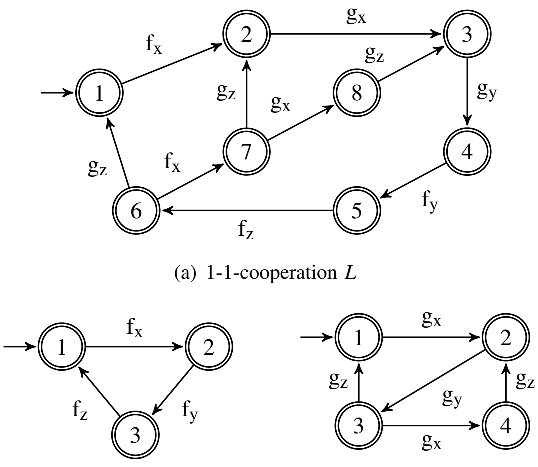
$$\iota_{I'K'}^{IK}(a_{ik}):=a_{\iota_{I'}^I(i)\iota_{K'}^K(k)}\textit{ for }a_{ik}\in\Sigma_{IK}.$$

*Hence, $\iota_{I'K'}^{IK}$ is a isomorphism and $\iota_{I'K'}^{IK}(\mathscr{L}_{IK})=\mathscr{L}_{I'K'}$. The set of all these isomorphisms $\iota_{I'K'}^{IK}$ defined by corresponding bijections $\iota_{I'}^I$ and $\iota_{K'}^K$ is denoted by $\mathscr{I}_{I'K'}^{IK}$.*

To illustrate the concepts of this paper, we consider the following example.

**Example 2.** *We consider a system of servers, each of them managing a resource, and clients, which want to use these resources. We assume that as a means to enforce a given* privacy policy *a server has to manage its resource in such a way that no client may access this resource while it is in use by another client (*privacy requirement*). This may be required to ensure* anonymity *in such a way that clients and their actions on a resource cannot be linked by an observer.*

We formalise this system at an abstract level, where a client may perform the actions $f_x$ (send a request), $f_y$ (receive a permission) and $f_z$ (send a free-message), and a server may perform the corresponding actions $g_x$ (receive a request), $g_y$ (send a permission) and $g_z$ (receive a free-message). The possible sequences of actions of a client resp. of a server are given by the automaton $\mathbb{SF}$ resp. $\mathbb{SG}$. The automaton $\mathbb{L}$ describes the 1-1-cooperation of one client and one server (see Figure 5). These automata define the client-server system $\mathscr{L}_{IK}$.



(a) 1-1-cooperation $L$

(b) Schedule $SF$ (c) Schedule $SG$

Figure 5. Automata $\mathbb{L}$, $\mathbb{SF}$ and $\mathbb{SG}$ for Example 2

## IV. A Class of Liveness Properties

Usually, behaviour properties of systems are divided into two classes: *safety* and *liveness* properties [8]. Intuitively a safety property stipulates that "something bad does not happen" and a liveness property stipulates that "something good eventually happens". In [8], both classes, as well as system behaviour, are formalised in terms of $\omega$-languages, because especially for liveness properties infinite sequences of actions have to be considered.

**Definition 2** (linear satisfaction)**.** *According to [8], a property $E$ of a system is a subset of $\Sigma^{\omega}$. If $S \subset \Sigma^{\omega}$ represents the behaviour of a system, then $S$* linearly satisfies *$E$ iff $S \subset E$.*

In [8], it is furthermore shown that each property $E$ is the intersection of a safety and a liveness property.

Safety properties $E_s \subset \Sigma^{\omega}$ are of the form $E_s = \Sigma^{\omega} \setminus F\Sigma^{\omega}$ with $F \subset \Sigma^*$, where $F$ is the set of "bad things".

Liveness properties $E_l \subset \Sigma^{\omega}$ are characterised by $\text{pre}(E_l) = \Sigma^*$. A typical example of a liveness property is

$$E_l = (\Sigma^* M)^{\omega} \text{ with } \emptyset \neq M \subset \Sigma^+. \tag{1}$$

This $E_l$ formalises that "always eventually a finite action sequence $m \in M$ happens".

We describe system behaviour by prefix closed languages $B \subset \Sigma^*$. So, in order to apply the framework of [8], we have to transform $B$ into an $\omega$-language. This can be done by the *limit* $\lim(B)$ [24]. For prefix closed languages $B \subset \Sigma^*$, their limit is defined by

$$\lim(B) := \{w \in \Sigma^{\omega} | \text{pre}(w) \subset B\}.$$

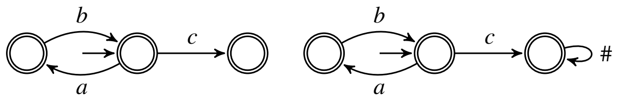
If $B$ contains maximal words $u$ (deadlocks), then these $u$ are not captured by $\lim(B)$. Formally the set $\max(B)$ of all maximal words of $B$ is defined by

$$\max(B) := \{u \in B | \text{ if } v \in B \text{ with } u \in \text{pre}(v), \text{ then } v = u\}.$$

Now, using a dummy action #, $B$ can be unambiguously described by

$$\hat{B} := B \cup \max(B)\#^* \subset \hat{\Sigma}^*,$$

where $\# \notin \Sigma$ and $\hat{\Sigma} := \Sigma \cup \{\#\}$. By this definition, in $\hat{B}$ the maximal words of $B$ are continued by arbitrary many #'s. So, $\hat{B}$ does not contain maximal words.



(a) Automaton for $B$ (b) Automaton for $\hat{B}$

Figure 6. Automata for $B$ and $\hat{B}$

Let for example $B$ be given by the automaton in Figure 6(a), then $\hat{B}$ is given by the automaton in Figure 6(b).

By this construction, we now can assume that system behaviour is formalised by prefix closed languages $\hat{B} \subset \Sigma^*\#^* \subset \hat{\Sigma}^*$ without maximal words, and the corresponding infinite system behaviour $S \subset \Sigma^{\omega}$ is given by $S := \lim(\hat{B})$.

For such an $S$ and safety properties $E = \hat{\Sigma}^{\omega} \setminus F\hat{\Sigma}^{\omega}$ with $F \subset \hat{\Sigma}^*$ it holds

$$S \subset E \text{ iff } S \cap F\hat{\Sigma}^{\omega} = \emptyset \text{ iff } \text{pre}(S) \cap F = \emptyset \text{ iff } \hat{B} \cap F = \emptyset.$$

If $F \subset \Sigma^*$, then $\hat{B} \cap F = \emptyset$ iff $B \cap F = \emptyset$. Therefore,

$$S \subset E \text{ iff } B \cap F = \emptyset \text{ for } F \subset \Sigma^*. \tag{2}$$

So, by (2) our approach in [22] is equivalent to the $\omega$-notation of safety properties described by $F \subset \Sigma^*$.

Linear satisfaction (cf. Def. 2) is too strong for systems in our focus with respect to liveness properties, because $S = \lim(\hat{B})$ can contain "unfair" infinite behaviours, which are not elements of $E$.

Let for example $I \supset \{1,2\}$ and $K \supset \{1\}$, then $\lim(\widehat{\mathscr{L}_{IK}}) \cap \Sigma^{\omega}_{\{1\}\{1\}} \neq \emptyset$, which means that infinite action sequences exist, where only the partners with index 1 cooperate. So, if a property specification involves actions of a partner with index 2, as for instance $E = \Sigma^*_{IK}\Sigma_{\{2\}\{1\}}\Sigma^{\omega}_{IK}$, then this property is not linearly satisfied because $\lim(\widehat{\mathscr{L}_{IK}}) \not\subset E$.

Instead of neglecting such unfair infinite behaviours, we use a weaker satisfaction relation, called *approximate satisfaction*, which implicitly expresses some kind of fairness.

**Definition 3** (approximate satisfaction)**.** *A system $S \subset \hat{\Sigma}^{\omega}$* approximately satisfies *a property $E \subset \hat{\Sigma}^{\omega}$ iff each finite behaviour (finite prefix of an element of $S$) can be continued to an infinite behaviour, which belongs to $E$. More formally, $pre(S) \subset pre(S \cap E)$.*

In [9], it is shown, that for safety properties linear satisfaction and approximate satisfaction are equivalent.

With respect to approximate satisfaction, liveness properties stipulate that "something good" eventually is possible.

Many practical liveness properties are of the form (1). Let us consider a prefix closed language $B \subset \Sigma^*$ and a formal language $\emptyset \neq M \subset \Sigma^+$. By definition 3 $\lim(\hat{B})$ approximately satisfies $(\hat{\Sigma}^* M)^{\omega}$ iff each $u \in B$ is prefix of some $v \in B$ with

$$v^{-1}(B) \cap M \neq \emptyset. \tag{3}$$

If $B$ and $M$ are regular sets, then (3) can be checked by usual automata algorithms [23] without referring to $\lim(\hat{B}) \cap (\hat{\Sigma}^* M)^{\omega}$.

Let us now consider the prefix closed language $L \subset \Sigma^*$ of example 2 and the "phase" $P \subset \Sigma^+$ given by the automaton $\mathbb{P}$ in Figure 7.

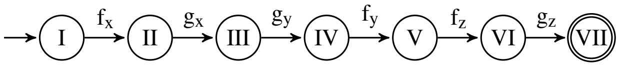


Figure 7. Automaton $\mathbb{P}$

$$\lim(\hat{L}) \text{ approximately satisfies the liveness property } (\hat{\Sigma}^* P)^{\omega} \subset \hat{\Sigma}^*, \text{ because the automaton } \mathbb{L} \text{ in Figure 5(a) is strongly connected and } P \subset L. \tag{4}$$

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-184-7

(4) states that in the 1-1-cooperation $\lim(\hat{L})$ always eventually a "complete run through the phase $P$" is possible. This is a typical reliability property.

Properties of the form $(\hat{\Sigma}^* M)^\omega$ with $\emptyset \neq M \subset \Sigma^+$ we call *always-eventually properties*.

Let now $\emptyset \neq \mathring{M} \subset \Sigma^+_{\mathring{I}\mathring{K}}$ with fixed finite index sets $\mathring{I}$ and $\mathring{K}$. Then

$$(\hat{\Sigma}^*_{IK} \mathring{M})^\omega$$

is an always-eventually property for each finite index sets $I \supset \mathring{I}$ and $K \supset \mathring{K}$. Using bijections on $\mathring{I}$ and $\mathring{K}$ this can easily be generalised to each finite index sets $I$ and $K$ with $|I| \geq |\mathring{I}|$ and $|K| \geq |\mathring{K}|$, where $|I|$ denotes the cardinality of the set $I$. More precisely, let $\mathscr{I}^{\mathring{I}\mathring{K}}_{I'K'}$ be the set of all isomorphisms $\iota^{\mathring{I}\mathring{K}}_{I'K'} : \Sigma^*_{\mathring{I}\mathring{K}} \to \Sigma^*_{I'K'}$ generated by bijections $\iota^{\mathring{I}}_{I'} : \mathring{I} \to I'$ and $\iota^{\mathring{K}}_{K'} : \mathring{K} \to K'$ in such a way that

$$\iota^{\mathring{I}\mathring{K}}_{I'K'}(a_{ik}) := a_{\iota^{\mathring{I}}_{I'}(i)\iota^{\mathring{K}}_{K'}(k)}$$

for $a_{ik} \in \Sigma_{\mathring{I}\mathring{K}}$. Then

$$(\hat{\Sigma}^*_{IK} \iota^{\mathring{I}\mathring{K}}_{I'K'}(\mathring{M}))^\omega$$

is an always-eventually property for each $I \supset I'$, $K \supset K'$ and $\iota^{\mathring{I}\mathring{K}}_{I'K'} \in \mathscr{I}^{\mathring{I}\mathring{K}}_{I'K'}$. For finite index sets $\mathring{I}$, $I$, $\mathring{K}$ and $K$ let

$$\mathscr{I}[(\mathring{I},\mathring{K}),(I,K)] := \bigcup_{I' \subset I, K' \subset K} \mathscr{I}^{\mathring{I}\mathring{K}}_{I'K'}.$$

Note that $\mathscr{I}[(\mathring{I},\mathring{K}),(I,K)] = \emptyset$ if $|\mathring{I}| > |I|$ or $|\mathring{K}| > |K|$.

**Definition 4** (uniformly parameterised reliability property)**.** *Let $\mathring{I}$, $I$, $\mathring{K}$ and $K$ be finite index sets with $|\mathring{I}| \leq |I|$ and $|\mathring{K}| \leq |K|$. If $\emptyset \neq \mathring{M} \subset \Sigma^+_{\mathring{I}\mathring{K}}$, then the family*

$$\mathscr{A}^{\mathring{M}}_{IK} := [(\hat{\Sigma}^*_{IK} \iota^{\mathring{I}\mathring{K}}_{I'K'}(\mathring{M}))^\omega]_{\iota^{\mathring{I}\mathring{K}}_{I'K'} \in \mathscr{I}[(\mathring{I},\mathring{K}),(I,K)]}.$$

*is a* strong uniformly parameterised always-eventually property *(uniformly parameterised reliability property).*

*We say that lim$(\widehat{\mathscr{L}_{IK}})$* approximately satisfies such a family $\mathscr{A}^{\mathring{M}}_{IK}$ *iff lim$(\widehat{\mathscr{L}_{IK}})$ approximately satisfies each of the properties $(\hat{\Sigma}^*_{IK} \iota^{\mathring{I}\mathring{K}}_{I'K'}(\mathring{M}))^\omega$ for $\iota^{\mathring{I}\mathring{K}}_{I'K'} \in \mathscr{I}[(\mathring{I},\mathring{K}),(I,K)]$.*

**Remark 2.** *We use the adjective* strong, *because in [7] uniform parameterisations of general properties are defined, which, in case of always-eventually properties, are weaker than definition 4.*

Let us return to example 2 and let

$$\mathring{P} := (\pi^{\{1\}\{1\}}_{11})^{-1} P \subset \Sigma^+_{\{1\}\{1\}} \text{ and}$$
$$\mathring{E} := (\widehat{\Sigma_{\{1\}\{1\}}}^* \mathring{P})^\omega \subset \widehat{\Sigma_{\{1\}\{1\}}}^\omega. \tag{5}$$

Because $\pi^{\{1\}\{1\}}_{11} : \Sigma^*_{\{1\}\{1\}} \to \Sigma^*$ is an isomorphism, by (4) $\lim(\widehat{\mathscr{L}_{\{1\}\{1\}}})$ approximately satisfies $\mathring{E}$.

Now by definition 4 $\lim(\widehat{\mathscr{L}_{IK}})$ approximately satisfies $\mathscr{A}^{\mathring{P}}_{IK}$ iff in $\lim(\widehat{\mathscr{L}_{IK}})$ for each pair of clients and servers always eventually a complete run through a phase $P$ is possible.

## V. Cooperations based on phases

The schedule $SG$ of example 2 shows that a server may cooperate with two clients partly in an interleaving manner. To formally capture such behaviour, cooperations are structured into phases [18]. This formalism is based on iterated shuffle products and leads to sufficient conditions for liveness properties (cf. Section VI).

Shuffling two words means arbitrarily inserting one word into the other word, like shuffling two decks of cards. In [21], this is formalised as follows:

A word $w \in \Sigma^*$ is called a *shuffle* of words $w_1, \ldots, w_m \in \Sigma^*$ if the positions of $w$ can be coloured using $m$ colors so that the positions with color $i \in \{1, \ldots, m\}$, when read from left to right, form the word $w_i$. *Shuffle of a set* $P \subset \Sigma^*$, is $\{w : w$ is a shuffle of some $w_1, \ldots, w_m \in P$, for some $m \in \mathbb{N}\}$.

However, we now provide an alternative formalisation, which is more adequate to the considerations in this paper.

**Definition 5** (iterated shuffle product)**.** *Let $t \in \mathbb{N}$, and for each $t$ let $\Sigma_t$ be a copy of $\Sigma$. Let all $\Sigma_t$ be pairwise disjoint. The index $t$ describes the bijection $a \leftrightarrow a_t$ for $a \in \Sigma$ and $a_t \in \Sigma_t$ (which is equivalent to a colouring with color $t$ in the formalism of [21]). Let*

$$\Sigma_{\mathbb{N}} := \bigcup_{t \in \mathbb{N}} \Sigma_t, \text{ and for each } t \in \mathbb{N}$$

*let the homomorphisms $\tau^{\mathbb{N}}_t$ and $\Theta^{\mathbb{N}}$ be defined by*

$$\tau^{\mathbb{N}}_t : \Sigma^*_{\mathbb{N}} \to \Sigma^* \text{ with } \tau^{\mathbb{N}}_t(a_s) = \begin{cases} a \mid & a_s \in \Sigma_t \\ \varepsilon \mid & a_s \in \Sigma_{\mathbb{N}} \setminus \Sigma_t \end{cases} \text{ and}$$

$$\Theta^{\mathbb{N}} : \Sigma^*_{\mathbb{N}} \to \Sigma^* \text{ with } \Theta^{\mathbb{N}}(a_t) := a \text{ for } a_t \in \Sigma_t \text{ and } t \in \mathbb{N}.$$

*The* iterated shuffle product $P^{\sqcup\!\sqcup}$ *of $P$ is now defined by*

$$P^{\sqcup\!\sqcup} := \Theta^{\mathbb{N}}[\bigcap_{t \in \mathbb{N}} (\tau^{\mathbb{N}}_t)^{-1}(P \cup \{\varepsilon\})] \text{ for } P \subset \Sigma^*.$$

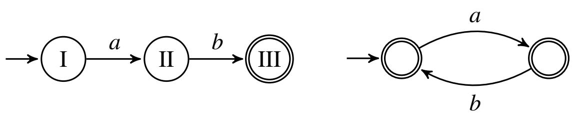
It is easy to see that this is equivalent to the definition from [21] above. Let for example $P = \{ab\}$. Now, according to [21], the word $w = aabb$ is a shuffle of two words $w_1$, $w_2 \in P$ because two colors, namely 1 and 2, can be used to colour the word $aabb$ so that $w_1 = w_2 = ab \in P$. According to definition 5, $aabb \in P^{\sqcup\!\sqcup}$ because $aabb = \Theta^{\mathbb{N}}(a_1 a_2 b_2 b_1)$ and $\tau^{\mathbb{N}}_1(a_1 a_2 b_2 b_1) = \tau^{\mathbb{N}}_2(a_1 a_2 b_2 b_1) = ab \in P$ and $\tau^{\mathbb{N}}_t(a_1 a_2 b_2 b_1) = \varepsilon$ for $t \in \mathbb{N} \setminus \{1,2\}$.

Following the ideas in [18], we structure cooperations into phases.

**Definition 6** (based on a phase)**.** *A prefix closed language $B \subset \Sigma^*$ is based on a phase $P \subset \Sigma^*$, iff $B = \mathrm{pre}(P^{\sqcup\!\sqcup} \cap B)$.*

If $B$ is based on $P$, then $B \subset \mathrm{pre}(P^{\sqcup\!\sqcup}) = (\mathrm{pre}(P))^{\sqcup\!\sqcup}$ and $B = \mathrm{pre}(P)^{\sqcup\!\sqcup} \cap B$.

Let for example $P = \{ab\}$ be given by the Automaton $\mathbb{P}$ in Figure 8(a) and $B$ be given by the automaton $\mathbb{B}$ in Figure 8(b). Then $P^{\sqcup\!\sqcup} \cap B = \{ab\}^*$. This implies that $B$ is based on $P$.



(a) Automaton $\mathbb{P}$ for P= {ab}  (b) Automaton $\mathbb{B}$ for $B$

Figure 8. Automata $\mathbb{P}$ and $\mathbb{B}$

Generally, each $B$ is based on infinitely many phases. If $B$ is based on $P$, then $B$ is based on $P'$ for each $P' \supset P$. Each $B \subset \Sigma^*$ is based on $\Sigma$ because $\Sigma^{\sqcup\sqcup} = \Sigma^*$. Figure 9 shows how we use phases to structure cooperations. The appropriate phases for our purposes as well as *closed behaviours* (words, in which all phases are completed) will be discussed in Section VI.

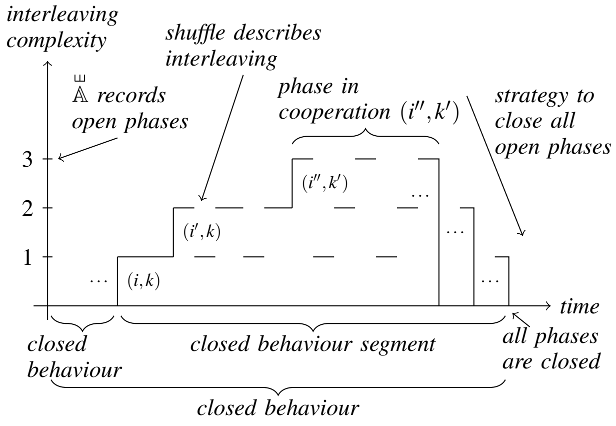


Figure 9. Phases and closed behaviours

We will now provide an automaton representation $\overset{\sqcup\sqcup}{\mathbb{A}}$ for $P^{\sqcup\sqcup}$, which will illustrate "how a language $B$ is based on a phase $P$". Let $P \subset \Sigma^*$ and $\mathbb{A} = (\Sigma, Q, \Delta, q_0, F)$ with $\Delta \subset Q \times \Sigma \times Q$, $q_0 \in Q$ and $F \subset Q$ be an (not necessarily finite) automaton that accepts $P$. To exclude pathological cases we assume $\varepsilon \notin P \neq \emptyset$. A consequence of this is in particular that $q_0 \notin F$. Let $\mathbb{N}_0^Q$ denote the set of all functions from $Q$ in $\mathbb{N}_0$. For the construction of $\overset{\sqcup\sqcup}{\mathbb{A}}$ the set $\mathbb{N}_0^Q$ plays a central role. In $\mathbb{N}_0^Q$ we distinguish the following functions:

$$0 \in \mathbb{N}_0^Q \text{ with } 0(x) = 0 \text{ for each } x \in Q,$$

and for $q \in Q$ the function

$$1_q \in \mathbb{N}_0^Q \text{ with } 1_q(x) = \begin{cases} 1 \mid & x = q \\ 0 \mid & x \in Q \setminus \{q\} \end{cases}.$$

As usual for numerical functions, a partial order as well as addition and partial subtraction are defined.

For $f, g \in \mathbb{N}_0^Q$ let $f \geqslant g$ iff $f(x) \geqslant g(x)$ for each $x \in Q$, $f + g \in \mathbb{N}_0^Q$ with $(f+g)(x) := f(x) + g(x)$ for each $x \in Q$, and for $f \geqslant g$, $f - g \in \mathbb{N}_0^Q$ with $(f-g)(x) := f(x) - g(x)$ for each $x \in Q$.

The key idea of $\overset{\sqcup\sqcup}{\mathbb{A}}$ is, to record in the functions of $\mathbb{N}_0^Q$ how many open phases are in each state $q \in Q$ respectively. Its state transition relation $\overset{\sqcup\sqcup}{\Delta}$ is composed of four subsets whose elements describe (a) the entry into a new phase, (b) the transition within an open phase, (c) the completion of an open phase, (d) the entry into a new phase with simultaneous completion of this phase. With these definitions we now define the *shuffle automaton* $\overset{\sqcup\sqcup}{\mathbb{A}}$.

**Definition 7** (shuffle automaton)**.**
*The* shuffle automaton $\overset{\sqcup\sqcup}{\mathbb{A}} = (\Sigma, \mathbb{N}_0^Q, \overset{\sqcup\sqcup}{\Delta}, 0, \{0\})$ *w.r.t.* $\mathbb{A}$ *is an automaton with infinite state set* $\mathbb{N}_0^Q$*, the initial state* 0*, which is the only final state and*

$$\begin{aligned}
\overset{\sqcup\sqcup}{\Delta} := &\{(f,a,f+1_p) \in \mathbb{N}_0^Q \times \Sigma \times \mathbb{N}_0^Q \mid \\
&\qquad (q_0,a,p) \in \Delta \textit{ and it exists } (p,x,y) \in \Delta\} \cup \\
&\{(f,a,f+1_p-1_q) \in \mathbb{N}_0^Q \times \Sigma \times \mathbb{N}_0^Q \mid \\
&\qquad f \geqslant 1_q, (q,a,p) \in \Delta \textit{ and it exists } (p,x,y) \in \Delta\} \cup \\
&\{(f,a,f-1_q) \in \mathbb{N}_0^Q \times \Sigma \times \mathbb{N}_0^Q \mid \\
&\qquad f \geqslant 1_q, (q,a,p) \in \Delta \textit{ and } p \in F\} \cup \\
&\{(f,a,f) \in \mathbb{N}_0^Q \times \Sigma \times \mathbb{N}_0^Q \mid (q_0,a,p) \in \Delta \textit{ and } p \in F\}.
\end{aligned}$$

*Accepting of a word* $w \in \Sigma^*$ *is defined as usual [23].*

Generally $\overset{\sqcup\sqcup}{\mathbb{A}}$ is a non-deterministic automaton with an infinite state set. In the literature, such automata are called multicounter automata [21] and it is known that they accept the iterated shuffle products [26]. For our purposes, deterministic computations of these automata are very important. To analyse these aspects more deeply we use our own notation and proof of the main theorems. In [18], it is shown that $\overset{\sqcup\sqcup}{\mathbb{A}}$ accepts $P^{\sqcup\sqcup}$.

Let for example $P = \{ab\}$ (cf. Figure 8(a)). Then the states $f : Q \to \mathbb{N}_0$ of the automaton $\overset{\sqcup\sqcup}{\mathbb{P}}$ are described by the sets $\{(q,n) \in Q \times \mathbb{N}_0 | f(q) = n \neq 0\}$.

$$\emptyset \xrightarrow{a} \{(\mathrm{II},1)\} \xrightarrow{a} \{(\mathrm{II},2)\} \xrightarrow{b} \{(\mathrm{II},1)\} \xrightarrow{b} \emptyset$$

is the only computation of $aabb \in P^{\sqcup\sqcup}$ in $\overset{\sqcup\sqcup}{\mathbb{P}}$; it is an accepting computation.

**Example 3.** *Let* $L$ *be defined by the automaton* $\mathbb{L}$ *in Figure 5(a) and* $P \subset \Sigma^+$ *be defined by the automaton* $\mathbb{P}$ *in Figure 7, then* $L \cap P^{\sqcup\sqcup}$ *is accepted by the product automaton of* $\mathbb{L}$ *and* $\overset{\sqcup\sqcup}{\mathbb{P}}$ *that is given in Figure 10.*

*This automaton is strongly connected and isomorphic to* $\mathbb{L}$ *(without considering final states), which proves that* $L$ *is based on phase* $P$*. The states* $(7, \{(\mathrm{VI},1), (\mathrm{II},1)\})$ *and* $(8, \{(\mathrm{VI},1), (\mathrm{III},1)\})$ *show that* $L$ *is "in this states involved in two phases".*

Note that this product automaton, as well as the product automaton in Figure 11(b) and 12(b), is finite and deterministic.

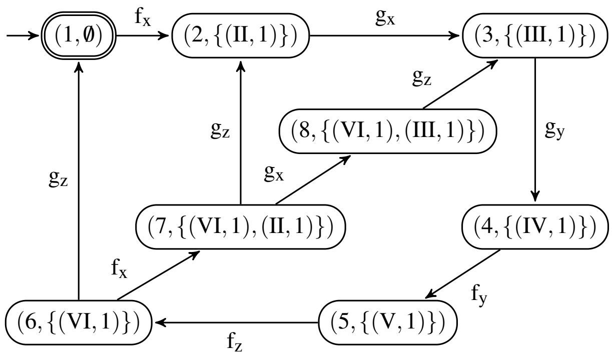


Figure 10. Product automaton of $\mathbb{L}$ and $\overset{\sqcup}{\mathbb{P}}$

## VI. Sufficient Conditions for a Class of Liveness Properties

The following definition is the key to sufficient conditions for strong uniformly parameterised always-eventually properties.

**Definition 8** (set of closed behaviours). *Let $B,M \subset \Sigma^*$. $M$ is a* set of closed behaviours *of $B$, iff $x^{-1}(B) = B$ for each $x \in B \cap M$.*

In Figure 10, the initial state $(1,\emptyset)$ is the only final state of that strongly connected product automaton, so $P^{\sqcup}$ is a set of closed behaviours of $L$.

Now, we get a sufficient condition for uniformly parameterised always-eventually properties.

**Theorem 1.** *Let $I$, $K$, $\mathring{I}$ and $\mathring{K}$ be finite index sets with $|\mathring{I}| \le |I|$ and $|\mathring{K}| \le |K|$. Let $\mathscr{L}_{IK}$ be a uniformly parameterised system of cooperations and let $\mathscr{C}_{IK} \subset \Sigma^*_{IK}$ be a set of closed behaviours of $\mathscr{L}_{IK}$, such that $\mathscr{L}_{IK} = pre(\mathscr{L}_{IK} \cap \mathscr{C}_{IK})$.*

*If $lim(\widehat{\mathscr{L}_{\mathring{I}\mathring{K}}})$ approximately satisfies $(\widehat{\Sigma^*_{\mathring{I}\mathring{K}}\mathring{M}})^{\omega}$, with $\mathring{M} \subset \Sigma^+_{\mathring{I}\mathring{K}}$, then*

$$lim(\widehat{\mathscr{L}_{IK}}) \text{ approximately satisfies } \mathscr{A}^{\mathring{M}}_{IK}.$$

For the proof of Theorem 1 see the appendix. The following theorem gives a set of closed behaviours of $\mathscr{L}_{IK}$.

**Theorem 2.** *Let $P^{\sqcup}$ be a set of closed behaviours of $L$ and let $\pi_{\Phi}(P^{\sqcup})$ resp. $\pi_{\Gamma}(P^{\sqcup})$ be a set of closed behaviours of $SF$ resp. $SG$, then*

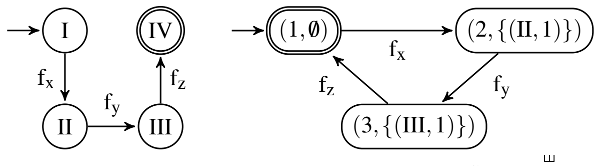
$$\mathscr{C}_{IK} := \bigcap_{(i,k)\in I\times K} (\pi^{IK}_{ik})^{-1}(P^{\sqcup})$$

*is a set of closed behaviours of $\mathscr{L}_{IK}$.*

Theorem 2 is proven in [7]. We now show that $\pi_{\Phi}(P^{\sqcup})$ is a set of closed behaviours of $SF$, which is given in Figure 5(b). The automaton $\mathbb{PF}$ in Figure 11(a) is the minimal automaton of $\pi_{\Phi}(P) \subset \Phi^+$.

By Theorem 3, which is given in the appendix

$$SF \cap \pi_{\Phi}(P^{\sqcup}) = SF \cap (\pi_{\Phi}(P))^{\sqcup}.$$

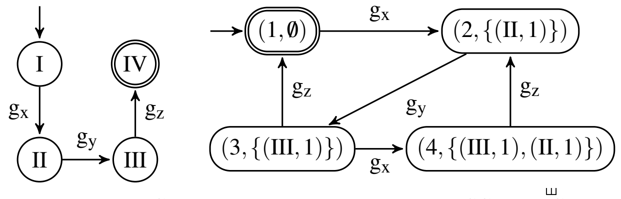


(a) Automaton $\mathbb{PF}$ (b) Product automaton of $\mathbb{SF}$ and $\overset{\sqcup}{\mathbb{PF}}$

Figure 11. Automaton $\mathbb{PF}$ and product automaton of $\mathbb{SF}$ and $\overset{\sqcup}{\mathbb{PF}}$

So, $SF \cap \pi_{\Phi}(P^{\sqcup})$ is accepted by the product automaton of $\mathbb{SF}$ and $\overset{\sqcup}{\mathbb{PF}}$ that is depicted in Figure 11(b). By the same argument as for the product automaton of $\mathbb{L}$ and $\overset{\sqcup}{\mathbb{P}}$ $SF$ is based on $\pi_{\Phi}(P)$, and $\pi_{\Phi}(P^{\sqcup})$ is a set of closed behaviours of $SF$.

Likewise, the automaton $\mathbb{PG}$ in Figure 12(a) is the minimal automaton of $\pi_{\Gamma}(P) \subset \Gamma^+$, $SG \cap \pi_{\Gamma}(P^{\sqcup})$ is accepted by the product automaton of $\mathbb{SG}$ and $\overset{\sqcup}{\mathbb{PG}}$ in Figure 12(b), $SG$ is based on $\pi_{\Gamma}(P)$, and $\pi_{\Gamma}(P^{\sqcup})$ is a set of closed behaviours of $SG$.



(a) Automaton $\mathbb{PG}$ (b) Product automaton of $\mathbb{SG}$ and $\overset{\sqcup}{\mathbb{PG}}$

Figure 12. Automaton $\mathbb{PG}$ and product automaton of $\mathbb{SG}$ and $\overset{\sqcup}{\mathbb{PG}}$

So, by Figure 10, 11(b) and 12(b) all assumptions of Theorem 2 are fulfilled. (6)

Now to apply Theorem 1 together with Theorem 2 it remains to find conditions such that each $u \in \mathscr{L}_{IK}$ is prefix of some $v \in \mathscr{L}_{IK} \cap \mathscr{C}_{IK}$. This set of closed behaviours $\mathscr{C}_{IK}$ consists of all words $w \in \Sigma^*_{IK}$, in which all phases are completed.

Considering example 2, we have shown that each phase is initiated by an $F$-action (Figure 7), each $F$-partner is involved in at most one phase (Figure 11(b)), and, each $G$-partner is involved in at most two phases (Figure 12(b)).

Now to construct for each $u \in \mathscr{L}_{IK}$ a $v \in \mathscr{L}_{IK} \cap \mathscr{C}_{IK}$ with $u \in \text{pre}(v)$ one may imagine that the following strategy could work.

1) For each $G$-partner involved in two phases, complete one of this phases.
2) For each $G$-partner involved in one phases, complete this phase.
3) Complete the phases, where only an $F$-partner is involved in.

If $L$ is based on $P$, $SF$ based on $\pi_\Phi(P)$ and $SG$ based on $\pi_\Gamma(P)$, then by Theorem 3 the assumptions of Theorem 2 imply $L = \mathrm{pre}(L \cap P^{\sqcup\!\sqcup})$, $SF = \mathrm{pre}(SF \cap \pi_\Phi(P^{\sqcup\!\sqcup}))$ and $SG = \mathrm{pre}(SG \cap \pi_\Gamma(P^{\sqcup\!\sqcup}))$.

This is in [7] the starting point of a more general form of such a "completion (of phases) strategy", where also "success conditions" for that strategy are given. It is shown, that under certain regularity restrictions these conditions can be verified by semi-algorithms based on finite state methods. These restrictions are:

> The product automata as in Figure 10, 11(b) and 12(b) must be finite and deterministic. (7)

We only get semi-algorithms but no algorithms, because the product automata are constructed step by step and this procedure does not terminate if the corresponding product automaton is not finite.

Using (7), (3) and the Theorems 1 and 2, the approximate satisfaction of uniformly parameterised always-eventually properties can be *verified by semi-algorithms based on finite state methods*. This verification method only depends on $L$, $SF$, $SG$, $P$ and $\mathring{M}$ and *doesn't refer to the general index sets $I$ and $K$.*

In [7], it is shown that the success conditions are fulfilled in example 2. So, by (4), Theorem 1, Theorem 2 and (6) in example 2 $\lim(\widehat{\mathscr{L}_{IK}})$ approximately satisfies $\mathscr{A}_{IK}^{\mathring{P}}$ for each finite index sets $I$ and $K$, where $\mathring{P}$ is defined in (5).

## VII. Conclusions and Future Work

The main result of this paper is a finite state verification framework for *uniformly parameterised reliability properties*. The uniformly parameterisation of reliability properties exactly fits to the scalability and reliability issues of complex systems and systems of systems, which are characterised by the composition of a set of identical components, interacting in a uniform manner described by the schedules of the partners.

In this framework, the concept of structuring cooperations into phases enables completion of phases strategies. Consistent with this, corresponding success conditions can be formalised [7], which produce finite state semi-algorithms (independent of the concrete parameter setting) to verify reliability properties of uniformly parameterised cooperations. The next step should be to integrate these semi-algorithms in our SH verification tool [25].

Furthermore, we plan a generalisation of the presented approach to systems whose global behaviour is composed of behavioural patterns. The aim is, to eventually derive a set of construction principles for reliable parameterised systems.

Another future work perspective is the application of the approach presented in this paper to the Security Modeling Framework (SeMF) [27]. In SeMF, beside system behaviour, also local views of agents and agents knowledge about system behaviour are considered.

## Acknowledgement

Roland Rieke developed the work presented here in the context of the project MASSIF (ID 257475) being co-funded by the European Commission within FP7.

## Appendix

### A. Basic Notations

The set of all infinite words over $\Sigma$ is defined by

$$\Sigma^\omega = \{(a_i)_{i\in\mathbb{N}} | a_i \in \Sigma \text{ for each } i \in \mathbb{N}\},$$

where $\mathbb{N}$ denotes the set of natural numbers. On $\Sigma^\omega$ a *left concatenation* with words from $\Sigma^*$ is defined. Let $u = b_1 \dots b_k \in \Sigma^*$ with $k \geq 0$ and $b_j \in \Sigma$ for $1 \leq j \leq k$ and $w = (a_i)_{i\in\mathbb{N}} \in \Sigma^\omega$ with $a_i \in \Sigma$ for all $i \in \mathbb{N}$, then $uw = (x_j)_{j\in\mathbb{N}} \in \Sigma^\omega$ with $x_j = b_j$ for $1 \leq j \leq k$ and $x_j = a_{j-k}$ for $k < j$. For $w \in \Sigma^\omega$ the prefix set $\mathrm{pre}(w) \subset \Sigma^*$ is defined by $\mathrm{pre}(w) = \{u \in \Sigma^* |$ it exists $v \in \Sigma^\omega$ with $uv = w\}$. For $L \subset \Sigma^*$ the *$\omega$-language* $L^\omega \subset \Sigma^\omega$ is defined by $L^\omega = \{(a_i)_{i\in\mathbb{N}} \in \Sigma^\omega |$ it exists a strict monotonically increasing function $f : \mathbb{N} \to \mathbb{N}$ with $a_1 \dots a_{f(1)} \in L$ and $a_{f(i)+1} \dots a_{f(i+1)} \in L$ for each $i \in \mathbb{N}\}$ . $f : \mathbb{N} \to \mathbb{N}$ is called *strict monotonically increasing* if $f(i) < f(i+1)$ for each $i \in \mathbb{N}$.

### B. Proof of Theorem 1

To prove Theorem 1 the following lemma is needed.

**Lemma 1.**

$$\mathscr{L}_{IK} \supset \mathscr{L}_{I'K'} \textit{ for } I' \times K' \subset I \times K.$$

For the proof of Lemma 1 see [18] (proof of Theorem 1).

*Proof:* Proof of Theorem 1.

If $\lim(\widehat{\mathscr{L}_{\mathring{I}\mathring{K}}})$ approximately satisfies $(\widehat{\Sigma_{\mathring{I}\mathring{K}}^{*}}\mathring{M})^\omega$, then by (3) (with $u = \varepsilon$) there exists $v \in \mathscr{L}_{\mathring{I}\mathring{K}}$ with $v^{-1}(\mathscr{L}_{\mathring{I}\mathring{K}}) \cap \mathring{M} \neq \emptyset$. As $\iota_{I'K'}^{\mathring{I}\mathring{K}}$ is an isomorphism

$$\iota_{I'K'}^{\mathring{I}\mathring{K}}(\mathscr{L}_{\mathring{I}\mathring{K}}) = \mathscr{L}_{I'K'} \text{ and}$$
$$(\iota_{I'K'}^{\mathring{I}\mathring{K}}(v))^{-1}(\mathscr{L}_{I'K'}) \cap \iota_{I'K'}^{\mathring{I}\mathring{K}}(\mathring{M}) \neq \emptyset. \tag{8}$$

As $\mathscr{C}_{IK}$ is a set of closed behaviours of $\mathscr{L}_{IK}$ and each $u \in \mathscr{L}_{IK}$ is prefix of some $v \in \mathscr{L}_{IK} \cap \mathscr{C}_{IK}$, there exists $x \in u^{-1}(\mathscr{L}_{IK})$ with $(ux)^{-1}(\mathscr{L}_{IK}) = \mathscr{L}_{IK}$.

By Lemma 1 $\mathscr{L}_{IK} \supset \mathscr{L}_{I'K'}$ for each $I' \subset I$ and $K' \subset K$, so $(ux)^{-1}(\mathscr{L}_{IK}) \supset \mathscr{L}_{I'K'}$.

Now (8) implies

$$(\iota_{I'K'}^{\mathring{I}\mathring{K}}(v))^{-1}((ux)^{-1}(\mathscr{L}_{IK})) \cap \iota_{I'K'}^{\mathring{I}\mathring{K}}(\mathring{M}) \neq \emptyset. \tag{9}$$

As $(\iota_{I'K'}^{\mathring{I}\mathring{K}}(v))^{-1}((ux)^{-1}(\mathscr{L}_{IK})) = (ux\iota_{I'K'}^{\mathring{I}\mathring{K}}(v))^{-1}(\mathscr{L}_{IK})$, (9) and (3) complete the proof of Theorem 1. ■

### C. Homomorphism Theorem for $P^{\sqcup\!\sqcup}$

**Theorem 3** (homomorphism theorem for $P^{\sqcup\!\sqcup}$**).**
*Let $\mu : \Sigma^* \to \Sigma'^*$ be an alphabetic homomorphism, then holds*

$$\mu(P^{\sqcup\!\sqcup}) = (\mu(P))^{\sqcup\!\sqcup}.$$

For the proof of Theorem 3 see [7] (proof of Theorem 6).

## References

[1] S. Bullock and D. Cliff, "Complexity and emergent behaviour in ICT systems," Hewlett-Packard Labs, Tech. Rep. HP-2004-187, 2004.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[3] Z. Benenson, F. Freiling, T. Holz, D. Kesdogan, and L. Penso, "Safety, liveness, and information flow: Dependability revisited," in *Proceedings of the 4th ARCS International Workshop on Information Security Applications*, pp. 56–65.

[4] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "Donar: decentralized server selection for cloud services," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 231–242.

[5] A. Shraer, C. Cachin, A. Cidon, I. Keidar, Y. Michalevsky, and D. Shaket, "Venus: verification for untrusted cloud storage," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, ser. CCSW '10. New York, NY, USA: ACM, 2010, pp. 19–30.

[6] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Trans. Dependable Sec. Comput.*, vol. 1, no. 1, pp. 11–33, 2004.

[7] P. Ochsenschläger and R. Rieke, "Behaviour Properties of Uniformly Parameterised Cooperations," Fraunhofer SIT, Tech. Rep. SIT-TR-2010/2, 2010.

[8] B. Alpern and F. B. Schneider, "Defining Liveness," *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, October 1985.

[9] U. Nitsche and P. Ochsenschläger, "Approximately Satisfied Properties of Systems and Simple Language Homomorphisms," *Information Processing Letters*, vol. 60, pp. 201–206, 1996.

[10] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Computer Security Foundations Symposium, IEEE*, vol. 0, pp. 51–65, 2008.

[11] C. N. Ip and D. L. Dill, "Verifying Systems with Replicated Components in Mur$\varphi$," *Formal Methods in System Design*, vol. 14, no. 3, pp. 273–310, 1999.

[12] F. Derepas and P. Gastin, "Model Checking Systems of Replicated Processes with SPIN," in *Proceedings of the 8th International SPIN Workshop on Model Checking Software (SPIN'01)*, ser. Lecture Notes in Computer Science, M. B. Dwyer, Ed., vol. 2057. Toronto, Canada: Springer, May 2001, pp. 235–251.

[13] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre, "Incremental Verification by Abstraction." in *TACAS*, ser. Lecture Notes in Computer Science, T. Margaria and W. Yi, Eds., vol. 2031. Springer, 2001, pp. 98–112.

[14] S. Basu and C. R. Ramakrishnan, "Compositional analysis for verification of parameterized systems," *Theor. Comput. Sci.*, vol. 354, no. 2, pp. 211–229, 2006.

[15] R. Milner, *Communication and Concurrency*, ser. International Series in Computer Science. NY: Prentice Hall, 1989.

[16] J. C. Bradfield and C. Stirling, "Modal Logics and Mu-Calculi: An Introduction," in *Handbook of Process Algebra*, J. A. Bergstra, A. Ponse, and S. A. Smolka, Eds. Elsevier Science, 2001, ch. 1.4.

[17] T. E. Uribe, "Combinations of Model Checking and Theorem Proving," in *FroCoS '00: Proceedings of the Third International Workshop on Frontiers of Combining Systems*. London, UK: Springer-Verlag, 2000, pp. 151–170.

[18] P. Ochsenschläger and R. Rieke, "Uniform Parameterisation of Phase Based Cooperations," Fraunhofer SIT, Tech. Rep. SIT-TR-2010/1, 2010.

[19] M. Jantzen, "Extending Regular Expressions with Iterated Shuffle," *Theor. Comput. Sci.*, vol. 38, pp. 223–247, 1985.

[20] J. Jedrzejowicz and A. Szepietowski, "Shuffle languages are in P," *Theor. Comput. Sci.*, vol. 250, no. 1-2, pp. 31–53, 2001.

[21] H. Björklund and M. Bojanczyk, "Shuffle Expressions and Words with Nested Data," in *Mathematical Foundations of Computer Science 2007*, 2007, pp. 750–761.

[22] P. Ochsenschläger and R. Rieke, "Security Properties of Self-similar Uniformly Parameterised Systems of Cooperations," in *Proceedings of the 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP)*. IEEE Computer Society, February 2011.

[23] J. Sakarovitch, *Elements of Automata Theory*. Cambridge University Press, 2009.

[24] D. Perrin and J.-E. Pin, *Infinite Words*. Elsevier, 2004, vol. Pure and Applied Mathematics Vol 141.

[25] P. Ochsenschläger, J. Repp, and R. Rieke, "The SH-Verification Tool," in *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*. Orlando, FL, USA: AAAI Press, May 2000, pp. 18–22.

[26] J. Jedrzejowicz, "Structural Properties of Shuffle Automata," *Grammars*, vol. 2, no. 1, pp. 35–51, 1999.

[27] A. Fuchs, S. Gürgens, and C. Rudolph, "Towards a Generic Process for Security Pattern Integration," in *Trust, Privacy and Security in Digital Business, 6th International Conference, TrustBus 2009, Linz, Austria, September 3–4, 2009, Proceedings*. Springer, 2009.

# P9

# TOOL BASED FORMAL MODELLING, ANALYSIS AND VISUALISATION OF ENTERPRISE NETWORK VULNERABILITIES UTILISING ATTACK GRAPH EXPLORATION

| | |
|---|---|
| **Title** | Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration |
| **Authors** | Roland Rieke |
| **Publication** | In *In U.E. Gattiker (Ed.), 13th Annual EICAR Conference*, 2004. |
| **ISBN/ISSN** | ISBN ISBN 87-987271-6-8 |
| **URL** | `http://sit.sit.fraunhofer.de/smv/publications/download/Eicar-2004.pdf` |
| **Status** | Published |
| **Publisher** | EICAR e.V. |
| **Publication Type** | EICAR 2004 Conference CD-rom: Best Paper Proceedings |
| **Copyright** | 2004, EICAR e.V. |
| **Contribution of Roland Rieke** | Author and presenter at the Eicar Conference 2004. |

Table 14: Fact Sheet Publication *P9*

Publication P9 [Rieke, 2004b] addresses the following research questions:

RQ5A *How can exploitation possibilities of networked systems' vulnerabilities be analysed?*

RQ5B *How can attacker behaviour be incorporated into the system model and the analysis?*

RQ5C *Which attacks would not be detected?*

A core concern of critical infrastructure protection is a careful analysis of what parts of the information infrastructure really need protection and what are the concrete threads as well as an evaluation of appropriate protection measures.

In this paper a methodology and a tool for the development and analysis of operational formal models is presented that addresses these issues in the context of network vulnerability analysis. A graph of all possible attack paths is automatically computed from the model of a government or enterprise network, of vulnerabilities, exploits and an attacker strategy. Based on this graph, security properties are specified and verified, abstractions of the graph are computed to visualise and analyse compacted information focussed on interesting aspects of the behaviour and cost-benefit analysis is performed. Survivability comes into play, when system's countermeasures and the behaviour of vital services it provides are also modelled and effects are analysed.

# Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration

*Roland Rieke*
*Fraunhofer - Institute Secure Telecooperation, Germany*

*Mailing Address: Fraunhofer - Institut Sichere Telekooperation, Rheinstrasse 75, D-64295 Darmstadt, Germany ; E-Mail: roland.rieke@sit.fraunhofer.de*

**Descriptors**
*critical infrastructure protection, attack simulation, verification tool, security properties, survivability analysis, cost-benefit analysis, intrusion detection, countermeasure evaluation, critical services, risk assessment*

# Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration

## Abstract

*A core concern of critical infrastructure protection is a careful analysis of what parts of the information infrastructure really need protection and what are the concrete threads as well as an evaluation of appropriate protection measures.*

*In this paper a methodology and a tool for the development and analysis of operational formal models is presented that addresses these issues in the context of network vulnerability analysis.*

*A graph of all possible attack paths is automatically computed from the model of a government or enterprise network, of vulnerabilities, exploits and an attacker strategy.*

*Based on this graph, security properties are specified and verified, abstractions of the graph are computed to visualise and analyse compacted information focussed on interesting aspects of the behaviour and cost-benefit analysis is performed.*

*Survivability comes into play, when system´s countermeasures and the behaviour of vital services it provides are also modelled and effects are analysed.*

## Introduction

Todays public, government and enterprise networks are facing a cumulation of risks because a multitude of more or less critical vulnerabilities to system security are found every month. At the same time, the published malicious incidents increase in scope and severity. On the other hand, technological advancements in anti-virus software, firewalls and intrusion detection systems provide a broad palette of proactive defence measures for network protection and impact reduction. The increasing complexity of the network structures and possible protection strategies on one hand and the attack possibilities on the other hand require tool based methods, to guide a systematic evaluation and assist the persons in charge with finally determining exactly what really needs protection and which strategy and means to apply.

A typical means by which an attacker tries to break into a network is, to use combinations of basic exploits to get more information or more credentials and to capture more hosts step by step. To find out if there is a combination that enables an attacker to reach critical network resources or block essential services it is required, to analyse all possible sequences of basic exploits so called *attack paths*. It is also important, to find out which protection could block successful attack paths most efficiently or at least detect attack attempts in an early phase.

For this type of vulnerability analysis, an operational formal model is presented that represents the information system and the behaviour of an attacker. In more detail, it comprises a model of the enterprise network structure and configuration including intrusion detection components, a model of vulnerabilities and corresponding basic exploits, a model of attacker capabilities and profile, and optionally a model of the system's countermeasures.

Based on that model, a reachability graph representing the complete system behaviour is automatically computed. Because this graph in the presented application scenario represents all possible attack paths, it is called *attack graph* in the following text. Now security properties can be specified and verified on the computed behaviour of the model.

The applied verification method is based on formal methods and is implemented in the *SH verification tool* (Ochsenschläger et al., 1999, 2000a) that has been adapted and extended to support the presented attack graph analysis methods.

Questions relating to security properties that can be answered by analysing the attack graph include the following:

Q 1 What *security goals* can be *broken* by a combination of a set of basic exploits selected as attacker profile ?

Q 2 Find the biggest *sources of trouble* in the system based on vulnerability-priorities network-structure and possible attack-patterns. Is there a *critical host or vulnerability* on all paths to some attacker goal ?

Q 3 Quick check of "*am I affected*" by a newly found vulnerability and what new attack-combinations/patterns are possible when adding this vulnerability ?

Q 4 What are the *effects of changes* to the network configuration on overall vulnerability ?

If the model additionally includes specifications of intrusion detection components, then their behaviour and required coaction to recognise attacks, even when evidence is scattered over several hosts, can be analysed.

Common questions concerning intrusion detection are:

Q 5 What *attacks* are *detected* ?

Q 6 What are the *effects of changes* to intrusion detection systems on overall detection of attacks ?

Abstractions of the attack graph can be computed to visualise and analyse compacted information focussed on interesting aspects of the behaviour. The mappings used to compute the abstracted behaviour have to be *property preserving*, to assure that properties are *transported* as desired from a lower to a higher level of abstraction and no critical behaviour is hidden by the mapping.

Aspects that can be visualised using appropriate abstractions on the attack graph are for example:

Q 7 How does the attack graph look like when only attacks that give the attacker new root access are shown (focussing on *gain of credentials*) ?

Cost-benefit analysis can be performed based on costs assigned to the atomic exploits representing level of effort for the attacker and benefits regarding relative importance of the captured hosts. Typical questions concerning cost-benefit are:

Q 8 What is the attack with the *least costs* breaking a given security property ?

Q 9 How much *impact* can an attacker produce given that he applies a given set of atomic exploits ?

Q 10 What is the *optimal position* of given intrusion detection systems regarding cost benefit balance ?

Liveness (in this context often called survivability) comes into play, if part of the behaviour of the enterprise network is also modelled. Analysing effects of countermeasures the system performs under attack or the behaviour of vital services it provides is possible. Careful modelling on an adequate abstraction level is required here to avoid typical state space explosion problems.
A typical liveness questions is:

Q 11 Is a *client* still able to *get answers* from a DB-server when the enterprise network is under attack ?

Some remarks on the remainder of this paper:

The first step in critical infrastructure protection is, to identify the organisation's critical infrastructures and to determine the threats against those infrastructures. This process is described in the next section particularly with regard to network vulnerability analysis. For this purpose, the components to be specified for modelling an attack scenario are described in detail.

The next step in critical infrastructure protection is, to analyse the vulnerabilities of the threatened infrastructure, to assess the risks of degradation or loss of a critical resources as well as to evaluate the effects of the application of countermeasures where risk is unacceptable. To support that process in the given context, in the subsequent section a methodology for the analysis of an attack graph is presented that helps to reveal complex attack combinations and supports the systematic evaluation of possible solutions to minimise risk with given resources.

In the last section an example scenario is presented and the dynamic behaviour of different variants is analysed. Finally some related work is commented, conclusions from this work are drawn and further research goals are sketched.

## Modelling an attack scenario

In this section the information model used and the formal analysis and verification methods and the tool are described, the required specifications are explained in detail and the computation of the attack graph is outlined. Figure 1 shows an overview of the components used to specify the model of the enterprise network system under attack.

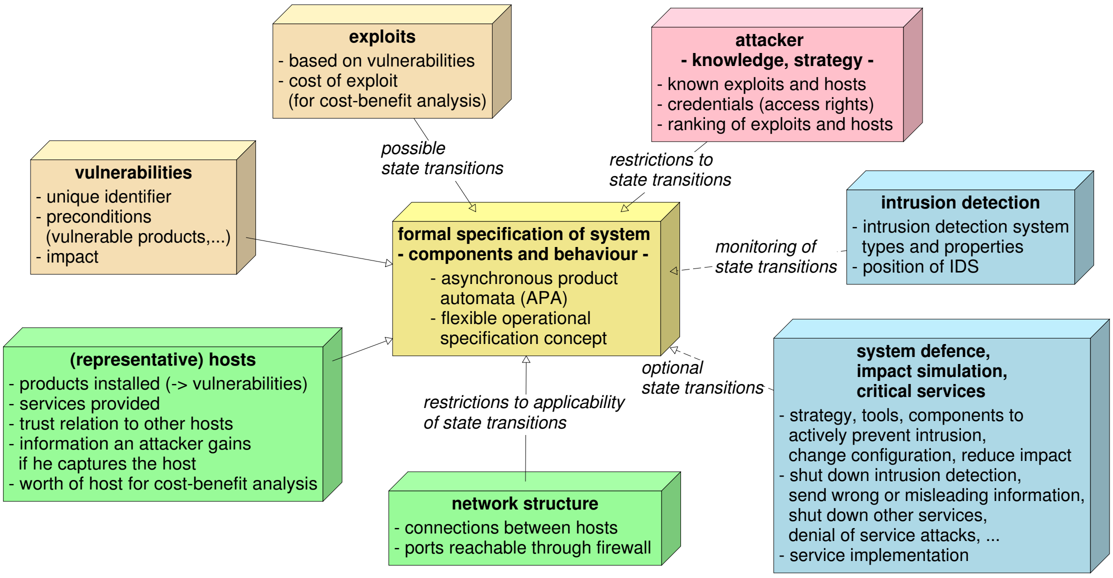

exploits
- based on vulnerabilities
- cost of exploit (for cost-benefit analysis)
attacker
- knowledge, strategy -
- known exploits and hosts
- credentials (access rights)
- ranking of exploits and hosts
vulnerabilities
- unique identifier
- preconditions (vulnerable products,...)
- impact
possible state transitions
restrictions to state transitions
formal specification of system
- components and behaviour -
- asynchronous product automata (APA)
- flexible operational specification concept
intrusion detection
- intrusion detection system types and properties
- position of IDS
monitoring of state transitions
(representative) hosts
- products installed (-> vulnerabilities)
- services provided
- trust relation to other hosts
- information an attacker gains if he captures the host
- worth of host for cost-benefit analysis
optional state transitions
restrictions to applicability of state transitions
system defence, impact simulation, critical services
- strategy, tools, components to actively prevent intrusion, change configuration, reduce impact
- shut down intrusion detection, send wrong or misleading information, shut down other services, denial of service attacks, ...
- service implementation
network structure
- connections between hosts
- ports reachable through firewall

Figure 1. Components of the model

## Information model

To model the enterprise network, the vulnerabilities and the intrusion detection systems, a data model loosely resembling the M2D2 information model (Morin et al., 2002) is used. M2D2 is a formally defined model for information related to the characteristics of the monitored information system, information about the vulnerabilities, information about the security tools used for the monitoring, and information about the events observed. Appropriate parts of this model are adopted and supplemented by concepts needed for description of exploits, attacker knowledge and strategy and information for cost benefit analysis.

### Modelling hosts and network topology

The set of all hosts of the information system consists of the union of the hosts of the enterprise network and the hosts of the attacker(s).

A somewhat abstracted view is used for the representation of network topology including firewalls in the information model. A relation stating what port on what host is reachable from one another is used as network model. The model is very flexible, so that this implicit representation may be changed to a more explicit representation of firewalls easily if this turns out to be useful.

### Modelling products, vulnerabilities and host configurations

Following the M2D2 model *products* are the primary entities that are vulnerable. A *host configuration* is a subset of products that is installed on that host and *affects* is a relation between vulnerabilities and sets of products that are affected by a vulnerability. A host is vulnerable if its configuration is a superset of a vulnerable set of products. Additionally to the installed products a host configuration contains information about what services are currently running and on what ports they are listening.

The vulnerabilities are represented in form of specifications representing a (sub)set of *common vulnerabilities and exposures CVE/CAN* that *MITRE* (see http://cve.mitre.org/) provides to support standardisation of names for all publicly known vulnerabilities and security exposures. These specifications additionally include preconditions about the target host as well as network preconditions and describe effects that the vulnerabilities have on the attacker and possibly on the network and target host.

### Representative hosts

When analysing a complex enterprise network one usually faces a state space explosion problem because all possible combinations of exploits on all possible hosts have to be explored. Therefore it is advantageous to subsume all groups of hosts that have the same configuration, run the same products and are reachable with the same restrictions and that exhibit the same behaviour to one representative host for each such group. In the following text the term *host* will be used as a synonym referring to this representative host. What is suggested here, is to have an abstraction layer between the real enterprise network and the network of representative hosts that still contains all relevant attacks but reduces equivalent combinations. This abstraction could also be applied later after analysing the complete behaviour of the system by using an appropriate mapping but analysis takes much longer then because all sequences of possible combinations have to be computed.

**Summarising representative hosts**
An extension of the above sketched strategy (if the network is still too big for analysis) is to summarise hosts that are reachable with the same restrictions and add up their vulnerabilities to create a representative host with merged vulnerabilities of all summarised hosts. In this case some attacks may be found that are not possible in the real network and the decision if this approximation of system behaviour is good enough for analysis is up to the modeller. A strategy could be, to start with only one representative host per operating system that is configured to have installed all vulnerable products that the enterprise uses (for that operating system) and after that analysis go to a finer granularity as long as the computed state space is still manageable.

**Automated generation of formal specifications ?**
Note that it would be desirable to have an automated generation of formal specifications of system configuration directly derived from the output that network scanner tools like Nessus (see http://www.nessus.org/) provide.

Furthermore vulnerability specifications could be derived from vulnerability database information that for instance *ICAT* (see http://icat.nist.gov/) provides. First step would be to find a good structure and means for a formal description of vulnerabilities that can be used to collect a database of all known vulnerabilities. An agreed upon formal (and tool readable) description of intruder/host/service/network-preconditions and effects of exploitation would have to be developed. An international project like the CAMDIER proposal (Gattiker et al., 2003) might tackle such a task.

## Operational specification of the behaviour

The modelling of the behaviour of the given information model is based on *asynchronous product automata* (APA), a flexible operational specification concept for cooperating systems (Gürgens et al., 2002a). An APA consists of a family of so called *elementary automata* communicating by common components of their state (shared memory). APA are formally defined in figure 2.

Formally an *Asynchronous Product Automaton* consists of a family of *State Sets* $Z_S, S \in \mathbb{S}$, a family of *Elementary Automata* $(\Phi_e, \Delta_e), e \in \mathbb{E}$ and a *Neighbourhood Relation* $N : \mathbb{E} \longrightarrow \mathcal{P}(\mathbb{S})$; $\mathcal{P}(X)$ is the power set of X and $\mathbb{S}$ and $\mathbb{E}$ are index sets with the names of state components and elementary automata. For each Elementary Automaton $(\Phi_e, \Delta_e)$ with *Alphabet* $\Phi_e$, $\Delta_e \subseteq \bigtimes_{S\in N(e)}(Z_S) \times \Phi_e \times \bigtimes_{S\in N(e)}(Z_S)$ is its *State Transition Relation*. For each element of $\Phi_e$ the state transition relation $\Delta_e$ defines state transitions that change only the state components in $N(e)$. An APA's (global) *States* are elements of $\bigtimes_{S\in\mathbb{S}}(Z_S)$. To avoid pathological cases it is generally assumed that $\mathbb{S} = \bigcup_{e\in\mathbb{E}}(N(e))$ and $N(e) \neq \emptyset$ for all $e \in \mathbb{E}$. Each APA has one *Initial State* $s_0 = (q_{0S})_{S\in\mathbb{S}} \in \bigtimes_{S\in\mathbb{S}}(Z_S)$. In total, an APA $\mathbb{A}$ is defined by $\mathbb{A} = ((Z_S)_{S\in\mathbb{S}}, (\Phi_e, \Delta_e)_{e\in\mathbb{E}}, N, s_0)$.

The behaviour of an APA is represented by all possible sequences of state transitions starting with initial state $s_0$. The sequence $(s_0,(e_1,a_1),s_1)(s_1,(e_2,a_2),s_2)(s_2,(e_3,a_3),s_3)\ldots$ with $a_i \in \Phi_{e_i}$ represents one possible sequence of actions of an APA.

State transitions $(s,(e,a),\bar{s})$ may be interpreted as labeled edges of a directed graph whose nodes are the states of an APA: $(s,(e,a),\bar{s})$ is the edge leading from $s$ to $\bar{s}$ and labeled by $(e,a)$. The subgraph reachable from the node $s_0$ is called the *reachability graph* of an APA.

Figure 2. APA definition

**APA state components representing the information model**

The information model described above is specified for the proposed analysis method using the following *APA state components*:

$\boxed{\text{S 1}}$ a specification of the enterprise network topology and host configurations

- reachability of ports on all hosts
- trust relations between hosts
- knowledge available at each host that might be valuable for an attacker as for example ip-numbers of other reachable hosts
- services running on each host
- installed products on each host

$\boxed{\text{S 2}}$ a specification of vulnerabilities of products
⇒ leads to a specification of vulnerabilities for each host $\boxed{\text{S 2'}}$ when combined with products installed on each host specified above

S 3 a specification of attacker knowledge and strategy

S 4 a specification of installed intrusion detection components

S 5 cost benefit ratings, when evaluation about relative values is intended

These specifications are represented in the data structures and initial configuration of the state components in the APA model (see figures 3 and 4).

**Modelling attacker and system behaviour**

*APA state transitions* are used to represent atomic exploits and optionally actions the enterprise network system can take to defend itself or to implement vital services (see figures 3 and 4).

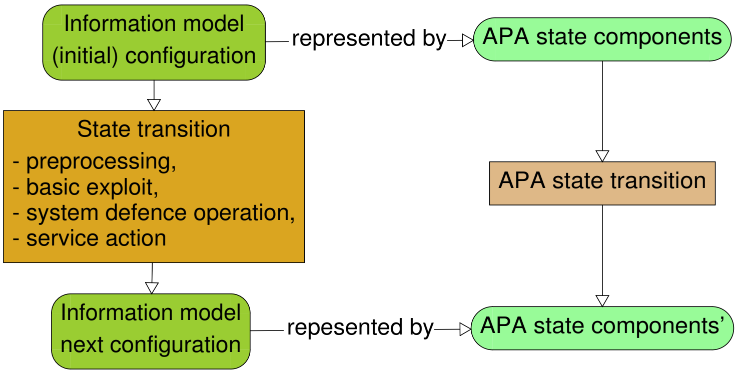


Figure 3. Representation of the information model using APA

**State transition pattern notation for APA**

For the definition of the state transition relation of an elementary automaton $e \in \mathbb{E}$, one has to specify all states of components $\mathrm{C} \in \mathrm{N}(e)$ (state components belonging to $e$) where $e$ is active, i.e. can perform a state transition, and the changes of the states caused by the state transition. APA transition pattern notation is formally defined in (Gürgens et al., 2002b).

A specification of a state transition pattern consists of the *name of the transition pattern*, a *role identifier*, some *predicates* for the conditions to be checked and some *expressions* to describe the changes in the neighbour state components.

A state transition can occur when all expressions are evaluable and all conditions are satisfied. All possible variants of bindings of variables to elements of the state components are generated automatically, so if for example a component contains different hosts and a variable is used to represent a chosen source host and another variable is used for an arbitrary target host of an exploit then all possible combinations of source and target host are computed and further evaluated.

**APA state transition patterns specify attacker and system behaviour**
This paper is primarily concerned with using state transition patterns to model attacker behaviour but as a possible extension other types of state transition patterns are also considered that can be used to model the behaviour of enterprise network components. To reflect the different purposes of the state transitions three different types are distinguished here. They are characterised by the *role* that is associated with the transition type. An instance of a transition furthermore has a *name* to identify it; this can be for example the name of the exploit it specifies.

T Attacker *Exploit* specifications of atomic exploits based on the given vulnerabilities model the actions an attacker can take in arbitrary order; note that more than one attacker can act in that role

T Defence *Operation* specify a model for system defence strategy, tools and components (optional)

T Service *Action* a model of critical services the system provides (optional)

For a state transition pattern T Attacker *Exploit* modelling an exploit, a template structure was developed, so that additional exploits can easily be added following that layout. This template can serve as a basis to develop an automatic mechanism that generates such patterns from a knowledge base containing specifications of the known exploits.

In contrast to the generic nature of T Attacker *Exploit*, the state transition patterns T Defence *Operation* and T Service *Action* are individual for the modelled enterprise network, therefore no specific structure is assumed here. They reflect the state changes triggered by the respective operations.

**Structure of state transition patterns for atomic exploits**
Figure 4 shows a graphical representation of the template for T Attacker *Exploit* including the neighbourhood relation (depicted by the edges) to the state components S1 - S5 (depicted by the circles) listed in the information model above.

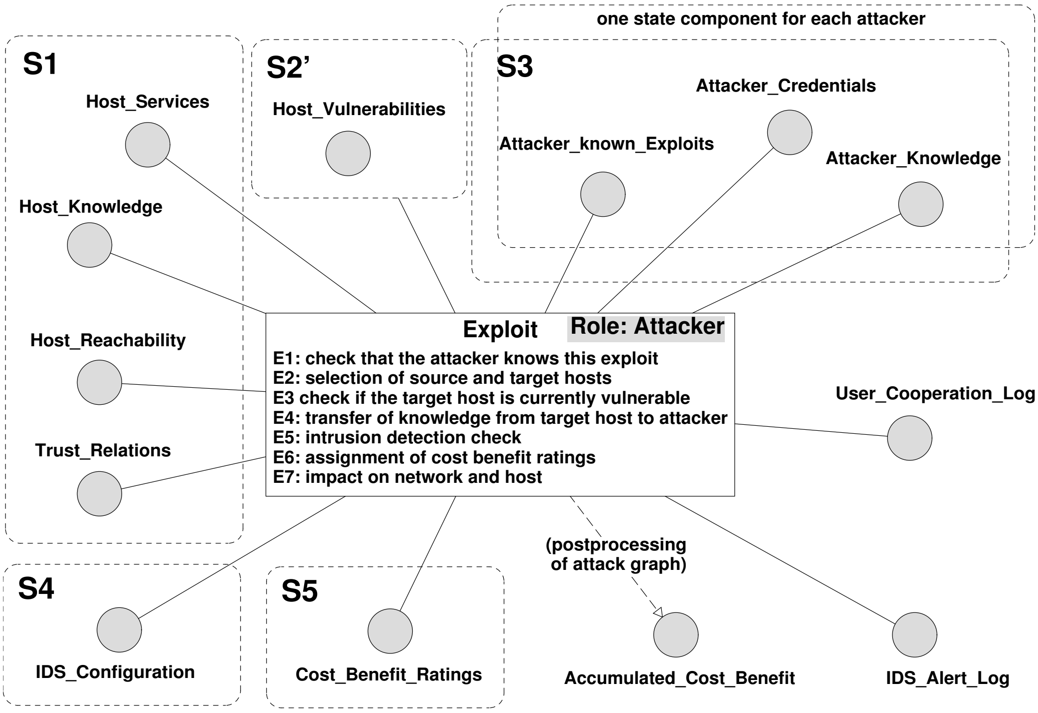

one state component for each attacker
S1
Host_Services
Host_Knowledge
Host_Reachability
Trust_Relations
S2'
Host_Vulnerabilities
S3
Attacker_Credentials
Attacker_known_Exploits
Attacker_Knowledge
Exploit
Role: Attacker
E1: check that the attacker knows this exploit
E2: selection of source and target hosts
E3 check if the target host is currently vulnerable
E4: transfer of knowledge from target host to attacker
E5: intrusion detection check
E6: assignment of cost benefit ratings
E7: impact on network and host
User_Cooperation_Log
(postprocessing
of attack graph)
S4
IDS_Configuration
S5
Cost_Benefit_Ratings
Accumulated_Cost_Benefit
IDS_Alert_Log

Figure 4. Transition pattern template for exploit modelling

According to this template, a state transition pattern modelling an exploit is constructed from the *role identifier*, here *Attacker* the *name* of the transition pattern which is identical to the *name of the exploit* and a body that comprises the following expressions:

E 1 a check that the attacker *knows* this exploit;
this is determined by an initial configuration that can be given directly or computed from a given set of exploits

E 2 a selection of source and target hosts for the exploit

- the source host is chosen from the host set the attacker already has adequate access to (in some cases the target also needs access to the source host for example to read a Trojan web page)
- the target host is chosen so that if the exploit succeeds the attacker will win some credentials or additional knowledge
⇒ induces monotone growing attacker knowledge (no cycles in attack graph), therefore reduces complexity (see also (Ammann et al., 2002))

$\boxed{\text{E 3}}$ a check if the target host is vulnerable as stated in the specification of the vulnerabilities needed by this exploit (possibly multiple different exploits can be based on the same vulnerability)

$\boxed{\text{E 4}}$ the transfer of knowledge from target host to attacker;
it has to be decided how to cope with changing knowledge of the captured host; is knowledge transferred once the host is captured or is a link from attacker to host knowledge inserted, so that the attacker always gets the updated contents ? Is attacker knowledge ever invalidated or is knowledge only valid for a time interval ? These questions influence the attack graph and may lead to cycles.

$\boxed{\text{E 5}}$ an intrusion detection check for that exploit

$\boxed{\text{E 6}}$ an assignment of cost benefit ratings to this exploit

$\boxed{\text{E 7}}$ an expression to implement the additional *impact on the network and host*; for example, to shut down or manipulate a host based intrusion detection system

The vulnerabilities checked in step $\boxed{\text{E 3}}$ above are represented in form of specifications representing the CVE/CAN vulnerabilities. These specifications include preconditions about the target host as well as network preconditions and describe effects that the vulnerabilities have on the attacker and possibly on the network and target host. A vulnerability is described by expressions with the following structure:

$\boxed{\text{V 1}}$ a check if the *target host is configured vulnerable*

- the target host has installed a product or products that are vulnerable with respect to the given vulnerability
- if necessary other preconditions are checked; for example, it could be essential for a vulnerability that a trust relation is established (as for example used in remote shell hosts allow/deny concepts)

$\boxed{\text{V 2}}$ a check if the target *host is currently running the respective products* (for example a vulnerable operating system or server version);
if a user interaction is required this includes a check if the vulnerable product is currently used (for example a vulnerable internet explorer)

$\boxed{\text{V 3}}$ a check for necessary *network preconditions*, including a check if the target host is reachable on the port the vulnerable product is using from the host the attacker selected as source
⇒ this implicitly includes firewall rules (the model could be extended to explicitly model firewalls through extra transitions but this would blow up the state space significantly)

V 4 an expression to cover the *effects for the attacker*; for example, to obtain additional user or root credentials on the target host

V 5 an expression to implement the *direct impact on the network and host*; for example, to shut down a service caused by buffer overflow

**Attacker knowledge and behaviour**

Attacker capabilities are modelled by the knowledge of exploits and hosts and the credentials on the known hosts that constitute the attackers profile. Knowledge of hosts changes during the computation of the attack graph because the attacker might gain new knowledge when capturing hosts. For example, if the attacker captures a portal or a host used as a firewall or a gateway he gets all information this host has. On the other hand, some knowledge may become outdated because the enterprise system changes ip-numbers or other configuration of hosts and reachability. Several different attackers can easily be included because an attacker is modelled as a role not a single instance and the tool can automatically generate multiple instances from one role definition. Optionally it is possible to specify extra transitions modelling an assumed impact an attacker might produce as for example shut down intrusion detection systems, send wrong or misleading information, shut down other services, denial of service attacks or other actions. But all this blows up the computation space and should be carefully used.

**Monotonicity and invalid knowledge**

It is not clear what is the best strategy to cope with dynamically changing configuration of hosts. To try keep the attacker knowledge monotone growing and get an attack graph without loops it is useful to model the knowledge as applicable only for some time interval but then if for example a host could change its ip-address arbitrarily the attack graph always grows with each change.

## Assembling components of the model

The applied specification method based on *asynchronous product automata (APA)* is supported by the *SH verification tool* developed at “Fraunhofer-Institute Secure Telecooperation” (Ochsenschläger et al., 1999, 2000a). This tool provides components for the complete cycle from formal specification to exhaustive validation. The tool has been adapted and extended for the presented field of application.

The project management of the SH verification tool allows to select alternative parts of the specification and automatically “glues” together selected parts of the specified components (see figure 1) to generate a combined model of enterprise network specification, vulnerability and exploit specification and attacker specification. This can be used to answer $\boxed{\text{Q 2}}$, $\boxed{\text{Q 3}}$ and $\boxed{\text{Q 4}}$ (see introduction). A very flexible selection of variants of analysis scenarios is implemented. The components are listed in a project tree and can be (de)activated by mouse-click. So it is easy for example to exchange libraries of specified vulnerabilities and exploits to analyse different versions and combinations of formal models and even compare different computed attack graphs or abstractions thereof in the analysis component of the tool.

### Computation of attack graphs

After an initial configuration is selected, the attack graph (reachability graph) is automatically computed by the SH verification tool according to the definition in figure 2.

Two extra transitions that have turned out to be very useful have been included in the model as preprocessing steps. One computes the vulnerabilities per host from the information on products installed per host and vulnerabilities per product, the other generates a set of known exploits for the attacker(s) from a given algorithm. If for example it is assumed that the attacker knows 3 different exploits, then all combinations of 3 exploits from the set of all specified exploits have to be computed and further analysed.

To stop computation automatically when specified conditions are reached (or invariants are broken), so called break conditions can be specified using regular expressions. A violation of a security property for example, can in many cases be specified as a break condition.

For a quick check if something went wrong with the definition of the model, some statistic information is collected during computation of the graph. It can be used to find out, what state transitions appeared how often and what different values have been assigned to the state components during the computation.

## Analysis of an attack graph

The main purpose of attack graph analysis is, to provide support for the persons in charge to assess the risks and the effects of possible countermeasures for the threatened network infrastructure.

The methodology for the analysis of an attack graph presented here that is outlined in figure 5 supports that process. It assists in revealing complex attack combinations and supports the systematic evaluation of possible solutions to minimise risk with given resources.

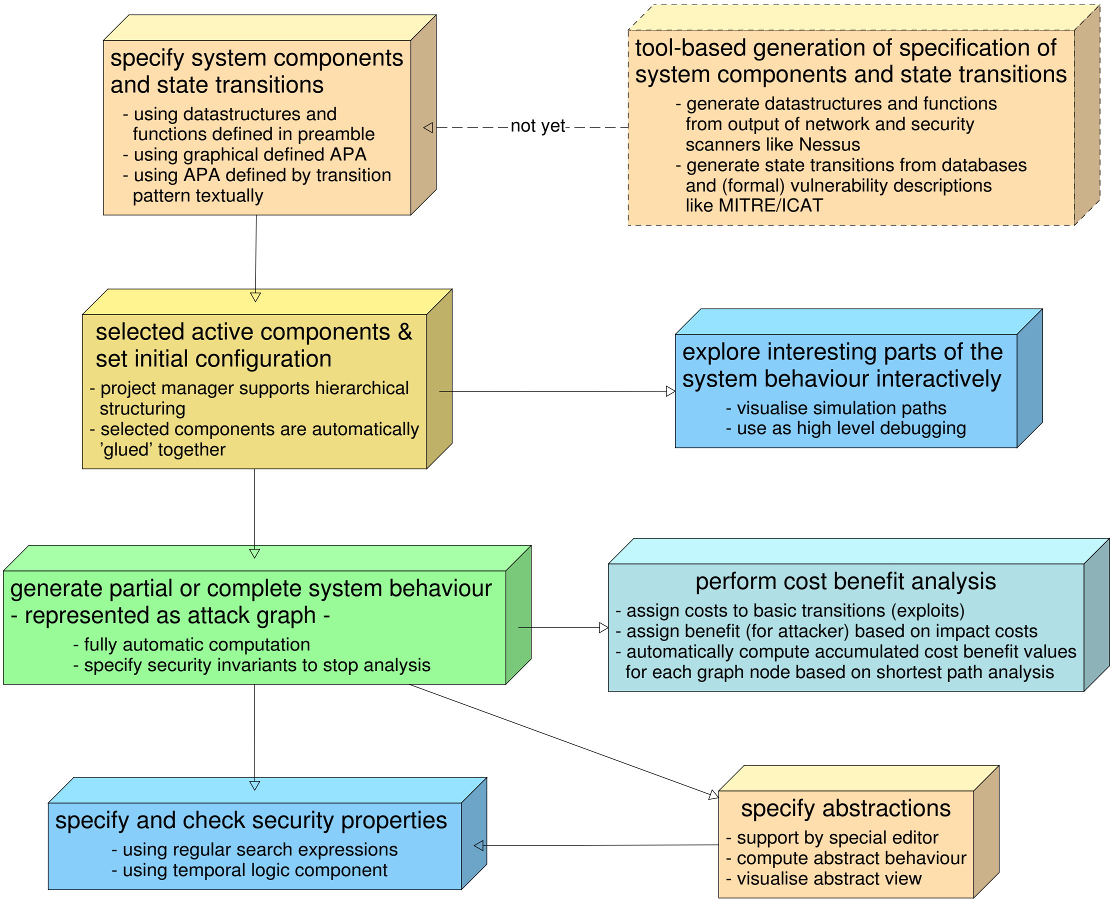

specify system components and state transitions
- using datastructures and functions defined in preamble
- using graphical defined APA
- using APA defined by transition pattern textually
not yet
tool-based generation of specification of system components and state transitions
- generate datastructures and functions from output of network and security scanners like Nessus
- generate state transitions from databases and (formal) vulnerability descriptions like MITRE/ICAT
selected active components & set initial configuration
- project manager supports hierarchical structuring
- selected components are automatically 'glued' together
explore interesting parts of the system behaviour interactively
- visualise simulation paths
- use as high level debugging
generate partial or complete system behaviour - represented as attack graph -
- fully automatic computation
- specify security invariants to stop analysis
perform cost benefit analysis
- assign costs to basic transitions (exploits)
- assign benefit (for attacker) based on impact costs
- automatically compute accumulated cost benefit values for each graph node based on shortest path analysis
specify and check security properties
- using regular search expressions
- using temporal logic component
specify abstractions
- support by special editor
- compute abstract behaviour
- visualise abstract view

Figure 5. Computation and analysis of attack graphs

In the following paragraphs it is shown how to find answers to the questions posed in the introduction through analyses that can be accomplished after an attack graph is successfully computed. Many other interesting evaluations can be performed without question.

## Finding violations of security properties

Security is not a singular property of a system. Depending on precisely what capabilities an attacker has, different properties for the system model have to be proven.

### Formal specification of properties

System properties that are explicitly given by break-conditions can be checked during computation of the attack graph. Alternatively, security properties given in form of search queries, Büchi-automata or temporal logic formulae can be verified after the graph is computed.

### Finding states violating a safety (security) property

If a security property can be specified by a regular expression so that it is possible to check for a violation by inspecting a single node or edge then the property can be proven by a simple “search query” on the reachability graph. Often this can be supported in the model by collecting necessary information during the computation of the graph.

### Model checking

If it is required to inspect some or all paths of the graph to check for the violation of a security property, as it is usually the case for liveness properties, then the temporal logic component of the SH verification tool can be used. Temporal logic formulae can also be checked on the abstract behaviour (under a simple homomorphism). A method for checking approximate satisfaction of properties fits exactly to the built-in simple homomorphism check (Ochsenschläger et al., 1999).

These methods provide appropriate support to answer question $\boxed{\text{Q 1}}$ from the introduction and are also helpful to research into many other questions.

## Abstraction and visualisation of attack graphs

Abstraction capabilities of the SH verification tool support the definition of mappings, summarising or omitting transitions in the attack graph. The result is a view focussed on some interesting aspect of the behaviour of the system. Technically this is implemented as a computation of the minimal automaton for an abstraction of the reachability graph that is specified via alphabetic language homomorphisms (Ochsenschläger et al., 2000b).

It is possible for example, to map multiple exploits with the same effects onto the same subsuming activity like “get-root-access”. This can be used to answer questions like $\boxed{\text{Q 7}}$ from the introduction. Another example is, to omit all exploits that are not detected by some intrusion detection component, in order to get a graph showing only the traces that an attack correlation component would see. Abstractions can also be defined using predicates. It is possible for example, to omit all transitions below a certain cost-benefit ratio using an appropriate predicate.

## Analysing IDS pattern detection

The transition patterns representing atomic exploits are modelled to include the behaviour of intrusion detection components, therefore their behaviour and their coaction to recognise attack pattern can be analysed. This helps to answer the question $\boxed{\text{Q 5}}$ (What attacks are detected ?) from the introduction.

Detections that are directly related to an atomic exploit are visible in the attack graph, because an intrusion detection check $\boxed{\text{E 5}}$ is included in each transition modelling an atomic exploit.

In more complex cases, evidence of attacks against the network is scattered over several atomic exploits on one host or several different hosts. The installed intrusion detection systems therefore have to collect and correlate information from different sources (Krügel and Toth, 2002).

Analysing the attack graph with regard to the required security properties leads to a detection of the paths that violate those properties. Abstraction helps to filter out the information concerning intrusion detection and gives a graph that visualises the correlation that is required to detect these violations. Now a scheme of coaction of intrusion detection components to detect this malicious behaviour or a superordinated component that checks for combined patterns can be designed.

Question $\boxed{\text{Q 6}}$ (What are the effects of changes to intrusion detection systems on overall detection of attacks ?) can be answered by comparing intrusion detection analysis of different attack graphs computed for different configurations selected in the project management component. It is useful to combine several features supported by the SH verification tool to answer this question. To filter out the intrusion detection information, abstractions of the different attack graphs are required. Based on this abstracted behaviour, a comparison of the behaviour of different versions is possible. The tool supports a comparison of those graphs and additionally the results of search queries and model checking helps finding the effects in question, but this task requires careful modelling, abstraction and finding the right properties to check.

## Simulation

If the attacker has too many alternatives or the network is too complex, the state space of the composition of the selected specifications and their complex interplay may become too big to compute the complete behaviour. In this case it is appropriate to inspect selected parts of the state space. Simulation of interesting attack combinations is possible by interactive selection of paths in the visual representation of the part of the attack graph already computed and automatic proceeding in the selected direction. Other variants of simulation are also supported by the tool (for instance *random driven*). The seamless transition between verification and simulation on the same model is a particular strength of the approach presented here.

### Cost benefit analysis

Cost benefit analysis as described in this paragraph is meant as a means to help assess the likely behaviour of an attacker. Cost ratings (from the view of an attacker) can be assigned to each exploit, for example to denote the time it takes for the attacker to execute the exploit or the resources needed to develop an exploit. If not only technical vulnerabilities are modelled but also human weaknesses are considered, then cost could mean for example the money needed to *buy* a password.

Based on these cost assignments, the shortest (least expensive) path from the root of the attack graph to a node representing a successful attack can be computed and visualised. This helps to answer question $\boxed{\text{Q 8}}$ (What is the attack with the *least costs* breaking a given security property ?) from the introduction.

A benefit for the attacker based on the negative impact he achieves can also be assigned, for example to indicate the *worth* regarding relative importance of the captured host.

Summarised costs and benefits can be compared for selected paths or the whole graph. For example searching for the node with the greatest benefit for the attacker answers question $\boxed{\text{Q 9}}$ from the introduction.

Comparing some configurations with available intrusion detection systems placed at different locations and computing attack graphs only for undetected attacks can help to decide what is a better position for the intrusion detection systems when looking at the maximum benefit for the attacker being undetected in the different scenarios (see also $\boxed{\text{Q 10}}$ from the introduction). To find a good coverage of intrusion detection given restricted resources, only relative evaluation of some predefined variants is intended here. It is shown in (Jha et al., 2002) that to decide which minimal set of security measures would guarantee the safety of the system is polynomially equivalent to the minimum hitting set problem (NP-complete).

### Survivability analysis

So far it was assumed that the enterprise network system does not react during an attack. This is in general a useful assumption to keep the graph of the system behaviour manageable. However the following extensions to the model can give valuable insight into related problems.

**Game: System against attacker**
In some cases it is interesting to consider some counter-play of the system. In Germany for example an ip-address for a dsl-connection is allocated dynamically and automatically changed every 24 hours. If for example the hosts of some teleworkers are part of the modelled enterprise system it is useful to check what effect this behaviour has on the attack analysis. Also if an attack is time consuming, it is possible, that it will be detected not only by an intrusion detection system but also possibly by some other security scanner tool or a human administrator checking the given configuration at certain time intervals. It is desirable to augment the model by some counteraction to describe for example a cut of a network connection in critical cases or the reconfiguration of a system.

**Mission critical e-services**
It is often very important, that even when an enterprise network system is under attack, at least some mission critical e-services survive that attack. Therefore it is essential, that it is possible to augment the attack scenario to include actions of the critical e-service and to analyse the extended scenario.

To verify if a given e-service survives an attack, a formal model of its components and their interplay must be added to the system model. The combined model can then be analysed by computing its dynamic behaviour and examining the generated state space. New safety and usually also liveness properties that constitute the required behaviour of the e-service have to be specified and verified. This helps in answering for example question Q 11 from the introduction. A methodology for developing an e-service so that it is robust against attacks has been described in (Rieke, 2003).

Because of the well known state space explosion problem, the extended scenarios have to be specified on a high abstraction level in order be able to compute the complete reachability graph. To find an appropriate abstraction level, it is essential to incorporate the hints given in the previous section concerning representative hosts. One should also consider to summarise similar attacks onto a representative abstract attack. For example only use the abstract attacks “get-user-access”, “get-root-access” and “from-user-to-root-access”.

## Specification and analysis of an example scenario

To illustrate the methods described so far, a small example scenario is given now. The components are specified, the respective attack graph is described and some typical analysis outcome is sketched.

## Scenario specification

Figure 6 shows the example scenario with the enterprise hosts named $\mathit{ms\_host}$, $\mathit{nix\_host}$, $\mathit{portal}$, $\mathit{db\_server}$ located inside the enterprise network and the host $\mathit{telework}$ connected from the internet as well as the host $\mathit{attacker}$. Vulnerabilities of the hosts needed for specification part $\boxed{\text{S 2'}}$ derived from the products installed and the product vulnerabilities are denoted below the host-names.

The installed intrusion detection components for specification part $\boxed{\text{S 4}}$ are depicted in figure 6 by the rhombic nodes. $\mathit{IDS\_type1}$ is a network based system that detects exploits named $\mathit{CAN\_2003\_0693\_ssh\_exploit}$ and $\mathit{rsh\_login}$ attempts. One IDS of that type is installed between the internet and the host $\mathit{portal}$, the other is installed to control the traffic between the $\mathit{portal}$ and the host $\mathit{db\_server}$. Furthermore a host based intrusion detection component $\mathit{IDS\_type2}$ that detects exploits of type $\mathit{CAN\_2002\_0649\_sql\_exploit}$ is installed directly on host $\mathit{db\_server}$.

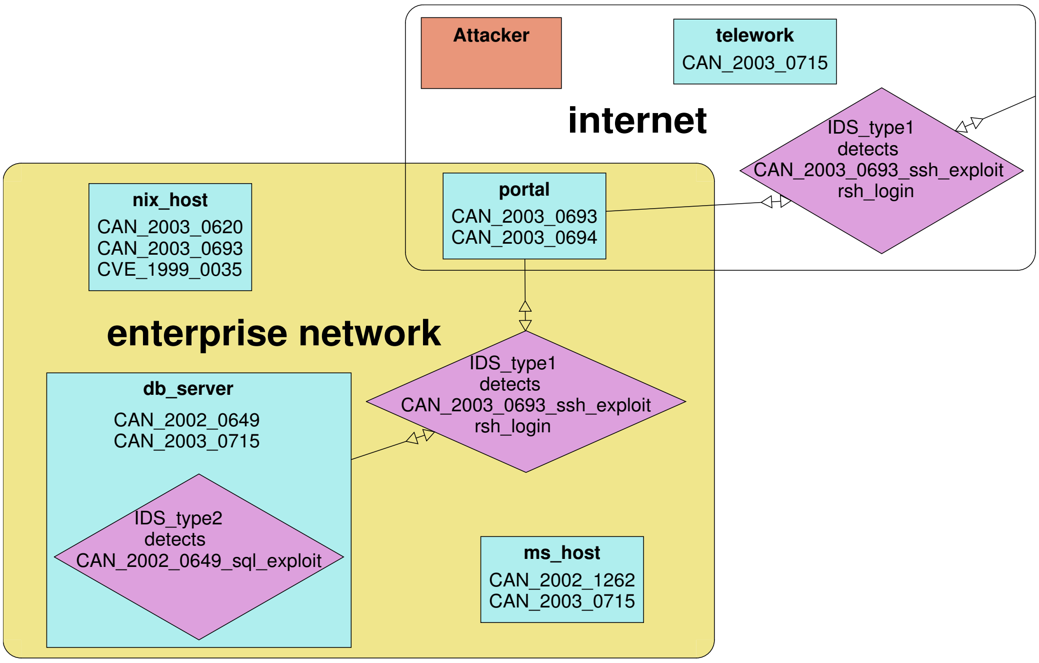


Figure 6. Example scenario

More information for specification part $\boxed{\text{S 1}}$ is provided by the tables in figure 7, showing the reachability of ports on all hosts and the active services. Some abbreviations are used here, namely $\mathit{zone\_internet}$ is an abbreviation for the hosts $\mathit{telework}$, $\mathit{attacker}$, $\mathit{portal}$ and $\mathit{zone\_intern}$ is used for $\mathit{portal}$, nix_host, db_server and ms_host. The abbreviation port_all means reachability for all ports and finally the abbreviation net means physically connected.

Knowledge to be captured is only available on the portal that knows the addresses of all hosts. This could be used for example by the attacker to find out the dynamic allocated address of the telework host, that might be not so well administrated as the enterprise hosts directly connected to the network.

| Host | Service | Port | User |
|---|---|---|---|
| telework | netbios_ssnd | netbios_ssn_port | root |
| nix_host | ftpd | ftp_port | root |
| nix_host | sshd | ssh_port | root |
| nix_host | rshd | rsh_port | root |
| db_server | ftpd | ftp_port | root |
| db_server | rshd | rsh_port | root |
| db_server | sql_res | ms_sql_m_port | db_user |
| ms_host | dcom |  | root |
| ms_host | netbios_ssnd | netbios_ssn_port | root |
| portal | sendmaild | smtp_port | root |
| portal | sshd | ssh_port | root |

| Source Host | Target Host | Port |
|---|---|---|
| zone_internet | zone_internet | port_all |
| zone_all | portal | ssh_port |
| zone_all | portal | smtp_port |
| portal | zone_intern | port_all |
| zone_intern | zone_all | net |
| zone_intern | zone_intern | ftp_port |
| zone_intern | zone_intern | rsh_port |
| zone_intern | zone_intern | ssh_port |
| db_server | ms_host | rpc_port |

Figure 7. Host reachability and installed services

### Attacker profile

It is assumed that the attacker knows all exploits that are specified in detail below, namely CAN_2002_0649_sql_exploit, CAN_2003_0620_man_db_exploit, CAN_2003_0693_ssh_exploit, CAN_2003_0693_ssh_exploit_stealth, CAN_2003_0694_sendmail_exploit, CAN_2003_0715_dcom_exploit, CVE_1999_0035_ftp_exploit and the pseudo exploit rsh_login.

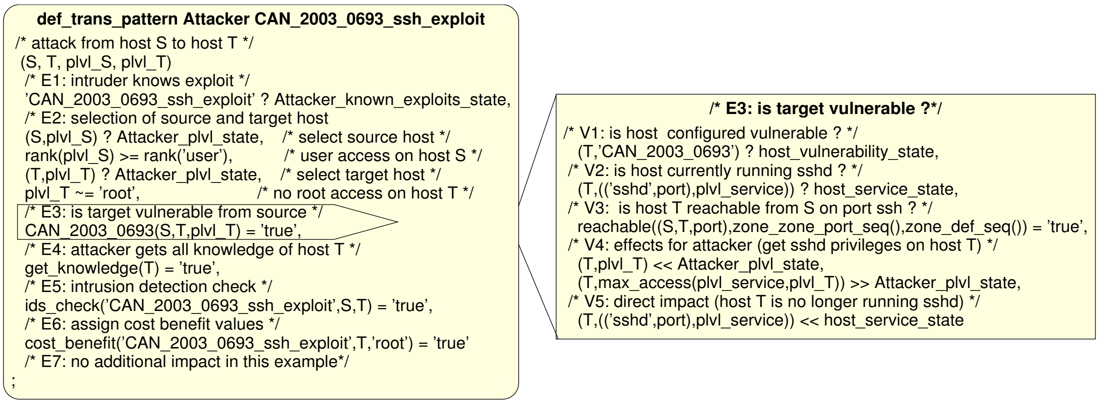
In the initial configuration the attacker has root credentials on the host attacker and no other access. The attacker nows the static addresses of all hosts except the dynamic address of the host telework. The attacker has no other knowledge. This completes the specification part S 3.

### Vulnerabilities and exploits

The vulnerabilities and exploits described below are used in the example scenario. They are not described in detail here; more details are found at *MITRE* (see http://cve.mitre.org/) and *ICAT* (see http://icat.nist.gov/) sites.

Vulnerability CVE_1999_0035, an error in ftpd allowing to read/write arbitrary files is used to manipulate files to establish remote shell trust and this in turn used in combination with the rsh_login which is not a real vulnerability but a weak configuration to get remote access. This old vulnerability has been included because this example was used in some of the papers cited in the section on related work, to make it easier to compare different approaches. The related exploit using this vulnerability is named CVE_1999_0035_ftp_exploit.

The vulnerabilities CAN_2003_0620 (a buffer overflows in man-db) and the related exploit CAN_2003_0620_man_db_exploit, CAN_2003_0693 (a buffer management error in OpenSSH) and the related exploits CAN_2003_0693_ssh_exploit and CAN_2003_0693_ssh_exploit_stealth, CAN_2003_0715 (a heap-based buffer overflow in DCOM) and the related exploit CAN_2003_0715_dcom_exploit, CAN_2003_0694 (a buffer overflow in sendmail) and the related exploit CAN_2003_0694_sendmail_exploit as well as CAN_2002_0649 (buffer overflows in SQL server) and the related exploit CAN_2002_0649_sql_exploit are used to directly get access rights on a remote host. An example of the implementation of an exploit in SH verification tool syntax is given in figure 8.



```
def_trans_pattern Attacker CAN_2003_0693_ssh_exploit
/* attack from host S to host T */
 (S, T, plvl_S, plvl_T)
  /* E1: intruder knows exploit */
  'CAN_2003_0693_ssh_exploit' ? Attacker_known_exploits_state,
  /* E2: selection of source and target host
  (S,plvl_S) ? Attacker_plvl_state,    /* select source host */
  rank(plvl_S) >= rank('user'),          /* user access on host S */
  (T,plvl_T) ? Attacker_plvl_state,    /* select target host */
  plvl_T ~= 'root',                          /* no root access on host T */
  /* E3: is target vulnerable from source */
  CAN_2003_0693(S,T,plvl_T) = 'true',
  /* E4: attacker gets all knowledge of host T */
  get_knowledge(T) = 'true',
  /* E5: intrusion detection check */
  ids_check('CAN_2003_0693_ssh_exploit',S,T) = 'true',
  /* E6: assign cost benefit values */
  cost_benefit('CAN_2003_0693_ssh_exploit',T,'root') = 'true'
  /* E7: no additional impact in this example*/
;
```

```
/* E3: is target vulnerable ?*/
/* V1: is host  configured vulnerable ? */
  (T,'CAN_2003_0693') ? host_vulnerability_state,
 /* V2: is host currently running sshd ? */
  (T,(('sshd',port),plvl_service)) ? host_service_state,
 /* V3:  is host T reachable from S on port ssh ? */
  reachable((S,T,port),zone_zone_port_seq(),zone_def_seq()) = 'true',
 /* V4: effects for attacker (get sshd privileges on host T) */
  (T,plvl_T) << Attacker_plvl_state,
  (T,max_access(plvl_service,plvl_T)) >> Attacker_plvl_state,
 /* V5: direct impact (host T is no longer running sshd) */
  (T,(('sshd',port),plvl_service)) << host_service_state
```

Figure 8. Transition pattern for CAN_2003_0693_ssh_exploit

## Analysis of the scenario

### Attack graph of the example scenario

The computed attack graph for this scenario has 142 nodes and 544 edges. Figure 9 shows a small section of it. The oval nodes depict single states, the rectangular nodes depict states with a hidden subgraph that can be expanded by mouse-click. The red (dotted) nodes mark states where the attacker has been detected by an intrusion detection component.

### Check security properties

As an example for a security property to be checked for the scenario it is assumed that it is essential that an attacker can not gain any access at the db_server. The search query

```
({Attacker_plvl_state:<y>| sfind(('db_server','db_user'),y) =0},,
 {Attacker_plvl_state:<x>| sfind(('db_server','db_user'),x) >0});
```

checks if there are transitions in the graph where the attacker gains access as $db\_user$ at the $db\_server$. For this query 66 matching edges are found.

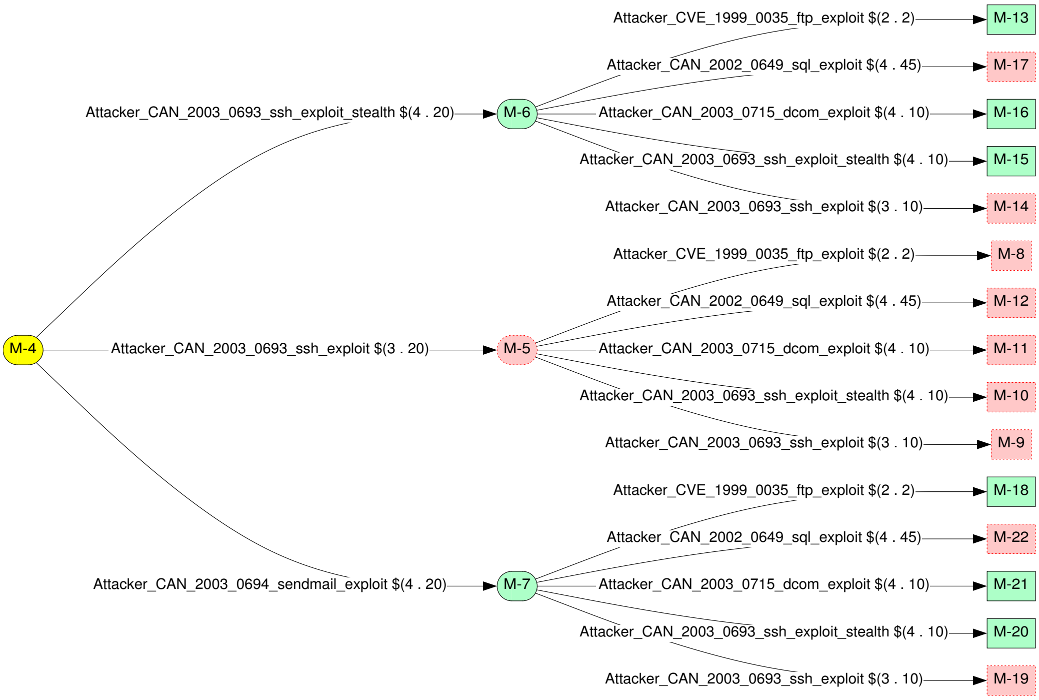

Attacker_CVE_1999_0035_ftp_exploit $(2 . 2)
M-13
Attacker_CAN_2002_0649_sql_exploit $(4 . 45)
M-17
Attacker_CAN_2003_0693_ssh_exploit_stealth $(4 . 20)
M-6
Attacker_CAN_2003_0715_dcom_exploit $(4 . 10)
M-16
Attacker_CAN_2003_0693_ssh_exploit_stealth $(4 . 10)
M-15
Attacker_CAN_2003_0693_ssh_exploit $(3 . 10)
M-14
Attacker_CVE_1999_0035_ftp_exploit $(2 . 2)
M-8
Attacker_CAN_2002_0649_sql_exploit $(4 . 45)
M-12
M-4
Attacker_CAN_2003_0693_ssh_exploit $(3 . 20)
M-5
Attacker_CAN_2003_0715_dcom_exploit $(4 . 10)
M-11
Attacker_CAN_2003_0693_ssh_exploit_stealth $(4 . 10)
M-10
Attacker_CAN_2003_0693_ssh_exploit $(3 . 10)
M-9
Attacker_CVE_1999_0035_ftp_exploit $(2 . 2)
M-18
Attacker_CAN_2002_0649_sql_exploit $(4 . 45)
M-22
Attacker_CAN_2003_0694_sendmail_exploit $(4 . 20)
M-7
Attacker_CAN_2003_0715_dcom_exploit $(4 . 10)
M-21
Attacker_CAN_2003_0693_ssh_exploit_stealth $(4 . 10)
M-20
Attacker_CAN_2003_0693_ssh_exploit $(3 . 10)
M-19

Figure 9. Attack graph of example scenario (small section)

**Maximal impact**

The computed attack graph for the example scenario has 18 so called *dead markings*. In cases where the graph has no loops these are the leafs of the graph. They denote states where no further processing occurs because the attacker has no more applicable atomic exploits available or has already captured all hosts.

Selecting an arbitrary dead marking and let the tool generate a way to the root node produces a path as shown in figure 10. The edge labels denote the atomic exploit chosen in that step as well as the target and source host. The first 2 edges represent state transitions for preprocessing steps as explained in the section on computation of attack graphs. The red (dotted) nodes $M85$, $M$-$122$, $M140$ denote states where the attacker has already been detected. There is much more information available for each transition but this is hidden here by a presentation abstraction to keep the example readable. The numbers after the $-sign are explained in the next paragraph.

Inspecting the attackers knowledge at the dead marking $M140$ shows that he gained root access on hosts attacker, ms_host, nix_host and portal and furthermore he gained db_user access on db_server but none on telework.

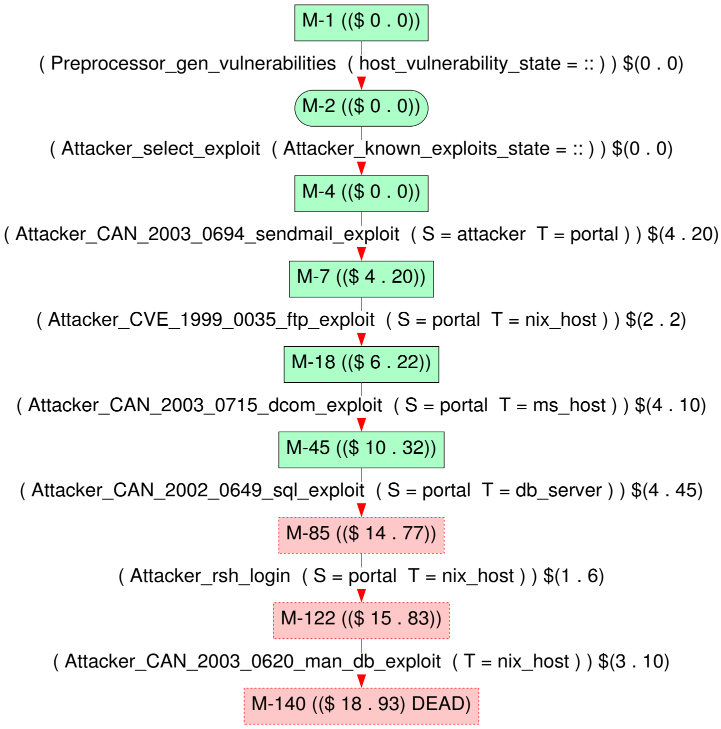

M-1 (($ 0 . 0))
( Preprocessor_gen_vulnerabilities ( host_vulnerability_state = :: ) ) $(0 . 0)
M-2 (($ 0 . 0))
( Attacker_select_exploit ( Attacker_known_exploits_state = :: ) ) $(0 . 0)
M-4 (($ 0 . 0))
( Attacker_CAN_2003_0694_sendmail_exploit ( S = attacker T = portal ) ) $(4 . 20)
M-7 (($ 4 . 20))
( Attacker_CVE_1999_0035_ftp_exploit ( S = portal T = nix_host ) ) $(2 . 2)
M-18 (($ 6 . 22))
( Attacker_CAN_2003_0715_dcom_exploit ( S = portal T = ms_host ) ) $(4 . 10)
M-45 (($ 10 . 32))
( Attacker_CAN_2002_0649_sql_exploit ( S = portal T = db_server ) ) $(4 . 45)
M-85 (($ 14 . 77))
( Attacker_rsh_login ( S = portal T = nix_host ) ) $(1 . 6)
M-122 (($ 15 . 83))
( Attacker_CAN_2003_0620_man_db_exploit ( T = nix_host ) ) $(3 . 10)
M-140 (($ 18 . 93) DEAD)

Figure 10. Path to root with cost benefit annotations

### Cost benefit evaluation

For cost benefit evaluations an adequate measure has to be defined. In the example scenario it is assumed, that costs reflect the effort an attacker uses in each step of the attack. Costs are directly assigned to the atomic exploits in this example, whereas the benefit for a transition is computed as the worth of the target host multiplied by the rank of the access right gained. The benefit for the attacker reflects the negative impact for the enterprise. Of course other kinds of measures for cost and benefit or other appropriate measures could be implemented following the proposed scheme. Assumed costs and benefits per exploit for specification part $\boxed{\text{S 5}}$ of the example scenario are assigned as shown in the tables in figure 11.

| Exploit | Cost |
|---|---|
| CAN_2003_0693_ssh_exploit | 3 |
| CAN_2003_0693_ssh_exploit_stealth | 4 |
| CVE_1999_0035_ftp_exploit | 2 |
| CAN_2003_0620_man_db_exploit | 3 |
| CAN_2003_0715_dcom_exploit | 4 |
| CAN_2003_0694_sendmail_exploit | 4 |
| CAN_2002_0649_sql_exploit | 4 |
| rsh_login | 1 |

| Host | Worth |
|---|---|
| telework | 1 |
| attacker | 0 |
| nix_host | 2 |
| ms_host | 2 |
| db_server | 9 |
| portal | 4 |

| Access | Rank |
|---|---|
| none | 1 |
| restricted_user | 2 |
| user | 3 |
| db_user | 4 |
| root | 5 |

Figure 11. Cost benefit values

Now by shortest path computation in a post-processing step on the attack graph, the values for cost and benefit can be summed up along the paths with the least cost to any node. The cost and benefit of a transition is depicted by the numbers after the $-sign at the edges in figure 10 that shows an example path in the attack graph with annotated cost benefit annotations. The sums along the path are depicted inside the nodes in the same figure.

A search for the node with the highest benefit score for the attacker (where most negative impact is achieved) returns the node $M135$ which has the same benefit rating namely 93 as the node $M140$ in figure 10.

### Cut down the graph

Defining a condition that stops further computation after an attack has been detected by an intrusion detection component generates only a subgraph with 57 nodes (33 dead) and 95 edges. The graph reduced to only undetected attacks generates a subgraph with only 24 nodes (4 dead) and 60 edges.

Cost benefit analysis for the graph with undetected attacks shows that the maximum benefit an attacker can obtain undetected in this scenario is 48. Inspecting the respective node in the attack graph shows that the attacker has gained root access on hosts attacker, ms_host, nix_host and portal but no access on db_server and telework.

### Abstraction

In some applications the SH verification tool already computed graphs of about 1 million edges in acceptable time and space. But it is impossible to visualise a graph of that size. So abstraction focussing on some interesting aspect is definitely a comfortable way to go in this case. An example for the usage of behaviour abstraction is shown in figure 12. The abstract view in this case shows that only one type of exploit can be used to attack the db_server and the graph is reduced from 544 edges to only one edge in the abstracted behaviour. The predicate used to define the corresponding mapping hides (maps to epsilon) all transitions that don't have the target host db_server.

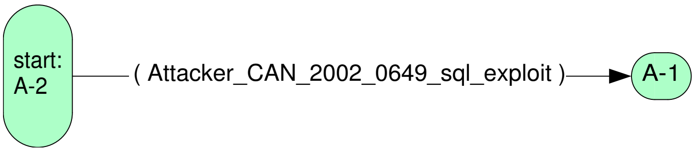


Figure 12. Attack graph abstraction showing transitions with target $db\_server$

Figure 13 shows an abstraction focussing on transitions with benefit $> 10$ and the resulting graph is also very concise.

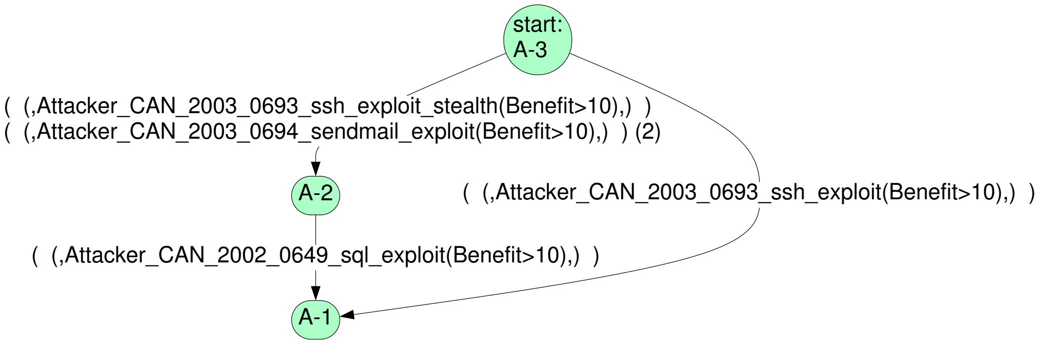

start:
A-3
( (,Attacker_CAN_2003_0693_ssh_exploit_stealth(Benefit>10),) )
( (,Attacker_CAN_2003_0694_sendmail_exploit(Benefit>10),) ) (2)
A-2
( (,Attacker_CAN_2003_0693_ssh_exploit(Benefit>10),) )
( (,Attacker_CAN_2002_0649_sql_exploit(Benefit>10),) )
A-1

Figure 13. Attack graph abstraction showing transitions with benefit $> 10$

**System countermeasures and critical services**

As an example for a check for critical services availability and to demonstrate how system countermeasures can be added to the framework defined so far, it is assumed that the host $db\_server$ always tries to answer queries from host $teleworker$. As a precondition the server checks if $sshd$ is running on the $portal$ because a “ssh-tunnel” on that host is used to reach $teleworker$. Now as shown in figure 8 (in condition V 5) the attacker kills the $sshd$ when executing the $CAN\_2003\_0693\_ssh\_exploit$. So if the attacker applies this exploit to attack the host $portal$, then afterwards the $sshd$ is not active on that host and so $db\_server$ cannot send an answer to $telework$ anymore. Now additionally a system countermeasure is considered that restarts the $sshd$ on the $portal$ from time to time. Two transitions patterns, namely T Defence *Restart_sshd* and T Service *Answer* add these actions to the model. The defence operation restarts $sshd$ when it is down and the service action checks for an active $sshd$ on $portal$. No other details are added to keep the model small.

Now a new computation of the attack graph results in a graph with 234 nodes (0 dead !) and 1136 edges. A section of this graph is shown in figure 14.

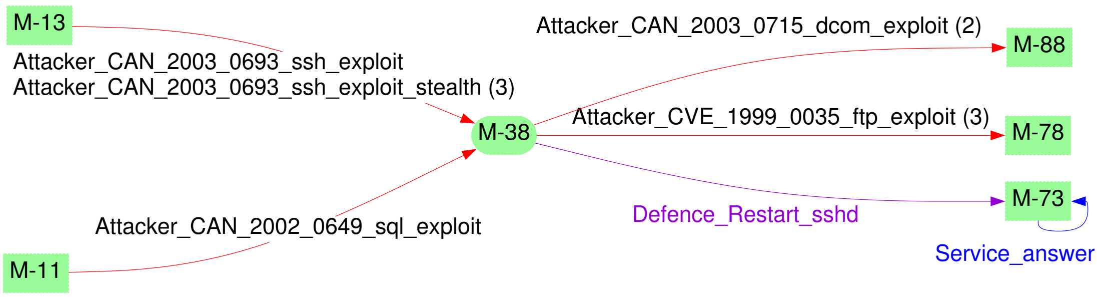

M-13
Attacker_CAN_2003_0693_ssh_exploit
Attacker_CAN_2003_0693_ssh_exploit_stealth (3)
M-11
Attacker_CAN_2002_0649_sql_exploit
M-38
Attacker_CAN_2003_0715_dcom_exploit (2)
M-88
Attacker_CVE_1999_0035_ftp_exploit (3)
M-78
Defence_Restart_sshd
M-73
Service_answer

Figure 14. Section of attack graph with service and countermeasure

A typical liveness question for the sketched situation is $\boxed{\text{Q 11}}$ from the introduction (Is a *client* still able to get answers from a DB-server when the enterprise network is under attack ?). Using an appropriate type of model checking, *approximate satisfaction* of temporal logic formulae can be checked by the SH verification tool (Ochsenschläger et al., 1999, 2000a). In terms of temporal logic the property above can be written as $\mathrm{G}\ \mathrm{F}\ \mathit{Service\_Answer}$ (always eventually $\mathit{Service\_Answer}$) which is found to be $\mathtt{true}$ by the tool.

Lifting the assertion that the attacker only attacks a host if he gains some credentials for the $\mathit{CAN\_2003\_0693\_ssh\_exploit}$ (see figure 8 the check for "no root access" on target host in $\boxed{\text{E 2}}$) leads to an attack graph with 3062 nodes (0 dead) and 22228 edges. This illustrates the dramatic influence of monotonicity assumptions on attack graph growth.

## Related work

The approach that Phillips and Swiler first presented in (Phillips and Swiler, 1998) is closest to the approach proposed in this paper. They described a prototype tool implementing their method in (Swiler et al., 2001). Similar to the computation method based on the SH verification tool outlined here, their method computes an attack graph starting from an initial node, but they don't describe abstraction methods to visualise compact presentations of the graph and they don't address liveness analysis that is used here to assure system response to critical services under attack.

Jha, Sheyner, Wing et al. use scenario graphs in (Jha and Wing, 2001) and attack graphs (Jha et al., 2002; Sheyner et al., 2002) that are computed and analysed based on model checking.

Ammann et al. presented an approach in (Ammann et al., 2002) that is focussed on reductions of complexity of the analysis problem from exponential to polynomial by explicit assumptions of monotonicity.

## Conclusions

Within the critical infrastructure protection context this paper aims at the protection of the core information infrastructure although the methods presented here could be extended and applied to other types of infrastructure and threats. The presented methodology for computation and analysis of attack graphs outlined in figure 5 is based on a formal specification of an organisation's critical network infrastructure, supplemented by a generic vulnerability and exploit specification and an attacker specification to model the threats against that infrastructure. The tool supported analysis of the attack graph assists in revealing vulnerabilities of the threatened infrastructure including complex attack combinations and supports the systematic evaluation of possible solutions to minimise risk with given resources. Contributions of this work are:

**Specification framework for critical network infrastructures and threats**
It is worked out in detail, how to formally specify topology and components of the information infrastructure and represent it by state components in asynchronous product automata (APA) notation. The operational formal system specification is completed by specifications of vulnerabilities, exploits and attacker capabilities represented by APA state transitions (see figures 1, 3 and 4). Specific templates to support and simplify formal modelling of enterprise networks under attack have been developed. Moreover, extensions to the model to add system defence operations and critical services actions are proposed, supplemented by some abstraction concepts, to prevent state space explosion problems in such models.

**Methodology and tool to analyse vulnerabilities and countermeasures**
From the APA specification an attack graph representing the behaviour of the model is automatically computed. Based on this graph, tool supported analysis methods are presented that can be used to answer the various questions posed in the introduction. Specific features of this approach comprise:

- an integrated interactive visualisation support to browse or debug the behaviour of the model and explore selected parts of the graph
- the usage of a well-elaborated and formally proven abstraction concept combined with an appropriate model checking component for analysis of security and liveness properties
- an integrated cost-benefit analysis method
- a seamless transition between verification and simulation on the same model when a complete computation of the attack graph is not possible
- a flexible configuration management simplifies evaluation and comparison of different solutions

**Further research objectives**
To seamlessly integrate the methods and tool presented here into a network vulnerability analysis framework, a tool-assisted transformation of a system configuration as provided by administration databases or gathered by network scanners into formal specifications is required. Likewise, some improvement towards generic formal vulnerability and exploit specifications is needed.

An in-depth research objective is, to develop methods and tool support to reduce state space explosion by further elaborating the ideas on abstraction of the system specification as sketched in the paragraphs about "representative hosts". For such a *tool assisted specification abstraction*, it has to be (automatically) proven, that the system specification is *appropriately transformed* into the abstracted specification, to assure that system properties are transported from a lower to a higher level of abstraction and no critical behaviour is hidden.

Another interesting perspective is, to extend the specification and analysis method described in this paper for application in other similar structured scenarios, as for example, to model a networked infrastructure system of a country including specifications of mutual dependencies as described in (Luiijf et al., 2003). Such a model could be used to analyse vulnerabilities and to raise risk awareness. It could help to reveal complex attack combinations and support systematic evaluation of possible solutions. This approach aims at optimising security and protection of networked systems with given resources.

## References

Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 217–224. ACM Press New York, NY, USA, 2002. ISBN 1-58113-612-9.

Urs E. Gattiker, Hervé Debar, Gasper Lvarencic, Giannis A Pikrammenos, Jerman Borka, Roland Rieke, Atta Badii, Yong Hua Song, Theis Søndergaard, Rainer Fahs, Helga Treiber, and Mario Wolframm. Cyber attack methods detection & information exploitation research project proposal, 2003. URL `http://www.eicar.org/camdier/index.html`.

S. Gürgens, P. Ochsenschläger, and C. Rudolph. Role based specification and security analysis of cryptographic protocols using asynchronous product automata. In *DEXA 2002 International Workshop on Trust and Privacy in Digital Business*. DEXA, 2002a. URL `http://www.sit.fhg.de/english/META/meta_publications/doc/Dexa2002-abstract.pdf`. Copyright: ©2002, IEEE. All rights reserved.

S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and Provability

- a Formal Framework. GMD Report 150, Fraunhofer-Institute for Secure Telecooperation, 2002b.

Somesh Jha and Jeannette M. Wing. Survivability analysis of networked systems. In *Proceedings of the 23rd international conference on Software engineering*, pages 307–317. IEEE Computer Society, 2001.

Somesh Jha, Oleg Sheyner, and Jeannette M. Wing. Two formal analyses of attack graphs. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*, pages 49–63. IEEE Computer Society, 2002.

Christopher Krügel and Thomas Toth. Distributed pattern detection for intrusion detection. In *Network and Distributed System Security Symposium Conference Proceedings: 2002*, 1775 Wiehle Ave., Suite 102, Reston, Virginia 20190, U.S.A., 2002. Internet Society. URL `citeseer.nj.nec.com/501183.html`.

E. Luiijf, H. Burger, and M. Klaver. Critical infrastructure protection in the netherlands: A quick-scan. In *EICAR Conference Best Paper Proceedings*, May 2003.

Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. M2d2: A formal data model for ids alert correlation. In *Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002, Zurich, Switzerland, October 16-18, 2002, Proceedings*, volume 2516 of *Lecture Notes in Computer Science*, pages 115–137. Springer, 2002.

P. Ochsenschläger, J. Repp, and R. Rieke. The SH-Verification Tool. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, pages 18–22, Orlando, FL, USA, May 2000a. AAAI Press. ISBN 0-1-57735-113-4.

Peter Ochsenschläger, Jürgen Repp, Roland Rieke, and Ulrich Nitsche. The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing, The International Journal of Formal Method*, 11:1–24, 1999.

Peter Ochsenschläger, Jürgen Repp, and Roland Rieke. Abstraction and composition – a verification method for co-operating systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 12:447–459, June 2000b.

Cynthia A. Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98, Proceedings of the 1998 Workshop on New Security Paradigms, September 22-25, 1998, Charlottsville, VA, USA*, pages 71–79. ACM Press, 1998.

Roland Rieke. Development of formal models for secure e-services. In *Eicar Conference 2003*, May 2003. URL http://www.sit.fhg.de/english/META/meta_publications/doc/Eicar-2003.pdf.

Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *2002 IEEE Symposium on Security and Privacy, May 12-15, 2002, Berkeley, California, USA*, pages 273–284. IEEE Comp. Soc. Press, 2002.

Laura P. Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. In *DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume 2,June 12 - 14, 2001, Anaheim, California*, pages 1307–1321. IEEE Computer Society, 2001.

# P10

# MODELLING AND ANALYSING NETWORK SECURITY POLICIES IN A GIVEN VULNERABILITY SETTING

| | |
|---|---|
| **Title** | Modelling and Analysing Network Security Policies in a Given Vulnerability Setting |
| **Authors** | Roland Rieke |
| **Publication** | In Javier Lopez, editor, *Critical Information Infrastructures Security, First International Workshop, CRITIS 2006, Samos Island, Greece. Revised Papers*, volume 4347 of *Lecture Notes in Computer Science*, pages 67–78, 2006. |
| **ISBN/ISSN** | ISBN 978-3-540-69083-2 |
| **DOI** | `http://dx.doi.org/10.1007/11962977_6` |
| **Status** | Published |
| **Publisher** | Springer Berlin Heidelberg |
| **Publication Type** | Conference Proceedings (LNCS, Vol. 4347) |
| **Copyright** | 2006, Springer |
| **Contribution of Roland Rieke** | Author and presenter at the CRITIS workshop 2006. |

Table 15: Fact Sheet Publication *P10*

Publication P10 [Rieke, 2006] addresses the following research questions:

RQ6a *What are the effects of changes to the network configuration on overall vulnerability?*

RQ6b *What is the most likely attacker behaviour and most effective countermeasure?*

RQ6c *Will countermeasures of the system under attack succeed?*

A typical means by which an attacker or his malware try to break into a network is, to use combinations of basic exploits to get more information or more credentials and to capture more hosts step by step. To find out if there is a combination that enables an attacker to reach critical network resources or block essential services, it is required to analyse all possible sequences of basic exploits, so called *attack paths*. Based on such an analysis, it is now possible to find out whether a given security policy successfully blocks attack paths and is robust against changes in the given vulnerability setting.

For this type of security policy analysis, a formal modelling framework is presented that, on the one hand, represents the information system and the security policy, and, on the other hand, a model of attacker capabilities and profile. It is extensible to comprise intrusion detection components and optionally a model of the system's countermeasures. Based on such an operational model, a graph representing all possible attack paths can be automatically computed. Now security properties can be specified and verified on this *attack graph*. If the model is too complex to compute the behaviour, then simulation can be used to validate the effectiveness of a security policy. The impact of changes to security policies can be computed and visualised by finding differences in the attack graphs. A unique feature of the presented approach is, that abstract representations of these graphs can be computed that allow comparison of focussed views on the behaviour of the system. This guides optimal adaptation of the security policy to the given vulnerability setting.

# Modelling and Analysing Network Security Policies in a Given Vulnerability Setting

Roland Rieke*

Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany
rieke@sit.fraunhofer.de

**Abstract.** The systematic protection of critical information infrastructures requires an analytical process to identify the critical components and their interplay, to determine the threats and vulnerabilities, to assess the risks and to prioritise countermeasures where risk is unacceptable. This paper presents an integrated framework for model-based symbolic interpretation, simulation and analysis with a comprehensive approach focussing on the validation of network security policies. A graph of all possible attack paths is automatically computed from the model of an ICT network, of vulnerabilities, exploits and an attacker strategy. Constraints on this graph are given by a model of the network security policy. The impact of changes to security policies can be computed and visualised by finding differences in the attack graphs. A unique feature of the presented approach is, that abstract representations of these graphs can be computed that allow comparison of focussed views on the behaviour of the system. This guides optimal adaptation of the security policy to the given vulnerability setting.

**Keywords:** threats analysis, attack simulation, critical infrastructure protection, network security policies, risk assessment, security modelling and simulation.

## 1 Introduction

Information and communication technology (ICT) is creating innovative systems and extending existing infrastructure to such an interconnected complexity that predicting the effects of small internal changes (e.g. firewall policies) and external changes (e.g. the discovery of new vulnerabilities and exploit mechanisms) becomes a major problem. The security of such a complex networked system essentially depends on a concise specification of security goals, their correct and consistent transformation into security policies and an appropriate deployment and enforcement of these policies. This has to be accompanied by a concept to adapt the security policies to changing context and environment, usage patterns and attack situations. To help to understand the complex interrelations of security policies, ICT infrastructure and vulnerabilities and to validate security

* Part of the work presented in this paper was developed within the project SicAri being funded by the German Ministry of Education and Research.

goals in such a setting, tool based modelling techniques are required that can efficiently and precisely predict and analyse the behaviour of such complex interrelated systems. These methods should guide a systematic evaluation of a given network security policy and assist the persons in charge with finally determining exactly what really needs protection and which security policy to apply.

A typical means by which an attacker or his malware try to break into a network is, to use combinations of basic exploits to get more information or more credentials and to capture more hosts step by step. To find out if there is a combination that enables an attacker to reach critical network resources or block essential services, it is required to analyse all possible sequences of basic exploits, so called *attack paths*. Based on such an analysis, it is now possible to find out whether a given security policy successfully blocks attack paths and is robust against changes in the given vulnerability setting.

For this type of security policy analysis, a formal modelling framework is presented that, on the one hand, represents the information system and the security policy, and, on the other hand, a model of attacker capabilities and profile. It is extensible to comprise intrusion detection components and optionally a model of the system's countermeasures. Based on such an operational model, a graph representing all possible attack paths can be automatically computed. It is called *attack graph* in the following text. Now security properties can be specified and verified on this attack graph. If the model is too complex to compute the behaviour, then simulation can be used to validate the effectiveness of a security policy. The impact of changes to security policies can be computed and visualised by finding differences in the attack graphs. Furthermore, abstract representations of these graphs can be computed that allow comparison of focussed views on the behaviour of the system. If there are differences in the detailed attack graphs but no differences in the abstract representations thereof, this proves that the different policies are equally effective on the enforcement of security goals on the abstract level, even if variations in the attack paths are covered by different policy rules. The subsequent paper is structured as follows. Section 2 gives an overview of related work. The modelling approach is described in Sect. 3, while Sect. 4 presents an exemplary analysis of network security policy adaptation aspects in a given scenario. Finally, the paper ends with an outlook in Sect. 5.

## 2 Related Work

The network vulnerability modelling part of the framework presented in this paper is adopted from the approach introduced in [1] and is similar in design to an approach by Phillips and Swiler in [2] and [3]. A major contribution of [1] was the use of abstraction methods to visualise compact presentations of the graph and the inclusion of liveness analysis. Related work of Jha, Sheyner, Wing et al. used attack graphs that are computed and analysed based on model checking in [4] and [5]. Ammann et al. presented an approach in [6] that is focussed on reduction of complexity of the analysis problem by explicit assumptions of monotonicity. Recent work in this area by Noel, Jajodia et al. in [7] and [8] describes attack graph visualisation techniques while the work of Kotenko and Stepashkin in [9] is focussed on security metrics computations.

To model the ICT network, the vulnerabilities and the intrusion detection systems, a data model loosely resembling the formally defined M2D2 information model [10] is used. Appropriate parts of this model are adopted and supplemented by concepts needed for description of exploits, attacker knowledge and strategy and information for cost benefit analysis.

The model of the network security policies used in this paper is based on the Organisation Based Access Control (Or-BAC) model. A formal approach to use Or-BAC to specify network security policies was presented in [11]. This approach is used here to model the network security policies in the attack graph analysis framework.

The modelling framework is based on Asynchronous Product Automata (APA), a flexible operational specification concept for cooperating systems [12]. An APA consists of a family of so called elementary automata communicating by common components of their state (shared memory). The applied verification method is implemented in the SH verification tool [13] that has been adapted and extended to support the presented attack graph analysis methods.

Major focus of the combined modelling framework presented in this paper, is the integration of formal network vulnerability modelling on the one hand and network security policy modelling on the other hand. This aims to help adaptation of a network security policy to a given and possibly changing vulnerability setting. Recent methods for analysis of attack graphs are extended to support analysis of abstract representations of these graphs.

## 3 Modelling Critical ICT Infrastructures and Threats

The proposed operational model comprises, (1) an asset inventory including critical network components, topology and vulnerability attributions, (2) a network security policy, (3) vulnerability specifications and exploit descriptions, and (4) an attacker model taking into account the attackers knowledge and behaviour.

### 3.1 ICT Network Components

The set of all hosts of the information system consists of the union of the hosts of the ICT network and the hosts of the attacker(s). Following the M2D2 model, *products* are the primary entities that are vulnerable. A *host configuration* is a subset of products that is installed on that host and *affects* is a relation between vulnerabilities and sets of products that are affected by a vulnerability. A host is *vulnerable* if its configuration is a superset of a vulnerable set of products and the affected services are currently running.

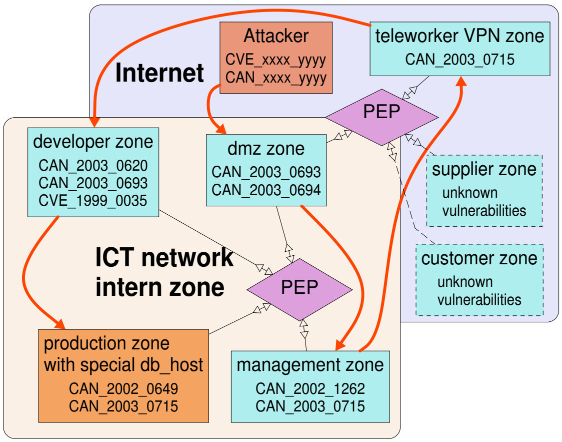
In order to conduct a subsequent comparative analysis of attack paths, an asset prioritisation as to criticality or worth regarding relative importance of a host is required.

### 3.2 Network Security Policies

The model of the network security policies is based on the Organisation Based Access Control (Or-BAC) model. The approach to use Or-BAC to specify network security policies as presented in [11] is adopted here to model the network security policies in the attack graph analysis framework. The advantage of this choice is, that it is possible to link the policies in the formal model at an abstract level to the low level vendor specific policy rules for the policy enforcement points (PEPs) such as firewalls in the concrete ICT network. Please refer to [11] for such a transformation concept exemplified on the iptables packet filtering mechanism used in Linux.

Following the Or-BAC based concept, the network vulnerability policy is given at an abstract level in terms of *roles* (an abstraction of subjects), *activities* (an abstraction of actions) and *views* (an abstraction of objects). A *subject* in this model is any host. An *action* is a network service such as snmp, ssh or ftp. Actions are represented by a triple of protocol, source port and target port. An *object* is a message sent to a target host. Currently only the target host or rather the role of the target host is used for the view definition here. To specify the access control policy using this approach, *permissions* are given between role, activity and view.

To illustrate the concept described here, a small example scenario is given in Fig. 1(a). Modelling concepts and typical analysis outcome will be illustrated using this example scenario throughout the paper. One possible attack path is sketched in the scenario. The policy rules for the example scenario are defined by the table in Fig. 1(b).



(a) ICT network and vulnerabilities

| Role (source) | View (target) | Activity (service) |
|---|---|---|
| *internet* | *internet* | *any* |
| *any* | *dmz* | *ssh* |
| *any* | *dmz* | *smtp* |
| *dmz* | *intern* | *ssh* |
| *intern* | *any* | *net* |
| *intern* | *internet* | *ftp* |
| *intern* | *internet* | *rsh* |
| *intern* | *dmz* | *ssh* |
| *db_host* | *production* | *rpc* |
| *teleworker* | *dmz* | *any* |

(b) Network security policy

**Fig. 1.** Scenario and network security policy

### 3.3 Vulnerabilities

Vulnerability specifications for the formal model are derived from the Common Vulnerabilities and Exposures (CVE/CAN) descriptions. The MITRE Corporation provides a CVE web site (`http://www.cve.mitre.org/`) with a list of virtually all known vulnerabilities. The CVE name is the 13 character ID used by the CVE standards group to uniquely identify a vulnerability. Additional information about the vulnerabilities also covers preconditions about the target host as well as network preconditions. Furthermore, the impact of an exploitation of a vulnerability is described. The specifications for the formal model of the vulnerabilities additionally comprise the vulnerability *range* and *impact type* assessments provided by the National Institute of Standards and Technology (NIST) (`http://nvd.nist.gov/`).

**Vulnerability Severity.** The Common Vulnerability Scoring System (CVSS) [14] provides universal severity ratings for security vulnerabilities. These ratings are used in the model as an example for a measure of the threat level. Another example for such a measure is the metric used by the US-CERT (cf. `http://www.kb.cert.org/vuls/html/fieldhelp#metric`). These measures are based on information about the vulnerability being widely known, reported exploitation incidents, number of infected systems, the impact of exploiting the vulnerability and the knowledge and the preconditions required to exploit the vulnerability. Because the approximate values included in those measures may differ significantly from one site to another, prioritising of vulnerabilities based on such measures should be used with caution.

To have a vulnerable product installed on some host, does not necessarily imply, that someone can exploit that vulnerability. A target host *is configured vulnerable*, if (1) the target host has installed a product or products that are vulnerable with respect to the given vulnerability, and (2) necessary other preconditions are fulfilled (e.g. some vulnerabilities require that a trust relation is established as for example used in remote shell hosts allow/deny concepts).

A second precondition to exploit a vulnerability is, that the target host *is currently running the respective products* such as a vulnerable operating system or server version. If a user interaction is required this also requires that the vulnerable product is currently used (e.g. a vulnerable Internet explorer).

The third necessary preconditions is, that the *network security policy permits* that the target host is reachable on the port the vulnerable product is using from the host the attacker selected as source.

### 3.4 Attacker and System Behaviour

**Attacker Knowledge.** The knowledge of exploits and hosts and the credentials on the known hosts constitute an attackers profile. Knowledge about hosts changes during the computation of the attack graph because the attacker might gain new knowledge when capturing hosts. On the other hand, some knowledge may become outdated because the enterprise system changes ip-numbers or other configuration of hosts and reachability. In case a vulnerability is exploited, the model has to cover the *effects for the attacker* (for example, to obtain additional user or root credentials on the target host) and also the *direct impact on the network and host* such as, to shut down a service caused by buffer overflow.

**Dynamic System Behaviour.** The information model presented so far covers the description of a (static) configuration of an ICT network and its vulnerabilities. In the formal model such a configuration describing the *state* of the ICT network is represented by *APA state components* (APA representation of an ICT network is covered in more detail in [1]).

To describe how actions of attacker(s) and actions of the system can change the state of the ICT network model, specifications of *APA state transitions* are used. These state transitions represent atomic exploits and optionally the actions that the ICT network system can take to defend itself or to implement vital services. Formally, a state transition can occur, when all expressions are evaluable and all conditions are satisfied. So called *interpretation variables* are used to differentiate the variants of execution of the same transition. All possible variants of bindings of interpretation variables from the state components are generated automatically. So for example for a transition modelling an exploit, all possible combinations of bindings of source and target host are computed and further evaluated.

**Attacker Behaviour.** Attacker capabilities are modelled by the atomic exploits and by the strategy to select and apply them.

A state transition modelling an exploit is constructed from, (1) a predicate that states that the attacker *knows* this exploit, (2) an expression to select source and target hosts for the exploit, (3) a predicate that states that the target *host is vulnerable* by this exploit, (4) an expression for the impact of the execution of this exploit on the attacker and on the target host as for example the shut down of services. Optional add-ons are, an assignment of cost benefit ratings to this exploit and intrusion detection checks.

Several different attackers can easily be included because an attacker is modelled as a role not a single instance and the tool can automatically generate multiple instances from one role definition.

Modelling of Denial of Service (DoS) attacks aiming to block resources or communication channels either directly or by side effects require a much more detailed model of the resources involved. This could be accomplished using the presented framework but is out of scope of this paper.

Some experiments have been made to generate a set of known exploits for the attacker(s) from a given algorithm. If for example it is assumed that the attacker knows 3 different exploits, then all combinations of 3 exploits from the set of all specified exploits have to be computed and further analysed. Another example for an attacker strategy is, that the attacker uses only exploits for vulnerabilities with a severity above a given threshold. This is based on the assumption, that the vulnerability severity reflects the probability of exploitation of a vulnerability.

**Composition of a Model and Computation of an Attack Graph.** The SH verification tool [13] is used to analyse this model. It manages the components of the model, allows to select alternative parts of the specification and automatically "glues" together the selected components to generate a combined model of ICT network specification, vulnerability and exploit specification, network security policy and attacker specification.

After an initial configuration is selected, the attack graph (reachability graph) is automatically computed by the SH verification tool. Also, on the fly analysis allows, to stop computation automatically when specified conditions are reached (or invariants are broken), so called break conditions can be specified using regular expressions. A violation of a security property for example, can in many cases be specified as a break condition.

**Attack Graph of the Example Scenario.** The computed attack graph for the simple example scenario (assuming the attacker knows all exploits) has 500 nodes and 4136 edges. Now we assume as a more realistic attacker behaviour, that the attacker will only exploit vulnerabilities with a severity level above a given minimum. In the example scenario, a severity level of 4 results in an attack graph with 178 nodes and 1309 edges. This graph is still far too big to inspect it manually. Figure 2 shows a small section of it. Nodes with circle shape depict states where the successors are completely shown, nodes with rectangular shape depict nodes where the successors are cropped. For example the edge $M4 \longrightarrow M5$ depicts the application of an exploit where the ssh-vulnerability $CAN\_2003\_0693$ was used and the edge $M4 \longrightarrow M6$ depicts an exploit based on the same vulnerability but in this case operating stealth (not detected).

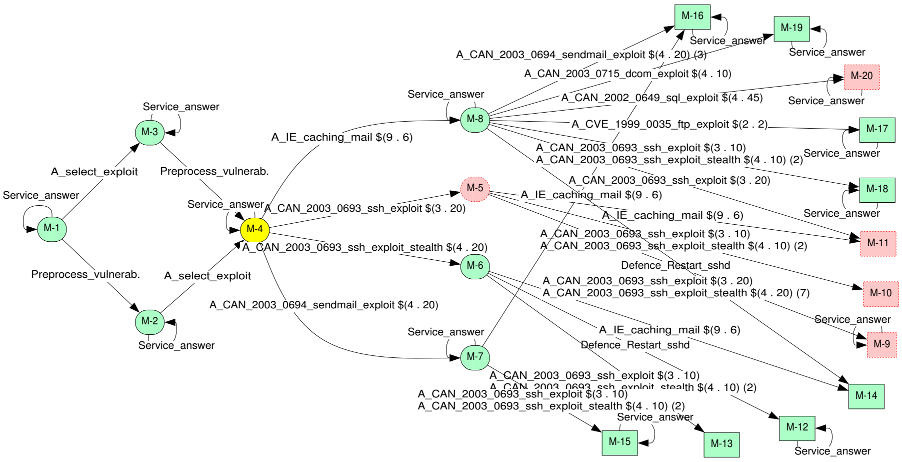

M-16
M-19
Service_answer
Service_answer
A_CAN_2003_0694_sendmail_exploit $(4 . 20) (3)
A_CAN_2003_0715_dcom_exploit $(4 . 10)
M-20
Service_answer
A_CAN_2002_0649_sql_exploit $(4 . 45)
Service_answer
Service_answer
M-3
M-8
A_IE_caching_mail $(9 . 6)
A_CVE_1999_0035_ftp_exploit $(2 . 2)
M-17
A_CAN_2003_0693_ssh_exploit $(3 . 10)
A_CAN_2003_0693_ssh_exploit_stealth $(4 . 10) (2)
Service_answer
A_select_exploit
Preprocess_vulnerab.
A_CAN_2003_0693_ssh_exploit $(3 . 20)
M-18
Service_answer
M-5
A_IE_caching_mail $(9 . 6)
Service_answer
A_CAN_2003_0693_ssh_exploit $(3 . 20)
A_IE_caching_mail $(9 . 6)
Service_answer
M-1
M-4
A_CAN_2003_0693_ssh_exploit $(3 . 10)
A_CAN_2003_0693_ssh_exploit_stealth $(4 . 10) (2)
M-11
A_CAN_2003_0693_ssh_exploit_stealth $(4 . 20)
Defence_Restart_sshd
M-6
Preprocess_vulnerab.
A_select_exploit
A_CAN_2003_0693_ssh_exploit $(3 . 20)
A_CAN_2003_0693_ssh_exploit_stealth $(4 . 20) (7)
M-10
A_CAN_2003_0694_sendmail_exploit $(4 . 20)
Service_answer
M-2
Service_answer
A_IE_caching_mail $(9 . 6)
M-9
Service_answer
Defence_Restart_sshd
M-7
Service_answer
A_CAN_2003_0693_ssh_exploit $(3 . 10)
A_CAN_2003_0693_ssh_exploit_stealth $(4 . 10) (2)
A_CAN_2003_0693_ssh_exploit $(3 . 10)
A_CAN_2003_0693_ssh_exploit_stealth $(4 . 10) (2)
M-14
Service_answer
M-12
M-15
M-13
Service_answer

**Fig. 2.** Attack graph of example scenario (small section)

## 4 Evaluation of the Model

**Abstractions.** Abstract representations of the attack graph can be computed to visualise and analyse compacted information focussed on interesting aspects of the behaviour. The mappings used to compute the abstract representations of the behaviour have to be *property preserving*, to assure that properties are *transported* as desired from a lower to a higher level of abstraction and no critical behaviour is hidden by the mapping. Such properties, namely *simplicity*, are given in [15] and and a check for simplicity is implemented in the SH verification tool [13]. In some applications the SH verification tool already computed graphs of about 1 million edges in acceptable time and space. But it is impossible to visualise a graph of that size. So abstraction focussing on some interesting aspect is definitely a comfortable way to go in this case.

**An Example for the Usage of Behaviour Abstraction.** For this experiment, the vulnerability *range* and *impact type* assessments provided by NIST (cf. Sect. 3.3) are utilised. Range types of the vulnerabilities in the example scenario are *remote* (remotely exploitable) and *local* (locally exploitable). Impact types used here are *unspecific* (provides unauthorised access), *user* (provides user account access) and *root* (provides administrator access).

**Step 1 - Define a Mapping.** Figure 3 defines a mapping of all transitions representing the exploit of a vulnerability to the respective range and impact types of the vulnerabilities.

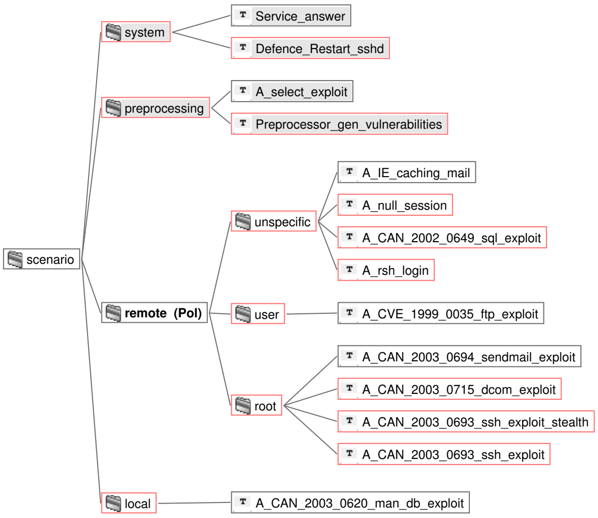

system
Service_answer
Defence_Restart_sshd
preprocessing
A_select_exploit
Preprocessor_gen_vulnerabilities
scenario
unspecific
A_IE_caching_mail
A_null_session
A_CAN_2002_0649_sql_exploit
A_rsh_login
remote (PoI)
user
A_CVE_1999_0035_ftp_exploit
root
A_CAN_2003_0694_sendmail_exploit
A_CAN_2003_0715_dcom_exploit
A_CAN_2003_0693_ssh_exploit_stealth
A_CAN_2003_0693_ssh_exploit
local
A_CAN_2003_0620_man_db_exploit

**Fig. 3.** Definition of an abstract representation of the attack graph

This mapping denotes, that all transitions (the leaves of the tree) are to be represented by their respective father nodes, namely *system*, *preprocessing*, *unspecific*, *user*, *root* and *local* in the abstract representation. The nodes *system* and *preprocessing* are coloured in grey, symbolising that they are mapped to $\epsilon$, that means the transitions represented by these nodes are invisible in the abstract representation. Please ignore the notation ($Pol$) at the node *remote* for the moment.

**Step 2 - Compute the Abstract Representation.** Figure 4 shows the computed abstract view focussing on the transition types *root*, *user*, *unspecific* and *local*. This graph with only 20 states and 37 edges was derived from the attack graph (cf. Fig. 2) with 178 states and 1309 edges. The simplicity of this mapping that guarantees that properties are preserved was automatically proven by the tool.

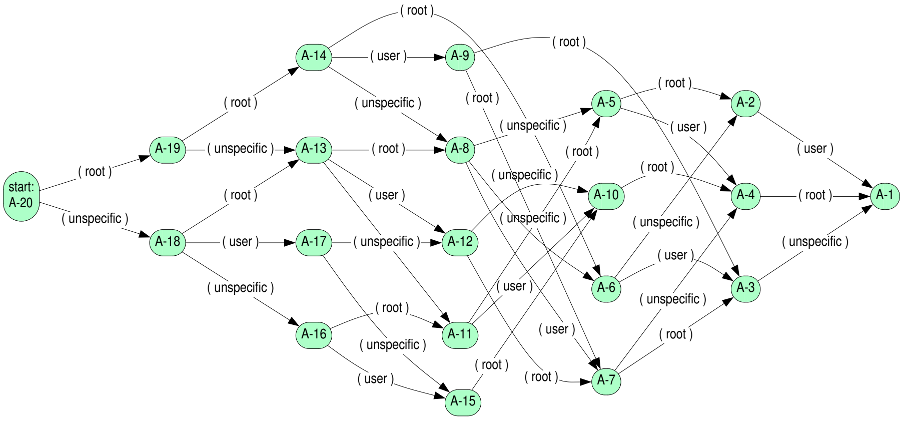
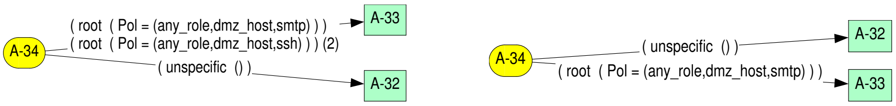


**Fig. 4.** Abstract view on an attack graph

**Step 3 - Optionally Refine the Mapping.** If you want to know for example, what policies are responsible to allow the attacks shown in Fig. 4 then a refinement of the abstraction defined in Fig. 3 is necessary. It is possible to "fine tune" the mapping so that the interpretation variables (cf. Sect. 3.4) stay visible in the abstract representation. In this case the binding of the interpretation variable $Pol$ that contains the respective policy can be visualised. This is denoted by ($Pol$) in the node *remote* in Fig. 3. The corresponding refined abstract representation is a graph with 34 states and 121 edges when computed on the attack graph in Fig. 2. The initial nodes and edges of this graph are shown in Fig. 5(a). In comparison to the initial edges of the graph in Fig. 4 now the details on the related policies are visible.


( root ( Pol = (any_role,dmz_host,smtp) ) )
( root ( Pol = (any_role,dmz_host,ssh) ) ) (2)
A-34
( unspecific () )
A-33
A-32
A-34
( unspecific () )
( root ( Pol = (any_role,dmz_host,smtp) ) )
A-32
A-33

(a) $(any\_role, dmz\_host, ssh/smtp)$ (b) $(any\_role, dmz\_host, smtp)$

**Fig. 5.** Details in the abstract view

**Step 4 - Adapt/Optimise the System Configuration.** Further analysis reveals, that, if the example policy given in Fig. 1(b) is changed to allow only *smtp* instead of *ssh* and *smtp* for *any_role* to *dmz_host* then the analysis yields a graph with only 94 states and 783 edges and performing the same steps as described above leads to the same graph (Fig. 4) in step 2 but a different one shown in Fig. 5(b) in the refinement step 3.

If alternatively the policy is restricted to allow only *ssh* instead of *ssh* and *smtp* in the above example, then again you get a different attack graph with 167 states and 1203 edges but the abstract view in step 2 is still the same.

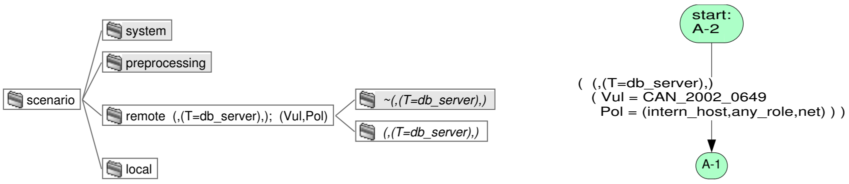
This stepwise analysis demonstrates that there may be differences in the detailed attack graphs but no differences in the abstract representations thereof. This indicates that the different policies are equally effective (or not) concerning the enforcement of security goals on the abstract level, even if variations in the attack paths are covered by different policy rules.

**Using Predicates to Define Abstractions.** Let us now assume that the host *db_server* in the scenario is the most valuable and mission critical host in the ICT network. So we want to know if in the given scenario, (1) attacks to the *db_server* are possible, (2) on which vulnerabilities they are based, and, (3) what policy rules are directly involved.

The abstraction in Fig. 6(a) exemplifies how predicates can be used to define such a mapping. In this mapping the predicate $(T = db\_server)$ matches only those transitions that model direct attacks to the target host *db_server*. The remote transitions that don't match that predicate are mapped to $\epsilon$ and so are invisible.

Evaluating this abstraction on the attack graph from Fig. 2 above results in the simple graph given in Fig. 6(b). This proves that, (1) in the current policy configuration attacks to the *db_server* are possible, (2) those attacks are based on exploits of the vulnerability $CAN\_2002\_0649$, and, (3) they are utilising the policy rule $(intern\_hosts, any\_role, net)$. So to prevent this attack, it has to be decided, if it is more appropriate to uninstall the product that is hurt by this vulnerability or to restrict the internal hosts in their possible actions by replacing the above policy with a more restrictive one.

Many further uses of these attack graphs are possible, such as cost benefit analysis or analysis of intrusion detection configurations.



(a) Abstraction used (b) Resulting graph

**Fig. 6.** Focus on attacks to the host *db_server*

Liveness properties in this context reflect survivability and business continuity aspects. When a system's countermeasures and the behaviour of vital services the system provides are included in the model, then these effects and the system's resilience can be analysed. Please refer to [1] for an example.

## 5 Further Research Objectives

The work presented in this paper brings together, (1) attack graph computation technology, (2) state-of-the-art policy modelling, and, (3) formal methods for analysis and computation of abstract representations of the system behaviour. The aim is, to guide a systematic evaluation and assist the persons in charge with optimising adaptation of the network security policy to an ever-changing vulnerability setting.

To seamlessly integrate the methods and tool presented here into a network vulnerability analysis framework, a tool-assisted transformation of up-to-date ICT system configuration and vulnerability databases into a formal specification of the model is required. This should preferably be based on automatically updated information of network scanners because administration databases are typically out-of-date. Recent work by Noel, Jajodia et al. in [7] and [8] already covers this aspect but more work is needed to facilitate the transformation of descriptions from vulnerability databases into formal vulnerability and exploit specifications.

A summarisation of severity ratings for single security vulnerabilities as provided by CVSS or US-CERT (cf. Sect. 3.3) based on attack graphs has been addressed in recent work of Kotenko and Stepashkin [9]. Interesting questions in such an approach are, which attacker strategy or bundle of strategies to apply and how to “condense” the information in the graph into a comprehensive measure of the security of an ICT network. Consideration of *resilience against unknown attacks* could also contribute to such a measure.

An even more advanced objective is, to extend this framework to support *policy-based, automated threat response* that makes use of alert information. Such a self-adaptive response mechanism could substantially improve the resilience of policy controlled ICT systems against network attacks.

## References

1. Rieke, R.: Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration. In: In U.E. Gattiker (Ed.), Eicar 2004 Conference CD-rom: Best Paper Proceedings, Copenhagen, EICAR e.V. (2004)
2. Phillips, C.A., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: NSPW '98, Proceedings of the 1998 Workshop on New Security Paradigms, September 22-25, 1998, Charlottsville, VA, USA, ACM Press (1998) 71–79
3. Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S.: Computer-attack graph generation tool. In: DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume 2,June 12 - 14, 2001, Anaheim, California, IEEE Computer Society (2001) 1307–1321
4. Jha, S., Sheyner, O., Wing, J.M.: Two formal analyses of attack graphs. In: 15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada, IEEE Computer Society (2002) 49–63
5. Sheyner, O., Haines, J.W., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: 2002 IEEE Symposium on Security and Privacy, May 12-15, 2002, Berkeley, California, USA, IEEE Comp. Soc. Press (2002) 273–284
6. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM conference on Computer and communications security, ACM Press New York, NY, USA (2002) 217–224
7. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, New York, NY, USA, ACM Press (2004) 109–118
8. Noel, S., Jacobs, M., Kalapa, P., Jajodia, S.: Multiple Coordinated Views for Network Attack Graphs. In: IEEE Workshop on Visualization for Computer Security (VizSec'05), Los Alamitos, CA, USA, IEEE Computer Society (2005)
9. Kotenko, I., Stepashkin, M.: Analyzing Network Security using Malefactor Action Graphs. International Journal of Computer Science and Network Security **6** (2006)
10. Morin, B., Mé, L., Debar, H., Ducassé, M.: M2d2: A formal data model for ids alert correlation. In: Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002, Zurich, Switzerland, October 16-18, 2002, Proceedings. Volume 2516 of Lecture Notes in Computer Science., Springer (2002) 115–137
11. Cuppens, F., Cuppens-Boulahia, N., Sans, T., Miège, A.: A formal approach to specify and deploy a network security policy. In: Second Workshop on Formal Aspects in Security and Trust (FAST). (2004)
12. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing, The International Journal of Formal Method **11** (1999) 1–24
13. Ochsenschläger, P., Repp, J., Rieke, R.: The SH-Verification Tool. In: Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000), Orlando, FL, USA, AAAI Press (2000) 18–22
14. Schiffmann, M.: A Complete Guide to the Common Vulnerability Scoring System (CVSS) (2005) http://www.first.org/cvss/cvss-guide.html.
15. Ochsenschläger, P., Repp, J., Rieke, R.: Verification of Cooperating Systems – An Approach Based on Formal Languages. In: Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000), Orlando, FL, USA, AAAI Press (2000) 346–350

# P11

# ABSTRACTION-BASED ANALYSIS OF KNOWN AND UNKNOWN VULNERABILITIES OF CRITICAL INFORMATION INFRASTRUCTURES

| | |
|---|---|
| **Title** | Abstraction-based analysis of known and unknown vulnerabilities of critical information infrastructures |
| **Authors** | Roland Rieke |
| **Publication** | *International Journal of System of Systems Engineering (IJSSE)*, 1:59–77, 2008. |
| **ISBN/ISSN** | ISSN 1748-0671 |
| **DOI** | `http://dx.doi.org/10.1504/IJSSE.2008.018131` |
| **Status** | Published |
| **Publisher** | InderScience |
| **Publication Type** | International Journal of System of Systems Engineering (IJSSE), Vol. 1 |
| **Copyright** | 2008, InderScience |
| **Contribution of Roland Rieke** | Author |

Table 16: Fact Sheet Publication *P11*

Publication P11 [Rieke, 2008a] addresses the following research question:

RQ7 *To which extent is a networked system resilient against exploits of unknown vulnerabilities?*

This journal paper is an extended version of P10. In addition to the results of P10 it provides an approach to analysis of unknown vulnerabilities. In order to analyse resilience of critical information infrastructures against exploits of unknown vulnerabilities, generic vulnerabilities for each installed product and affected service are added to the model. The reachability analysis now considers every possible choice of product, and so all alternatives are evaluated in the attack graph. The impact of changes to security policies or network structure can be visualised by differences in the attack graphs. Results of this analysis support the process of dependable configuration of critical information infrastructures.

# Abstraction-based analysis of known and unknown vulnerabilities of critical information infrastructures

Roland Rieke

Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany
rieke@sit.fraunhofer.de

**Abstract.** The systematic protection of critical information infrastructures requires an analytical process to identify the critical components and their interplay, to determine the threats and vulnerabilities, to assess the risks and to prioritise countermeasures where risk is unacceptable. The abstraction-based approach presented here builds on a model-based construction of an attack graph with constraints given by the network security policy. A unique feature of the presented approach is, that abstract representations of these graphs can be computed that allow comparison of focussed views on the behaviour of the system. In order to analyse resilience of critical information infrastructures against exploits of unknown vulnerabilities, generic vulnerabilities for each installed product and affected service are added to the model. The reachability analysis now considers every possible choice of product, and so all alternatives are evaluated in the attack graph. The impact of changes to security policies or network structure can be visualised by differences in the attack graphs. Results of this analysis support the process of dependable configuration of critical information infrastructures.

**Key words:** threats analysis, attack simulation, critical infrastructure protection, network security policies, risk assessment, security modelling and simulation, unknown vulnerabilities.

## 1 Introduction

Information and Communication Technology (ICT) is creating innovative systems and extending existing infrastructure to such an interconnected complexity that predicting the effects of small internal changes (e.g. firewall policies) and external changes (e.g. the discovery of new vulnerabilities and exploit mechanisms) becomes a major problem. The security of such a complex networked system essentially depends on a concise specification of security goals, their correct and consistent transformation into security policies and an appropriate deployment and enforcement of these policies. This has to be accompanied by a concept to adapt the security policies to changing context and environment, usage patterns and attack situations. To help to understand the complex interrelations of security policies, ICT infrastructure and vulnerabilities and to validate security goals in such a setting, tool-based modelling techniques are required that can efficiently and precisely predict and analyse the behaviour of such complex interrelated systems. Figure 1 shows an example of such an infrastructure. Known and unknown vulnerabilities may be part of each of the connected components and communication paths between them. Analysis methods should guide a sys-

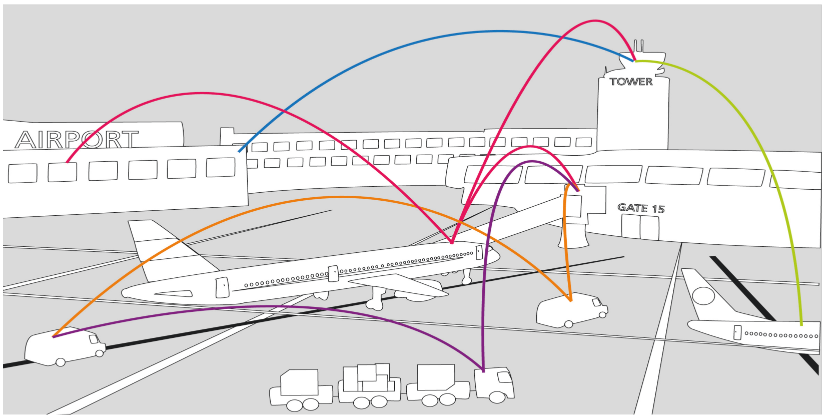


**Fig. 1.** Interplay of complex interrelated systems

tematic evaluation of such a critical information infrastructure assist the persons in charge with finally determining exactly how to configure protection measures and which security policy to apply.

A typical means by which an attacker (directly or using malware such as blended threats) tries to break into such a network is, to use combinations of basic exploits to get more information or more credentials and to capture more assets step by step. To find out if there is a combination that enables an attacker to reach critical network resources or block essential services, it is required to analyse all possible sequences of basic exploits, so called *attack paths*.

For this type of analytical analysis, a formal modelling framework is presented that, on the one hand, represents the information system and the security policy, and, on the other hand, a model of attacker capabilities and profile. It is extensible to comprise intrusion detection components and optionally a model of the system's countermeasures. Based on such an operational model, a graph representing all possible attack paths can be automatically computed. It is called an *attack graph* in the following text. Based on this attack graph, it is now possible to find out whether a given security policy successfully blocks attack paths and is robust against changes in the given vulnerability setting.

One problem now is, that it is usually impossible to visualise an attack graph of a realistic example directly because of the huge size. However, abstract representations of an attack graph can be computed and used to visualise and analyse compacted information focussed on interesting aspects of the behaviour. The impact of changes to security policies can be visualised by finding differences in the attack graphs or the abstract representations thereof.

This paper also shows, that abstract representations are very useful to analyse resilience of critical information infrastructure with respect to attacks based on unknown vulnerabilities because addition of unknown vulnerabilities results in very large attack graphs.

The subsequent paper is structured as follows. The modelling approach is described in Sect. 2, while Sect. 3 presents an exemplary analysis. Section 4 presents an approach to analyse resilience of critical information infrastructures against exploits of unknown vulnerabilities. Section 5 gives an overview of related work. Finally, this paper ends with an outlook in Sect. 6.

## 2 Modelling information infrastructures and threats

The proposed operational model comprises, (1) an asset inventory including critical network components, topology and vulnerability attributions, (2) a network security policy, (3) vulnerability specifications and exploit descriptions, and (4) an attacker model taking into account the attackers knowledge and behaviour.

### 2.1 ICT network components

The set of all hosts of the information system consists of the union of the hosts of the ICT network and the hosts of the attacker(s). Following the M2D2 model [1], *products* are the primary entities that are vulnerable. A *host configuration* is a subset of products that is installed on that host and *affects* is a relation between vulnerabilities and sets of products that are affected by a vulnerability. A host is *vulnerable* if its configuration is a superset of a vulnerable set of products and the affected services are currently running. In order to conduct a subsequent comparative analysis of attack paths, an asset prioritisation as to criticality or worth regarding relative importance of the assets is required.

### 2.2 Network security policies

The model of the network security policies is based on the Organisation-Based Access Control (Or-BAC) model [2]. The advantage of this choice is, that it is possible to link the policies in the formal model at an abstract level to the low level vendor specific policy rules for the Policy Enforcement Points (PEPs) such as firewalls in the concrete ICT network.

To illustrate the modelling concepts described here, a small example scenario is given in Fig. 2. Modelling concepts and typical analysis outcome will be illustrated using this example scenario throughout the paper. One possible attack path is sketched in the scenario.

Following the Or-BAC-based concept, the network vulnerability policy is given at an abstract level in terms of *roles* (an abstraction of subjects), *activities* (an abstraction of actions) and *views* (an abstraction of objects). A *subject* in this model is any host. An *action* is a network service such as snmp, ssh or ftp. Actions are represented by a triple of protocol, source port and target port. An *object* is a message sent to a target host. Currently only the target host or rather the role of the target host is used for the view definition here. To specify the access control policy using this approach, *permissions* are given between role, activity and view. For the example scenario the hostnames *telework*, *attacker*, *nix_host*, *ms_host*, *db_server* and *portal* are used. The roles of these hosts are given by the table in Fig. 3(a). The policy permissions are defined by the table in Fig. 3(b).

**Mobile components.** To model mobile components that can transport malware such as blended threats from one network zone to another, it is convenient to allow a host to play

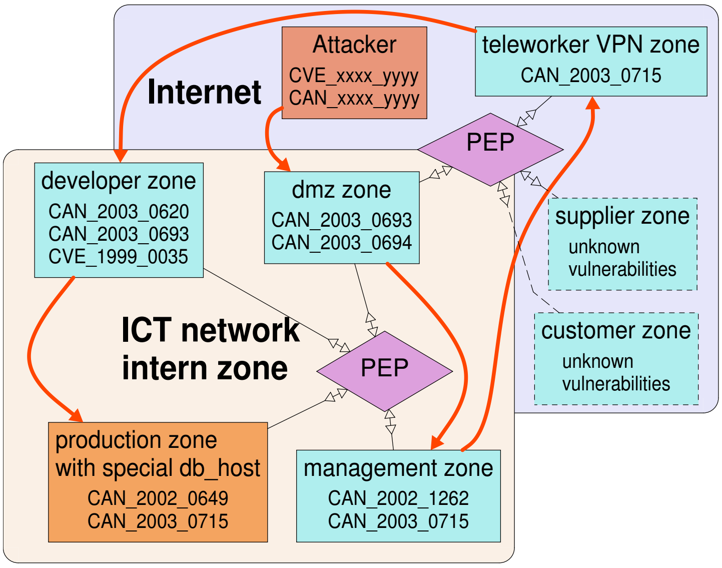


**Fig. 2.** ICT network and vulnerabilities

different roles. For example in Fig. 3(a) the host *telework* that plays the role *telework_host* can additionally be permitted to play the role *intern_host*. In this case an attack path could cross the zones from *telework_host* to *intern_host* without any restrictions by the network security policy. Similar problems exist in infrastructures with mobile components such as the example scenario shown in Fig.1.

### 2.3 Vulnerabilities

Vulnerability specifications for the formal model of the example scenario are derived from the Common Vulnerabilities and Exposures (CVE/CAN) descriptions. The MITRE Corporation provides a list of virtually all known vulnerabilities (`http://www.cve.mitre.org/`). The CVE name is the 13 character ID used by the CVE standards group to uniquely identify a vulnerability. Additional information about the vulnerabilities also covers preconditions about the target host as well as network preconditions. Furthermore, the impact of an exploitation of a vulnerability is described. The specifications for the formal model of the vulnerabilities additionally comprise the vulnerability *range* and *impact type* assessments provided by the National Institute of Standards and Technology (NIST) (`http://nvd.nist.gov/`). Of course, other kinds of vulnerabilities could be added to the model in a similar manner.

**Vulnerability severity.** The Common Vulnerability Scoring System (CVSS) [3] provides universal severity ratings for security vulnerabilities. These ratings are used in the model as an example for a measure of the threat level. Another example for such a measure is the metric used by the US-CERT (cf. `http://www.kb.cert.org/vuls/html/fieldhelp#metric`).

| Role | Hosts |
|---|---|
| *internet_host* | *attacker* |
| *dmz_host* | *portal* |
| *telework_host* | *telework* |
| *developer_host* | *nix_host* |
| *management_host* | *ms_host* |
| *db_host* | *db_server* |
| *intern_host* | *db_server*,<br>*ms_host*,<br>*nix_host* |

(a) Roles

| Role (source) | View (target) | Activity (service) |
|---|---|---|
| *internet_host* | *internet_host* | *any_activity* |
| *any_role* | *dmz_host* | *ssh* |
| *any_role* | *dmz_host* | *smtp* |
| *dmz_host* | *intern_host* | *ssh* |
| *intern_host* | *any_role* | *net* |
| *intern_host* | *internet_host* | *ftp* |
| *intern_host* | *internet_host* | *rsh* |
| *intern_host* | *dmz_host* | *ssh* |
| *db_host* | *production_host* | *rpc* |
| *teleworker_host* | *dmz_host* | *any_activity* |

(b) Network security policy

**Fig. 3.** Roles and network security policy

These measures are based on information about the vulnerability being widely known, reported exploitation incidents, number of infected systems, the impact of exploiting the vulnerability and the knowledge and the preconditions required to exploit the vulnerability. Because the approximate values included in those measures may differ significantly from one site to another, prioritising of vulnerabilities based on such measures should be used with caution.

To have a vulnerable product installed on some host, does not necessarily imply, that someone can exploit that vulnerability. A target host *is configured vulnerable*, if (1) the target host has installed a product or products that are vulnerable with respect to the given vulnerability, and (2) necessary other preconditions are fulfilled (e.g. some vulnerabilities require that a trust relation is established as for example used in remote shell hosts allow/deny concepts).

The second precondition to exploit a vulnerability is, that the target host *is currently running the respective products* such as a vulnerable operating system or server version. If a user interaction is required this also requires that the vulnerable product is currently used (e.g. a vulnerable Internet explorer).

The third necessary precondition is, that the *network security policy permits* that the target host is reachable on the port the vulnerable product is using from the host the attacker selected as source.

### 2.4 Attacker profile

The knowledge of exploits and hosts and the credentials on the known hosts constitute an attackers profile. Knowledge about hosts changes during the computation of the attack graph because the attacker might gain new knowledge when capturing hosts. On the other hand, some knowledge may become outdated because the enterprise system changes ip-numbers or other configuration of hosts and reachability. In case a vulnerability is exploited, the model has to cover the *effects for the attacker* (e.g. to obtain additional user or root credentials on the target host) and also the *direct impact on the network and host* such as, to shut down a service caused by buffer overflow.

### 2.5 Formal representation of the model

The information model presented so far covers the description of a (static) configuration of an ICT network and its vulnerabilities. In the formal model such a configuration describing the *state* of the ICT network is represented by *APA state components.*

**Definition 1.** *An* Asynchronous Product Automaton (APA) *consists of*

- *a family of* state sets $(Z_s)_{s\in\mathbb{S}}$,
- *a family of* elementary automata $(\Phi_e, \Delta_e)_{e\in\mathbb{E}}$ *and*
- *a* neighbourhood relation $N: \mathbb{E} \to \mathcal{P}(\mathbb{S})$
- *an* initial state $q_0$

$\mathbb{S}$ *and* $\mathbb{E}$ *are index sets with the names of state components and of elementary automata and* $\mathcal{P}(\mathbb{S})$ *is the power set of* $\mathbb{S}$.

*For each elementary automaton* $(\Phi_e, \Delta_e)$ *with* Alphabet $\Phi_e$, *its* state transition relation *is* $\Delta_e \subseteq \bigtimes_{s\in N(e)}(Z_s) \times \Phi_e \times \bigtimes_{s\in N(e)}(Z_s)$*. For each element of* $\Phi_e$ *the state transition relation* $\Delta_e$ *defines state transitions that change only the state components in* $N(e)$*. An APA's (global)* states *are elements of* $\bigtimes_{s\in\mathbb{S}}(Z_s)$*. To avoid pathological cases it is generally assumed that* $\mathbb{S} = \bigcup_{e\in\mathbb{E}}(N(e))$ *and* $N(e) \neq \emptyset$ *for all* $e \in \mathbb{E}$*. Each APA has one* initial state $q_0 = (q_{0s})_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$*. In total, an APA* $\mathbb{A}$ *is defined by*

$$\mathbb{A} = ((Z_s)_{s\in\mathbb{S}}, (\Phi_e, \Delta_e)_{e\in\mathbb{E}}, N, q_0)$$

**Finite state model of the scenario.** The components of the model described previously are now specified for the proposed analysis method using the *APA state components* $\mathbb{S} = \{$ A_known_exploits_state, A_plvl_state, affects_state, configuration_state, host_service_state, host_vulnerability_state, host_vulnerable_user_state, $\ldots$ $\}$.

The initial state is composed of $q_{0A\_known\_exploits\_state}$, $q_{0A\_plvl\_state}$, $\ldots$, where $q_{0A\_known\_exploits\_state}$ contains the exploits known by the attacker,$q_{0A\_plvl\_state}$ contains a sequence of pairs of host and access privileges of the attacker on that host (e.g. (attacker,root), (db_server,none), $\ldots$),$q_{0affects\_state}$ contains a sequence of pairs of vulnerability and affected product (e.g. (CAN_2002_0649,SQL_Server_2000), (CAN_2002_1262,vulnerable_IE), $\ldots$), $q_{0configuration\_state}$ contains a sequence of pairs of a host and a sequence of installed products (e.g. (db_server,W2000_Server.SQL_Server_2000), $\ldots$),$q_{0host\_service\_state}$ contains a sequence of pairs of host and associated service including used port and privileges (e.g. (db_server,((ftpd,ftp_port),root)), (db_server, ((sql_res, ms_sql_m_port), db_user)). $\ldots$), and, $q_{0host\_vulnerability\_state}$ which is empty (the vulnerabilities are computed from $affects\_state$ and $configuration\_state$ in a preprocessing transition).

To describe how actions of attacker(s) and actions of the system can change the state of the ICT network model, specifications of *APA state transitions* are used. These state transitions represent atomic exploits and optionally the actions that the system executes itself (e.g. to implement vital services).

The set of *elementary automata* $\mathbb{E} = \{$ Preprocessor_gen_vulnerabilities, Service_answer, A_select_exploit, Defence_Restart_sshd, A_CAN_2002_1262_IE_caching_exploit, A_CAN_2002_0649_sql_exploit, A_CAN_2003_0694_sendmail_exploit, $\ldots$ $\}$ represents the possible actions. The actions starting with $A\_\ldots$ are the actions the attacker can perform. If multiple attackers are modelled then $A$ is replaced by the name of the attacker.

A state transition can occur, when all expressions are evaluable and all conditions are satisfied. So called *interpretation variables* are used to differentiate the variants of execution of the same transition. All possible variants of bindings of interpretation variables from the state components are generated automatically. So for example for a transition modelling an exploit, all possible combinations of bindings of source and target host are computed and further evaluated.

**Definition 2.** *An elementary automaton* $(\Phi_e, \Delta_e)$ *is* activated *in a state* $q = (q_s)_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$ *as to an* interpretation $i \in \Phi_e$, *if there are* $(p_s)_{s \in N(e)} \in \bigtimes_{s \in N(e)}(Z_s)$ *with* $((q_s)_{s \in N(e)}, i, (p_s)_{s \in N(e)}) \in \Delta_e$. *An activated elementary automaton* $(\Phi_e, \Delta_e)$ *can execute a state transition and produce a* successor state $p = (p_s)_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$, *if* $q_r = p_r$ *for* $r \in \mathbb{S} \setminus N(e)$ *and* $(q_s)_{s \in N(e)}, i, (p_s)_{s \in N(e)} \in \Delta_e$. *The corresponding state transition is* $(q, (e, i), p)$.

A state transition in the given model could for example cause a change in the state component *A_plvl_state* from (attacker,root).(db_server,none)... into (attacker,root)(db_server,root)....

**Attacker behaviour.** Attacker capabilities are modelled by the atomic exploits and by the strategy to select and apply them.

A state transition modelling an exploit is constructed from, (1) a predicate that states that the attacker *knows* this exploit, (2) an expression to select source and target hosts for the exploit, (3) a predicate that states that the target *host is vulnerable* by this exploit, (4) an expression for the impact of the execution of this exploit on the attacker and on the target host as for example the shut down of services. Optional add-ons are, an assignment of cost benefit ratings to this exploit and intrusion detection checks.

Several different attackers can easily be included because an attacker is modelled as a role not a single instance and the tool can automatically generate multiple instances from one role definition.

Modelling of Denial of Service (DoS) attacks aiming to block resources or communication channels either directly or by side effects require a much more detailed model of the resources involved. This could be accomplished using the presented framework but is out of scope of this paper.

Some experiments have been made to generate a set of known exploits for the attacker(s) from a given algorithm. If for example it is assumed that the attacker knows 3 different exploits, then all combinations of 3 exploits from the set of all specified exploits have to be computed and further analysed. Another example for an attacker strategy is, that the attacker uses only exploits for vulnerabilities with a severity above a given threshold. This is based on the assumption, that the vulnerability severity reflects the probability of exploitation of a vulnerability.

**Composition of a model and computation of an attack graph.** The SH verification tool [4] is used to analyse this model. It manages the components of the model, allows to select alternative parts of the specification and automatically "glues" together the selected components to generate a combined model of ICT network specification, vulnerability and exploit specification, network security policy and attacker specification. After an initial

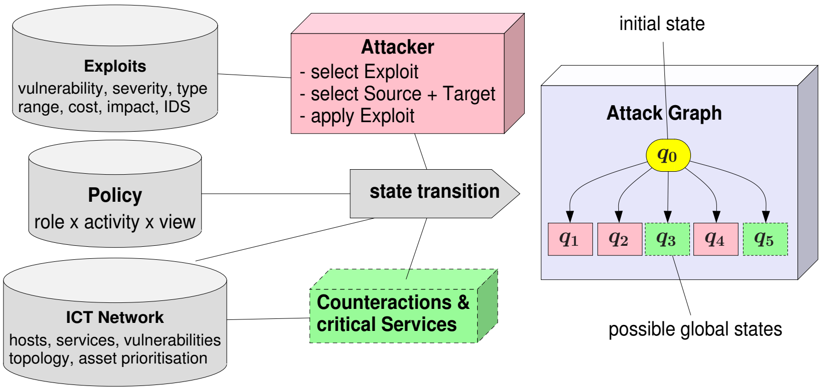


**Fig. 4.** Computation of the attack graph

configuration is selected, the attack graph (reachability graph) is automatically computed by the SH verification tool (see Fig. 4). Formally, the attack graph is the reachability graph of the corresponding APA model.

**Definition 3.** *The behaviour of an APA is represented by all possible coherent sequences of state transitions starting with initial state $q_0$. The sequence $(q_0, (e_1, i_1), q_1)$ $(q_1, (e_2, i_2), q_2)$ $(q_2, (e_3, i_3), q_3) \ldots$ $(q_{n-1}, (e_n, i_n), q_n)$ with $i_k \in \Phi_{e_k}$ represents one possible sequence of actions of an APA. $q_n$ is called the* goal *of this action sequence.*

*State transitions $(p, (e, i), q)$ may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA: $(p, (e, i), q)$ is the edge leading from $p$ to $q$ and labelled by $(e, i)$. The subgraph reachable from the node $q_0$ is called the* reachability graph *of an APA.*

*Let $\mathbb{Q}$ denote the set of all states $q \in \bigtimes_{s\in\mathbb{S}}(Z_s)$ that are reachable from the initial state $q_0$ and let $\Psi$ denote the set of all state transitions with the first component in $\mathbb{Q}$.*

*The set $L \subset \Psi^*$ of all action sequences with initial state $q_0$ including the empty sequence $\epsilon$ denotes the* action language *of the corresponding APA. The action language is prefix closed. By definition $q_0$ is the goal of $\epsilon$.*

**Attack graph of the example scenario.** The computed attack graph for the simple example scenario is shown in Fig. 5. This graph was computed under the assumption that the attacker knows all exploits. Even if we assume as a more realistic attacker behaviour, namely that the attacker will only exploit vulnerabilities with a severity level above a given minimum, then the graph is still far too big to inspect it manually. Figure 6 shows a detail of this attack graph. Please note that for better readability the interpretations are omitted at the edge labels. For example the edge $q_{13} \longrightarrow q_{38}$ depicts the application of an exploit where attacker A uses the *ssh* vulnerability CAN_2003_0693 and there is a second exploit (which is stealth, that means not detectable by intrusion detection systems) with the same state transition. The edge $q_{38} \longrightarrow q_{73}$ depicts an action of the system to restart the *ssh*

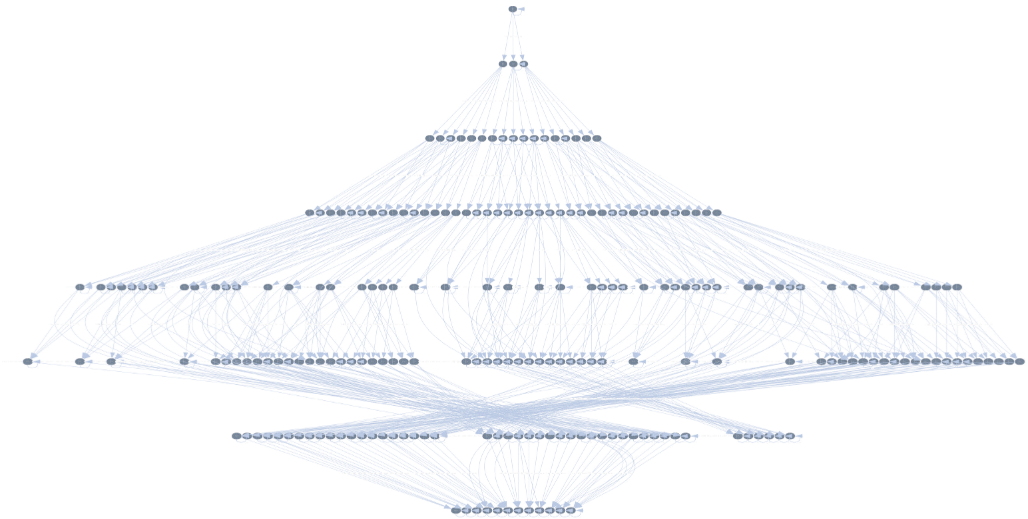
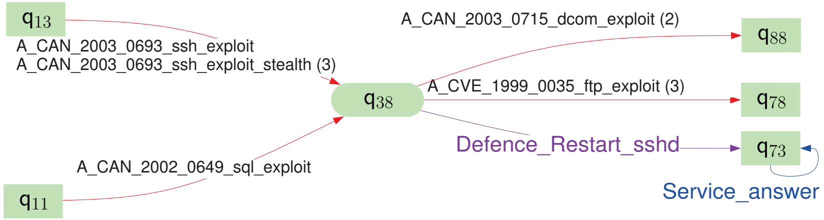


**Fig. 5.** Attack graph of simple example scenario

daemon and the edge $q_{73} \longrightarrow q_{73}$ depicts an action that models the availability of a critical service.



q13
A_CAN_2003_0693_ssh_exploit
A_CAN_2003_0693_ssh_exploit_stealth (3)
A_CAN_2003_0715_dcom_exploit (2)
q88
q38
A_CVE_1999_0035_ftp_exploit (3)
q78
Defence_Restart_sshd
q73
A_CAN_2002_0649_sql_exploit
Service_answer
q11

**Fig. 6.** Attack graph detail

For very large models, on the fly analysis allows to stop computation of the reachability graph automatically when specified conditions are reached or invariants are broken.

## 3 Evaluation of the model

### 3.1 Cost benefit analysis

Cost benefit analysis can be used as a means to help to assess the likely behaviour of an attacker. Cost ratings (from the view of an attacker) can be assigned to each exploit, for example to denote the time it takes for the attacker to execute the exploit or the resources needed to develop an exploit. Cost ratings can also be based on the severity ratings given by CVSS or US-CERT (cf. Sect.2.3). If not only technical vulnerabilities are modelled but also human weaknesses are considered, then cost could mean for example the money needed to *buy* a password.

Based on these cost assignments (weights of edges), the shortest path from the root of the attack graph to a node representing a successful attack can be computed using Dijkstra's well-known algorithm. This path represents the least expensive combined attack breaking a given security goal.

A benefit for the attacker based on the negative impact he achieves can also be assigned, for example to indicate the *worth* regarding relative criticality of the captured asset. Summarised costs and benefits can be compared for selected paths or the whole graph and used for example to find the node with the greatest benefit for a potential attacker. Please refer to [5] for an example.

Other security related properties such as the probability of being detected by intrusion detection systems can be associated with APA transitions. This information when evaluated in the analysis of an attack graph can lead to improvements of a given configuration of a critical information infrastructure.

### 3.2 Abstraction-based analysis

**Abstract representations.** The SH verification tool usually computes graphs of about 1 million edges in acceptable time and space. The problem now is, that it is impossible to visualise a graph of that size. However abstract representations of an attack graph can be computed and used to visualise and analyse compacted information focussed on interesting aspects of the behaviour.

Behaviour abstraction of an APA can be formalised by language homomorphisms, more precisely by *alphabetic language homomorphisms* $h : \Sigma^* \to \Sigma'^*$ on the action language.

By these homomorphisms certain transitions are ignored and others are renamed, which may have the effect, that different transitions are identified with one another. A mapping $h : \Sigma^* \to \Sigma'^*$ is called a *language homomorphism* if $h(\epsilon) = \epsilon$ and $h(yz) = h(y)h(z)$ for each $y, z \in \Sigma^*$. It is called *alphabetic*, if $h(\Sigma) \subset \Sigma' \cup \{\epsilon\}$.

The mappings used to compute the abstract representations of the behaviour have to be *property preserving*, to assure that properties are *transported* as desired from a lower to a higher level of abstraction and no critical behaviour is hidden by the mapping. Such properties, namely *simplicity*, are given in [6] and a check for simplicity is implemented in the SH verification tool [4]. The tool provides an editor to define homomorphisms on action languages, it computes corresponding minimal automata [7] for the homomorphic images and checks simplicity of the homomorphisms.

**Example of a mapping to define an abstract representation.** Figure 7 defines a mapping of the transitions representing an exploit of a vulnerability to the respective *range* and *impact type* assessments of the vulnerabilities as provided by NIST (cf. Sect. 2.3). Range types of the vulnerabilities in the example scenario are *remote* (remotely exploitable) and *local* (locally exploitable). Impact types used here are *unspecific* (provides unauthorised access), *user* (provides user account access) and *root* (provides administrator access).

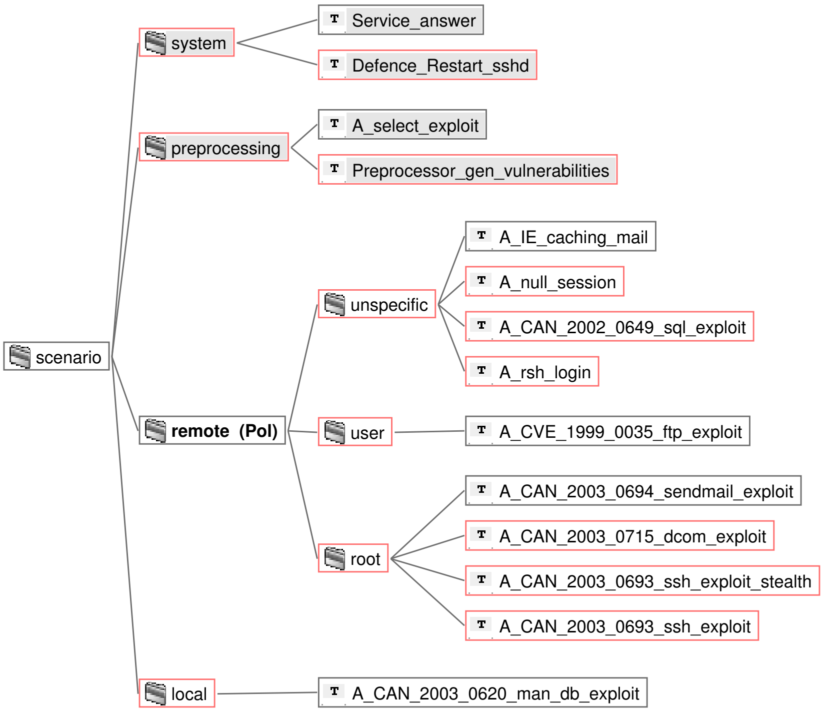

scenario
system
Service_answer
Defence_Restart_sshd
preprocessing
A_select_exploit
Preprocessor_gen_vulnerabilities
remote (Pol)
unspecific
A_IE_caching_mail
A_null_session
A_CAN_2002_0649_sql_exploit
A_rsh_login
user
A_CVE_1999_0035_ftp_exploit
root
A_CAN_2003_0694_sendmail_exploit
A_CAN_2003_0715_dcom_exploit
A_CAN_2003_0693_ssh_exploit_stealth
A_CAN_2003_0693_ssh_exploit
local
A_CAN_2003_0620_man_db_exploit

**Fig. 7.** Definition of an abstract representation of the attack graph

This mapping denotes, that all transitions (the leaves of the tree) are to be represented by their respective father nodes, namely *system*, *preprocessing*, $unspecific$, *user*, *root* and *local* in the abstract representation. The nodes *system* and *preprocessing* are coloured in grey, symbolising that they are mapped to $\epsilon$, that means the transitions represented by these nodes will be invisible in the abstract representation. Please ignore the notation ($Pol$) at the node *remote* for the moment.

Figure 8 shows the result of application of the mapping in Fig. 7 to the attack graph from Fig. 5. This computed abstract representation (a graph with only 20 states and 37 edges) gives a visualisation focussing on the transition types *root*, *user*, $unspecific$ and *local*. The simplicity of this mapping that guarantees that properties are preserved was automatically proven by the tool.

**Refined mapping.** To find out which policies permit the attacks shown in Fig. 8, a refinement of the abstraction defined in Fig. 7 is necessary. It is possible to "fine tune" the mapping so that the interpretation variables (cf. Sect. 2.5) stay visible in the abstract representation. In this case the binding of the interpretation variable $Pol$ that contains the respective policy can be visualised. This is denoted by ($Pol$) in the node *remote* in Fig. 7. The corresponding refined abstract representation is a graph with 34 states and 121 edges when computed on the attack graph in Fig. 5. The initial nodes and edges of this graph

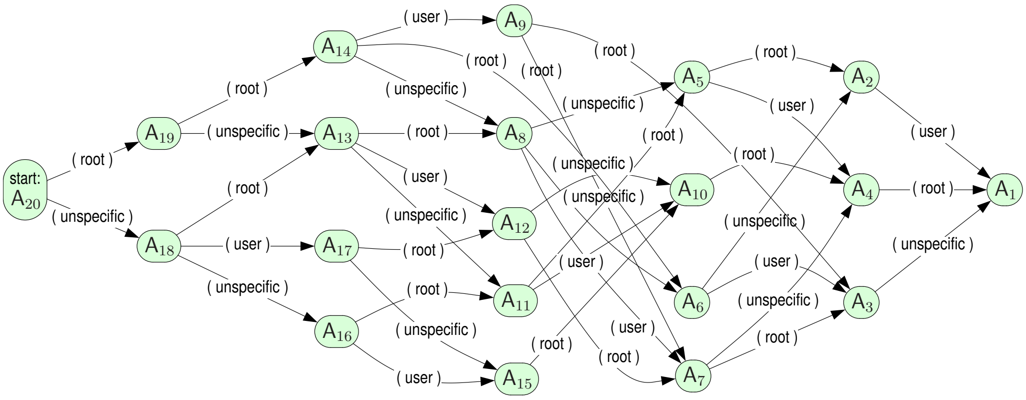
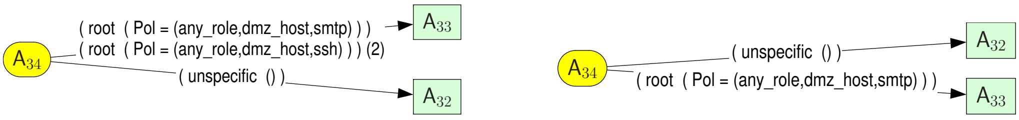


**Fig. 8.** Abstract view on an attack graph

are shown in Fig. 9(a). In comparison to the corresponding edges $A_{20} \longrightarrow A_{19}$ and $A_{20} \longrightarrow$



A34
( root ( Pol = (any_role,dmz_host,smtp) ) )
( root ( Pol = (any_role,dmz_host,ssh) ) ) (2)
( unspecific () )
A33
A32
A34
( unspecific () )
( root ( Pol = (any_role,dmz_host,smtp) ) )
A32
A33

(a) $(any\_role, dmz\_host, ssh/smtp)$

(b) $(any\_role, dmz\_host, smtp)$

**Fig. 9.** Details in the abstract view

$A_{18}$ of the graph in Fig. 8 now the details on the related policies are visible.

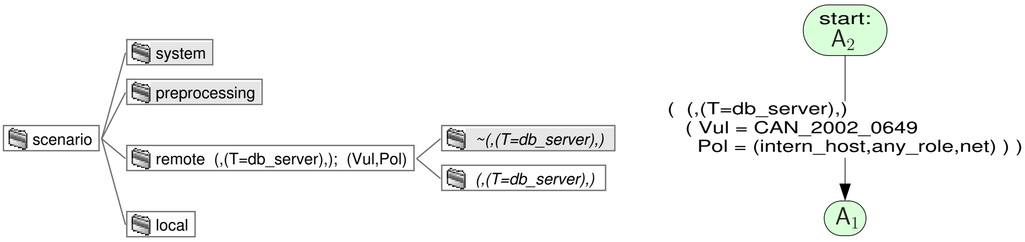
**Adapt/optimise the system configuration.** Further analysis reveals, that, if the example policy given in Fig. 3(b) is changed to allow only *smtp* instead of *ssh* and *smtp* for *any_role* to *dmz_host* then the analysis yields of course a smaller graph than the original shown in Fig.5, the coarse abstract representation in Fig. 8 is the same, but the finer mapping with interpretation variable *Pol* visible results in a different representation which is shown in Fig. 9(b).

If alternatively the policy is restricted to allow only *ssh* instead of *ssh* and *smtp* in the above example, then the result is yet a different attack graph but the abstract view in Fig. 8 is still the same.

This analysis demonstrates that there may be differences in the detailed attack graphs but no differences in the abstract representations thereof. This indicates that the different policies are equally effective (or not) concerning the enforcement of security goals on the abstract level, even if variations in the attack paths are covered by different policy rules.

**Using predicates to define abstractions.** Let us now assume that the host *db_server* in the scenario is the most valuable and mission critical host in the ICT network. So we want to know if in the given scenario, (1) attacks to the db_server are possible, (2) on which vulnerabilities they are based, and, (3) what policy rules are directly involved.

The abstraction in Fig. 10(a) exemplifies how predicates can be used to define such a mapping. In this mapping the predicate (T=db_server) matches only those transitions that model direct attacks to the target host db_server. Furthermore the bindings of the interpretation variables $Vul$ and $Pol$ that contain the respective vulnerability and policy are used in the mapping. The remote transitions that don't match that predicate are mapped to $\epsilon$ and so are invisible.



(a) Abstraction used

(b) Resulting graph

**Fig. 10.** Focus on attacks to the host *db_server*

Evaluating this abstraction on the attack graph from Fig. 5 above results in the simple graph given in Fig. 10(b). This proves that, (1) in the current policy configuration attacks to the db_server are possible, (2) those attacks are based on exploits of the vulnerability CAN_2002_0649, and, (3) they are utilising the policy permission (intern_hosts,any_role,net). So to prevent this attack, it has to be decided, whether it is more appropriate to uninstall the product that is hurt by this vulnerability or to restrict the internal hosts in their possible actions by replacing the above policy with a more restrictive one.

### 3.3 Liveness properties

As it is well known, system properties are divided into two types: safety (what happens is not wrong) and liveness properties (eventually something desired happens) [8]. Liveness properties in the context of ICT infrastructure security analysis cover availability and business continuity aspects for example with respect to denial of service attacks.

On account of liveness aspects system properties are formalised by $\omega$-languages (sets of infinite long words). So to investigate satisfaction of properties "infinite system behaviour" has to be considered. This is formalised by so called Eilenberg limits of action languages (more precisely: by Eilenberg limits of modified action languages where maximal words are continued by an unbounded repetition of a dummy action) [9].

The usual concept of linear satisfaction of properties (each infinite run of the system satisfies the property) is not suitable in this context because no fairness constraints are considered. We put a very abstract notion of fairness into the satisfaction relation for properties, which considers that independent of a finitely long prefix computation of a system certain desired events may occur eventually. To formalise such "possibility properties", which are of interest when considering what we call cooperating systems, the notion of approximate satisfaction of properties is defined in [9].

**Definition 4.** *A system* approximately satisfies *a property if and only if each finite behaviour can be continued to an infinite behaviour, which satisfies the property.*

For safety properties linear satisfaction and approximate satisfaction are equivalent [9]. To deduce approximately satisfied properties of a specification from properties of its abstract behaviour an additional property of abstractions called simplicity of homomorphisms on an action language [10] is required. Simplicity of homomorphisms is a very technical condition concerning the possible continuations of finite behaviours.

For regular languages simplicity is decidable. In [10] a sufficient condition based on the strongly connected components of corresponding automata is given, which easily can be checked. Especially: If the automaton or reachability graph is strongly connected, then each homomorphism is simple. The following theorem [9] shows that approximate satisfaction of properties and simplicity of homomorphisms exactly fit together for verifying cooperating systems.

**Theorem 1.** Simple homomorphisms *define exactly the class of such abstractions, for which holds that each property is approximately satisfied by the abstract behaviour if and only if the "corresponding" property is approximately satisfied by the concrete behaviour of the system.*

Formally, the "corresponding" property is expressed by the inverse image of the abstract property with respect to the homomorphism.

When a system's countermeasures and the behaviour of vital services the system provides are included in the model, then availability properties such as the system's resilience with respect to denial of service attacks can be analysed.

The state transitions $Defence\_restart\_sshd$ and $Service\_answer$ in Fig. 6 give an example for a modelling of system countermeasures and critical services availability. If for example as a side effect of an *ssh_exploit* the attacker kills the *sshd* then afterwards the *sshd* is not active on the respective host and so some service possibly cannot answer requests anymore. Now additionally a system countermeasure is considered that restarts the *sshd*. No other details are added to keep the model small. A typical liveness question in this scenario is "Will a client still get answers from a server when the network is attacked ?". Using the appropriate type of model checking, *approximate satisfaction* of temporal logic formulae can be checked by the SH verification tool [11], [4]. In terms of temporal logic the property in question above can be written as $G\ F\ Service\_answer$ (always eventually $Service\_answer$) which is found to be *true* by the tool.

## 4 Resilience against exploits of unknown vulnerabilities

One way to consider resilience of an information infrastructure against attacks to unknown vulnerabilities is, to define a new vulnerability for each installed product. For the model of the scenario used in this paper this has been done by definition of a new vulnerability called CAN_generic with a variable part for the affected service. In the same way a generic exploit based on this vulnerability is defined. Now in the preprocessing phase a state transition selects an arbitrary product and inserts a generic vulnerability CAN_generic for that product and the related service. Because the reachability analysis considers every possible choice of product, all alternatives are evaluated in the attack graph.

When analysing the (now much larger) attack graph, the mapping in Fig. 11 exemplarily shows a possible use of resilience analysis. The state transition modelling an exploit of an unknown generic vulnerability uses the additional interpretation variables $RS$ and $RT$, where $RS$ denotes the role of the source host and $RT$ the role of the target host. So the given predicate $(RS = RT)$ matches only those transitions that model attacks of hosts in the same role (within the same zone). Now the attacks that fulfil this predicate are mapped to $\epsilon$ (coloured in grey in the mapping) and so are invisible, whereas the attacks with $RS \neq RT$ (across roles/zones) are visible. Furthermore the bindings of the interpretation variables $VulServ$ and $Pol$ that contain the respective vulnerable service and policy are used in the mapping. All other transitions are mapped to $\epsilon$.

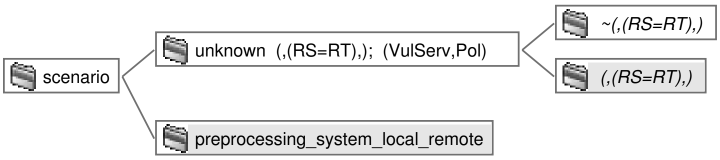

scenario
unknown (,(RS=RT),); (VulServ,Pol)
~(,(RS=RT),)
(,(RS=RT),)
preprocessing_system_local_remote

**Fig. 11.** Mapping for attacks against unknown vulnerabilities that cross zones

The abstract representation computed from that mapping is shown in Fig. 12. It gives a clear overview about what kind of zone crossing attacks would be possible in case that new unknown vulnerabilities were exploited. For each assumed vulnerable service it shows the policies that would allow the attack.

Using a modified definition of the mapping in Fig. 11 allows to look into more detail for example behind the edge $(VulServ = sendmaild\ Pol = (intern\_host, any\_role, net))$ shown in bold font in Fig. 12. A refined mapping with predicate $(RS \neq RT\ \wedge\ VulServ = sendmaild)$ and visible interpretation variables $RS$ and $RT$ results in an abstract representation with 4 parallel edges labelled $(RS = developer\_host\ RT = dmz\_host)$, $(RS = db\_host\ RT = dmz\_host)$, $(RS = intern\_host\ RT = dmz\_host)$ and $(RS = management\_host\ RT = dmz\_host)$ respectively. This shows that if an attacker knows a new exploit for an unknown vulnerability of the product providing the *sendmaild*, then the current policy rule $(intern\_host, any\_role, net)$ would allow to use the exploit to cross the 4 given zones.

Now if the policies are quite restrictive and no new cross role/zone attacks are found by the reachability analysis, then it can be concluded that the network configuration is resilient

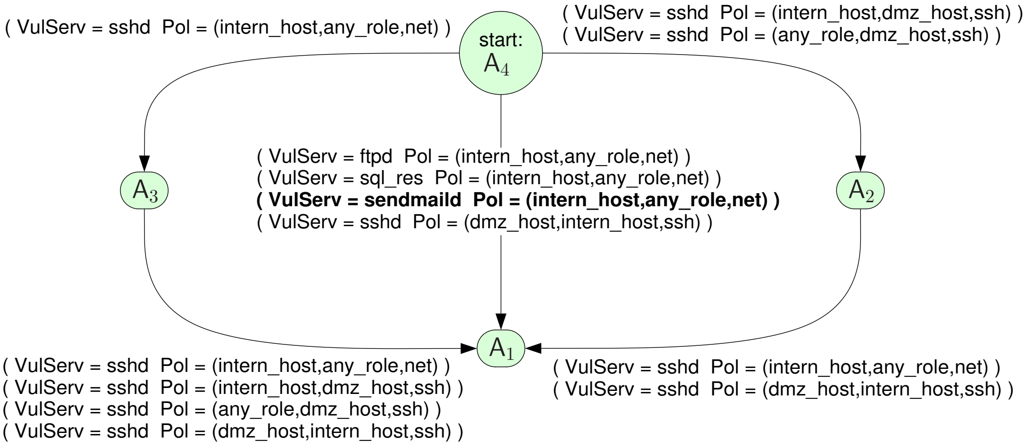

( VulServ = sshd  Pol = (intern_host,any_role,net) )
start:
$A_4$
( VulServ = sshd  Pol = (intern_host,dmz_host,ssh) )
( VulServ = sshd  Pol = (any_role,dmz_host,ssh) )
$A_3$
( VulServ = ftpd  Pol = (intern_host,any_role,net) )
( VulServ = sql_res  Pol = (intern_host,any_role,net) )
( VulServ = sendmaild  Pol = (intern_host,any_role,net) )
( VulServ = sshd  Pol = (dmz_host,intern_host,ssh) )
$A_2$
$A_1$
( VulServ = sshd  Pol = (intern_host,any_role,net) )
( VulServ = sshd  Pol = (intern_host,dmz_host,ssh) )
( VulServ = sshd  Pol = (any_role,dmz_host,ssh) )
( VulServ = sshd  Pol = (dmz_host,intern_host,ssh) )
( VulServ = sshd  Pol = (intern_host,any_role,net) )
( VulServ = sshd  Pol = (dmz_host,intern_host,ssh) )

**Fig. 12.** Abstract representation of attacks against unknown vulnerabilities

with respect to attacks against *one* unknown vulnerability. In the same way resiliency with respect to two or more unknown vulnerabilities can be analysed. Please note that in many cases this will not be possible because of state space explosion problems but computation of a section of the attack graph by giving a limitation on the number of edges to be computed is possible and should help to find problems and to successively restrict the configuration to an acceptable risk level.

## 5 Related work

This paper is based on the work presented in [12]. The network vulnerability modelling part of the framework presented here is adopted from the approach introduced in [5] and is similar in design to an approach by Phillips and Swiler in [13] and [14]. Related work of Jha, Sheyner, Wing et al. used attack graphs that are computed and analysed based on model checking in [15] and [16]. Ammann et al. presented an approach in [17] that is focussed on reduction of complexity of the analysis problem by explicit assumptions of monotonicity.

To seamlessly integrate the methods and tool presented here into a network vulnerability analysis framework, a tool-assisted transformation of up-to-date ICT system configuration and vulnerability databases into a formal specification of the model is required. This should preferably be based on automatically updated information of network scanners because administration databases are typically out-of-date. Recent work by Noel, Jajodia et al. in [18] and [19] already covers this aspect and also describes attack graph visualisation techniques.

The work of Kotenko and Stepashkin in [20] is focussed on security metrics computations and adaptive cooperative defence mechanisms [21].

To model the ICT network, the vulnerabilities and the intrusion detection systems, a data model loosely resembling the formally defined M2D2 information model [1] is used. Appropriate parts of this model are adopted and supplemented by concepts needed for description of exploits, attacker knowledge and strategy and information for cost benefit analysis. A formal approach to use an Organisation-Based Access Control (Or-BAC) model to specify network security policies was presented in [2]. This approach is adopted here to model the network security policies in the attack graph analysis framework.

The modelling framework is based on *Asynchronous Product Automata (APA)*, a flexible operational specification concept for cooperating systems [11]. The applied analysis method is implemented in the SH verification tool [4] that has been adapted and extended to support the presented attack graph evaluation methods.

Major focus of the combined modelling framework presented in this paper, is the integration of formal network vulnerability modelling on the one hand and network security policy modelling on the other hand. This aims to help adaptation of a network security policy to a given and possibly changing vulnerability setting. Recent methods for analysis of attack graphs are extended to support analysis of abstract representations of these graphs.

Extensions to the APA-based model checking techniques are needed to be able to verify entire families of systems, independent of the exact number of replicated components. Such an approach to abstraction-based analysis of parameterised policy controlled systems is presented in [22].

## 6 Further research objectives

The work presented in this paper brings together, (1) attack graph computation technology, (2) state-of-the-art policy modelling, and, (3) formal methods for analysis and computation of abstract representations of the system behaviour. The aim is, to guide a systematic evaluation and assist the persons in charge with optimising adaptation of the network security policy to an ever-changing vulnerability setting and so to improve the configuration of the information infrastructure.

**(Security) metrics in abstract representations.** A summarisation of severity ratings for single security vulnerabilities as provided by CVSS or US-CERT (cf. Sect. 2.3) based on attack graphs has been addressed in recent work of Kotenko and Stepashkin [20]. Interesting questions in such an approach are, which attacker strategy or bundle of strategies to apply and how to “condense” the information in the graph into a comprehensive measure of the security of an ICT network.

In an abstraction-based approach a method to assign measures to the nodes or edges in the abstract representations is needed. One idea is to look at the origin nodes (nodes in the attack graph) of an abstract node and then for example compute the minimum value of some measure from the set of origin nodes. If for example a shortest path analysis (cf. Sect.3.1) was computed on the attack graph, then each node of the attack graph is associated with a value for the shortest (least expensive) path to that node. If now the semantics of these values allows a comparison, then a function such as the minimum measure from the set of origin nodes can be associated with each node in the abstract representation. If the cost ratings of the transitions in the attack graph are based on severity ratings from CVSS or US-CERT (cf. Sect.2.3) then the function for the transformation of the values in the origin nodes to the abstract representation can be used for a metric of security of the critical information under the abstract view defined by the mapping. Consideration of *resilience* *against exploits of unknown vulnerabilities* (cf. Sect. 4) could also contribute to such a measure.

**Threat response mechanisms.** An even more advanced objective is, to extend this framework to support *policy-based, automated threat response* that makes use of alert information. Such a self-adaptive response mechanism could substantially improve the resilience of policy controlled ICT systems against network attacks. A framework for simulation of adaptive cooperative defense against internet attacks has been presented by Kotenko and Ulanov in [21]. Analysis of distributed coordinated attacks and the controlling mechanisms such as botnets using the abstraction-based approach presented in this paper would complement their approach.

**Acknowledgements.** The author likes to thank Peter Ochsenschläger who conducted most of the theoretical work on APA that this work is based on and Jürgen Repp who implemented most of the SH verification tool that was used as basis for the modelling and analysis presented here.

Part of the work presented in this paper was developed within the project SicAri being funded by the German Ministry of Education and Research.

## References

1. Morin, B., Mé, L., Debar, H., Ducassé, M.: M2d2: A formal data model for ids alert correlation. In: Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002, Zurich, Switzerland, October 16-18, 2002, Proceedings. Volume 2516 of Lecture Notes in Computer Science., Springer (2002) 115–137
2. Cuppens, F., Cuppens-Boulahia, N., Sans, T., Miège, A.: A formal approach to specify and deploy a network security policy. In: Second Workshop on Formal Aspects in Security and Trust (FAST). (2004)
3. Schiffmann, M.: A Complete Guide to the Common Vulnerability Scoring System (CVSS) (2005) http://www.first.org/cvss/cvss-guide.html.
4. Ochsenschläger, P., Repp, J., Rieke, R.: The SH-Verification Tool. In: Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000), Orlando, FL, USA, AAAI Press (2000) 18–22
5. Rieke, R.: Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration. In: In U.E. Gattiker (Ed.), Eicar 2004 Conference CD-rom: Best Paper Proceedings, Copenhagen, EICAR e.V. (2004)
6. Ochsenschläger, P., Repp, J., Rieke, R.: Verification of Cooperating Systems – An Approach Based on Formal Languages. In: Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000), Orlando, FL, USA, AAAI Press (2000) 346–350
7. Eilenberg, S.: Automata, Languages and Machines. Volume A. Academic Press, New York (1974)
8. Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters **21**(4) (1985) 181–185
9. Nitsche, U., Ochsenschläger, P.: Approximately satisfied properties of systems and simple language homomorphisms. Information Processing Letters **60** (1996) 201–206

10. Ochsenschläger, P.: Verification of cooperating systems by simple homomorphisms using the product net machine. In Desel, J., Oberweis, A., Reisig, W., eds.: Workshop: Algorithmen und Werkzeuge für Petrinetze, Humboldt Universität Berlin (1994) 48–53
11. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing, The International Journal of Formal Method **11** (1999) 1–24
12. Rieke, R.: Modelling and Analysing Network Security Policies in a Given Vulnerability Setting. In: Critical Information Infrastructures Security, First International Workshop, CRITIS 2006, Samos Island, Greece. Volume 4347 of LNCS., Springer (2006) 67–78 © Springer.
13. Phillips, C.A., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: NSPW '98, Proceedings of the 1998 Workshop on New Security Paradigms, September 22-25, 1998, Charlottsville, VA, USA, ACM Press (1998) 71–79
14. Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S.: Computer-attack graph generation tool. In: DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume 2,June 12 - 14, 2001, Anaheim, California, IEEE Computer Society (2001) 1307–1321
15. Jha, S., Sheyner, O., Wing, J.M.: Two formal analyses of attack graphs. In: 15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada, IEEE Computer Society (2002) 49–63
16. Sheyner, O., Haines, J.W., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: 2002 IEEE Symposium on Security and Privacy, May 12-15, 2002, Berkeley, California, USA, IEEE Comp. Soc. Press (2002) 273–284
17. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM conference on Computer and communications security, ACM Press New York, NY, USA (2002) 217–224
18. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, New York, NY, USA, ACM Press (2004) 109–118
19. Noel, S., Jacobs, M., Kalapa, P., Jajodia, S.: Multiple Coordinated Views for Network Attack Graphs. In: IEEE Workshop on Visualization for Computer Security (VizSec'05), Los Alamitos, CA, USA, IEEE Computer Society (2005)
20. Kotenko, I., Stepashkin, M.: Analyzing Network Security using Malefactor Action Graphs. International Journal of Computer Science and Network Security **6** (2006)
21. Kotenko, I., Ulanov, A.: Multi-agent Framework for Simulation of Adaptive Cooperative Defense against Internet Attacks. In: In Proceedings of International Workshop on Autonomous Intelligent Systems: Agents and Data Mining (AIS-ADM-07). Lecture Notes in Artificial Intelligence, Vol.4476. (2007)
22. Ochsenschläger, P., Rieke, R.: Abstraction Based Verification of a Parameterised Policy Controlled System. In: International Conference "Mathematical Methods, Models and Architectures for Computer Networks Security" (MMM-ACNS-07). Volume 1 of CCIS., Springer (2007) © Springer.

P12

# A HOLISTIC APPROACH TO SECURITY POLICIES – POLICY DISTRIBUTION WITH XACML OVER COPS

| | |
|---|---|
| **Title** | A Holistic Approach to Security Policies – Policy Distribution with XACML over COPS |
| **Authors** | Jan Peters, Roland Rieke, Taufiq Rochaeli, Björn Steinemann, and Ruben Wolf |
| **Publication** | In *Proc. of the Second International Workshop on Views On Designing Complex Architectures (VODCA 2006)*, volume 168, pages 143–157. |
| **ISBN/ISSN** | ISSN 1571-0661 |
| **DOI** | `http://dx.doi.org/10.1016/j.entcs.2006.08.025` |
| **Status** | Published |
| **Publisher** | Elsevier |
| **Publication Type** | Journal: Electronic Notes in Theoretical Computer Science |
| **Copyright** | 2007, Elsevier |
| **License** | This article is published under the terms of the Creative Commons Attribution-NonCommercial-No Derivatives License (CC BY NC ND). To view a copy of this license, visit `http://creativecommons.org/licenses/by-nc-nd/3.0/`. |
| **Contribution of Roland Rieke** | Co-Author with significant contribution.<br>Specific contributions are: (1) design of the policy provisioning framework for the SicAri [Peters, 2013; Rieke & Ebinger, 2008] platform architecture (distribution of XACML policies using COPS) [Rieke, 2004a; Peters et al., 2005], (2) design of the policy validation component for the SicAri platform [Repp et al., 2005], and (3) contributions to the overall policy architecture of the SicAri platform [Heinemann et al., 2006]. |

Table 17: Fact Sheet Publication *P12*

Publication P12 [Peters, Rieke, Rochaeli, Steinemann & Wolf, 2007] addresses the following research question:

RQ8 *Does a policy correctly implement high-level security goals?*

The potentials of modern information technology can only be exploited, if the underlying infrastructure and the applied applications sufficiently take into account all aspects of IT security. This paper presents the policy architecture of the SicAri project [Rieke & Ebinger, 2008; Peters, 2013] that aims to build a security platform for ubiquitous Internet usage, and gives an overview of the implicitly and explicitly used security mechanisms to enable access control for service oriented applications in distributed environments. The paper introduces the security policy integration concept with a special focus on distribution of security policies within the service infrastructure for transparent policy enforcement. Specifically, extensions to the COPS protocol to transport XACML payload for security policy distribution and policy decision requests/responses are described.

Available online at www.sciencedirect.com
ScienceDirect

Electronic Notes in Theoretical Computer Science 168 (2007) 143–157

Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

# A Holistic Approach to Security Policies – Policy Distribution with XACML over COPS

Jan Peters[a] Roland Rieke[b] Taufiq Rochaeli[c]
Björn Steinemann[b] Ruben Wolf[b]

[a] *Fraunhofer Institute for Computer Graphics Research IGD, Germany*

[b] *Fraunhofer Institute for Secure Information Technology SIT, Germany*

[c] *Technical University of Darmstadt, Department of Computer Science, IT-Security group, Germany*

**Abstract**

The potentials of modern information technology can only be exploited, if the underlying infrastructure and the applied applications sufficiently take into account all aspects of IT security. This paper presents the platform architecture of the SicAri project, which aims to build a security platform for ubiquitous Internet usage, and gives an overview of the implicitly and explicitly used security mechanisms to enable access control for service oriented applications in distributed environments. The paper will introduce the security policy integration concept with a special focus on distribution of security policies within the service infrastructure for transparent policy enforcement. We describe in details our extensions of the COPS protocol to transport XACML payload for security policy distribution and policy decision requests/responses.

*Keywords:* service platform, security policies, policy distribution, policy decision, policy enforcement, web services, XACML, COPS

## 1 Introduction

Professional usage of today's communication and collaboration infrastructures requires the consideration of appropriate security measures. In this paper, we introduce the SicAri [4] project – an interdisciplinary approach to information security. The project covers technical, cryptographic and usability issues, as well as various legal issues in information security with respect to legislation and jurisdiction. The overall goal is the conception and realization of a Java-based security platform and its tools for ubiquitous Internet usage.

This platform supplies a bunch of applications and provides various security services to the user in a transparent, seamless and integrated way. It is a modular and integrative platform that allows the connection of various end user devices, such as PCs, PDAs, and ambient intelligence devices and gives support for various network types (e. g., wired and wireless networks) and communication paradigms

doi:10.1016/j.entcs.2006.08.025

(e. g., client-server, peer-to-peer or ad-hoc networks). The behavior of the platform in terms of security related action can be determined by security policies. They provide a well-understood and suitable means to administer security issues. Such policies allow to separate the administration, decision finding, and enforcement of access control. But using policies raise additional questions in distributed environments where applications, services and nodes dynamically join and leave the system. The distribution of policies, especially at bootstrapping time, and their update and synchronization process has to be particularly considered. While open standards and open source solutions for single isolated tasks around security policies exist we have not been aware of any open holistic approach to them.

This paper first presents the SicAri platform from a conceptual, an architectural, and partially from an implementation point of view, but the main focus is on our distribution model of the underlying platform security policy which is based on the *Common Open Policy Service (COPS)* protocol. We extended the COPS protocol with a client type specification to transport *eXtensible Access Control Markup Language (XACML)* policies.

The subsequent paper is structured as follows. Section 2 gives an overview of the layers and the components of the platform. The holistic security policy approach is described in Section 3, while Section 4 presents the policy processing including policy enforcement, policy decision and policy distribution. Section 5 considers some related work. Finally, the paper ends with an outlook in Section 6.

## 2 The SicAri Platform

The main function of the *SicAri platform* is to provide interfaces to the user's application to access the services provided by the platform, which are basic services and application services (see below). Together with the middleware, the platform also provides the communication infrastructure between distributed components. This is realized through a consistent service management comprising service discovery, service description, and service invocation. If desired, local services are automatically provided as Web services to remote platforms during runtime, which enables the interoperability of our service platform with other service-oriented architectures.

In case of new users requiring new application services, the platform architecture supports modular and extensible building blocks, adding the possibility to incorporate new services. Therefore, reusability of existing building blocks should be as easy as possible for developers.

All security related aspects of a platform instance are regulated by a security policy. In the case of multiple platforms interacting with each other, all platforms running in the same *SicAri infrastructure* share the same security policy. Each platform has its own policy enforcement component. Access attempts to local or remote services and resources are checked against the security policy considering the requester's current session ID with respect to a single-sign-on mechanism, activated roles, the available permissions, and other parameters. In addition to policy decision and enforcement, the platform further covers the aspects of policy generation, administration, and validation (cf. Section 3).

Thereby, the platform's architecture features a holistic approach to security policies based on current standards and the support of implicit and explicit security mechanisms in heterogeneous and distributed service infrastructures. Mobile devices can easily be connected to this infrastructure directly, when running the SicAri middleware, resp. through security-aware gateway components and specific protocols, in case the full platform does not execute on the mobile device due to resource limitations.

In spite of the platform features mentioned above, the platform is not intended to completely replace the existing information infrastructure nor its existing security mechanism, such as firewall, etc. The platform rather gives the opportunity of building distributed and security-centered applications on top of its service infrastructure, which can easily be extended or be plugged into existing infrastructures.

### *2.1 Platform Architecture*

Figure 1 (left-hand side) gives a high-level overview of the generic service architecture. On top of the middleware, there is a service and application layer. A locally authenticated user can directly interact with applications defined on this layer. The applications themselves make use of the basic and application-specific services of the platform's service layer. On the one hand, these services integrate external databases, legacy systems and applications; on the other hand, they provide basic security mechanisms, such as authentication or access control. The middleware layer is responsible for the secure and seamless integration and communication of applications and services, locally and remote.

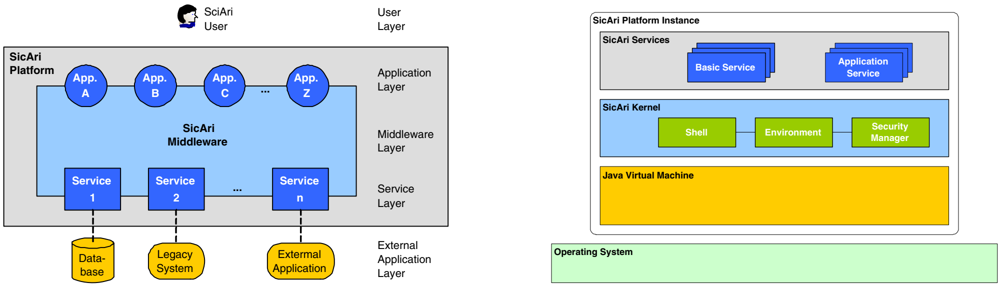


Fig. 1. High-level and layered architecture of the SicAri platform

The platform is based on the *SicAri kernel* (see right-hand side of Figure 1) which runs on top of a Java Virtual Machine and thereby attracts a broad spectrum of potential users and of potential devices, ranging from mobile devices and personal computers to scalable, distributed environments. The user application that runs on the platform accesses obligatory basic services and optional application services provided by the platform. The entirety of installed services thereby constitutes the characteristic and specific use case of the local platform instance. Thus, this platform is rather a collection of services which are flexibly loaded and configured within a common environment on top of a Java Virtual Machine, than a monolithic system designed as one static piece of software. Due to this architecture the platform offers the following advantages: minimalist design, modularity and reusability of services, extensibility, maintainability and intrinsic security features of the kernel.

### *2.2 Platform Components*

This sections briefly describes the basic components of the architecture.

**Environment.** The environment is a hierarchical name space for service object instances. Object instances can be registered, looked up, searched for, and removed. This mechanism for local service management and service discovery is extended by the transparent use of Web services for platform communication among distributed platforms. Services registered within this environment are implicitly encapsulated by a security proxy which enforces the current security policy on a search resp. access request to a service object.

**Shell.** The shell is a user interface for the environment. It allows administration of and access to the environment. Hence, it supports at least the afore mentioned operations *register*, *lookup*, *search*, and *remove* of object instances. Furthermore it allows navigation in the name space and invoking of Java methods. The SicAri shell compares to a UNIX Shell, where the environment stands for the file systems and services can be compared with files.

**Security Manager.** The security manager is responsible for policy enforcement. The Java programming language provides a standard security manager [8] which is accessible via an API. SicAri replaces this security manager with an own implementation.

**Service.** A service is a piece of software which fulfills a very specific task. It provides a small, well-defined programming interface. Typically there is no direct interaction between the user and a service. Every service is published as an object instance in the environment which allows access from other services and applications. Services are retrieved by means of the environment's search and lookup functionality. The service may be separated in a local access stub and one or more remote components which provide the functionality. Further, a service can integrate legacy applications and external data sources into the SicAri architecture, by means of a wrapper or proxy.

**SicAri kernel.** The SicAri kernel (or just *kernel*) consists of the Java implementations of the environment, the shell, and the SicAri security manager as defined above. Together they provide service bootstrapping and configuration, local service management (registration/searching), and a consistent security context by means of implicit access control.

**SicAri platform.** The platform consists of the kernel started on top of a Java Virtual Machine, a number of mandatory basic services, and optional application services. Any application can rely upon the availability of the basic services, as there are among others the authentication manager, the identity manager, the cryptographic key master, and the policy service.

**SicAri infrastructure.** The infrastructure is a compound of several platforms managed by the same security policy. These platforms may be distributed within the infrastructure.

**SicAri application.** An application is a software which fulfills a complex task. Since it usually interacts with the user, it provides both an interfaces for user-interaction and a programming interface. Applications make use of services in order to fulfill their tasks.

## 3 Holistic Approach to Security Policies

Policy-based control of networks and computer systems has the benefit that the controlling units of the system are kept decoupled from the management components and the rule base that governs the decisions. This enables the administrator to easily run, manage and change the system's behavior without having to modify the software or the controlled nodes. The system is controlled by policies that specify behavior rules which are interpreted by decision components and are asserted by enforcement components. Hence, if conditions change or new services or applications are added to the system one just adapts the policy rules. Using a central administration component the platform administrator does not have to deal with the multitude of different nodes in the system. This applies to network management issues, e. g. *Quality of Service (QoS)* or resource allocation, as well as to network and service security.

All security related tasks of the platform are controlled by a security policy. The platform covers various aspects of security policies, such as policy specification, policy patterns and policy compiler (see below), policy decision and enforcement, policy negotiation and provisioning, policy administration, and conflict resolution. Thereby, the security policy integration concept is based on manifold requirements with respect to policies, as for example:

- Control of all security related processes and tasks. Impossibility to bypass the policy enforcement component.
- Compatibility of the policy framework with the platform's plug-in approach. No need to change existing or upcoming services in order to enforce the platform's security policy. Transparency of policy control.
- Consideration of trade-off between expressiveness and complexity of the policy description language.
- Support for platform administrators during policy management.

It is another goal of the platform to bridge the gap between the informal specification of security policies (i. e., what the security administrator wants to enforce) and its corresponding machine-readable policy specification (i. e., what the system actually enforces).

### *3.1 Policy Architecture*

The policy architecture comprises the components of the policy framework and their interactions in order to guarantee that all security relevant processes in the platform are fulfilled according to the underlying security policy. Access control policies are based on the *Role-based Access Control (RBAC)* standard. The general concepts of RBAC are well-understood and extensively described in the literature, please refer to [6,12,18]. RBAC is assumed to be policy-neutral. This means that RBAC provides a flexible means to deal with arbitrary security policies. The policy integration concept of the SicAri platform requires the interaction of various components. Our implementation of RBAC uses XACML [11] as its specification language.

XACML thereby serves as ”glue” between a couple of policy components: Starting with the policy generation process which leads to XACML-based user-role assignments and XACML-based role-permission assignments, this XACML specification is used as basis for policy validation, afterwards. The validated policy then is distributed to resp. updated at all PDPs, and subsequently used for policy decisions.

Figure 2 depicts the component framework of the SicAri policy architecture. The remainder of this section gives a more detailed overview of all the components involved and their interactions with other components.

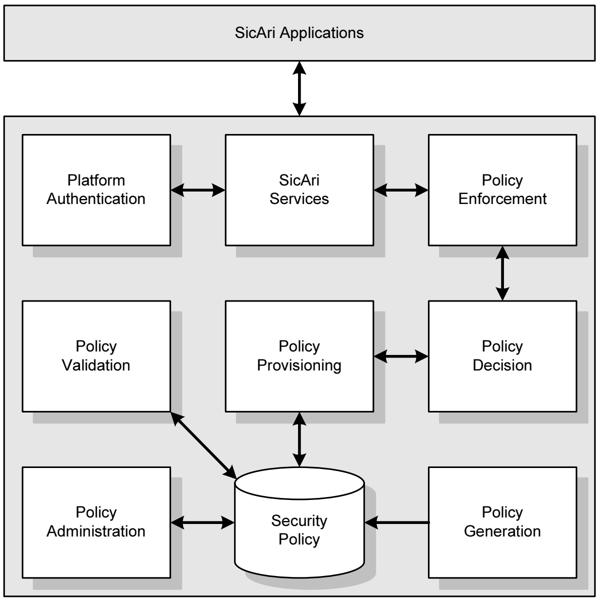


Fig. 2. Policy components of the SicAri platform

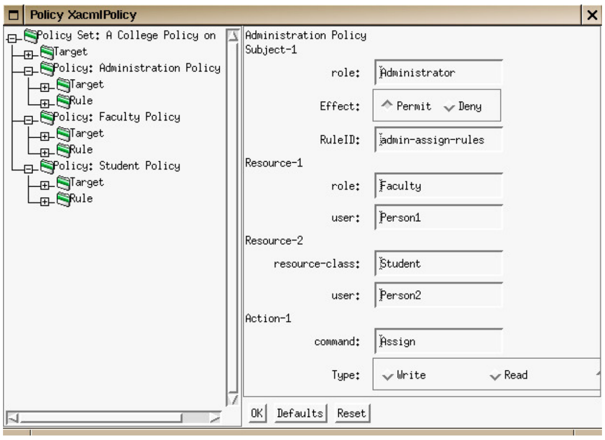
**Policy Generation.** We currently focus on generation of policies for access control to resources representing the required security properties. Such a policy is generated using *policy patterns* formalized in OWL [21] that allow to specify template policy archetypes for recurring security requirements. Policies are usually derived from security requirements of business or organizational goals. For example, the execution of the tasks ”credit request” and ”credit approval” with respect to a banking scenario require two different persons to give their affirmation to complete the task. This security requirement can be satisfied by introducing an assignment constraint in the model, for example, separation of duty.

Policy generation leads to a policy specification conforming with the RBAC profile of the XACML 2.0 standard [13].

**Policy Validation** The task of the policy validation component is to evaluate, whether a policy correctly implements given security goals. We extended the SH verification tool [14,15] to accept a subset of XACML as input and to translate it into transition patterns, which specify the behavior of *Asynchronous Product Automata (APA)*. APA are a class of general communicating automata and provide a means to model arbitrary distributed systems while transition patterns define the possible state transitions of the modeled system. Each policy rule is converted into such a transition pattern which then encodes the action that is controlled by that rule. This in turn results in an operational model of the policy system that can be executed in the SH verification tool. It allows to analyze the policy system's behavior, to simulate its potential information flow and to verify the wanted security goals. For that purpose the system's reachability graph is computed which spans all possible sequences of transition steps that are allowed by the given policy.

We have chosen to support a subset of XACML that comprises the most important elements and attributes of the language. One first goal was to reach the expressiveness that allows to handle one well-known XACML example which has been validated in the literature before [2,7]. Some concepts of XACML like obligations and rule combining algorithms are not yet supported.

**Policy Administration.** Even if the ability of automated security policy generation is provided by the platform, there may be the need of fine granular policy administration, e. g. a new user needs to be added, or the permissions of a user or role need to be changed. Therefore, we provide an administration API based on the RBAC standard and a corresponding graphical user interface (cf. Figure 3).



Administration Policy
Subject-1
role: Administrator
Effect: ● Permit ○ Deny
RuleID: admin-assign-rules
Resource-1
role: Faculty
user: Person1
Resource-2
resource-class: Student
user: Person2
Action-1
command: Assign
Type: ○ Write ○ Read

Fig. 3. Policy administration

**Policy Provisioning.** The components for policy generation, validation, and administration mutually share access to the global security policy database containing the current XACML-based policy specification. This whole policy specification resp. changes of policy subsets are subsequently distributed by policy provisioning components. The component covers distribution of policies, policy updates, as well as transport of policy decision requests and responses. Policy provisioning is outlined in more detail in the next section.

**Policy Decision.** The policy decision component uses an extended version of SUN's reference implementation of an XACML evaluator [19]. It comprises of library classes that can be used in building a *Policy Enforcement Point (PEP)* or a *Policy Decision Point (PDP)*. Since we use the RBAC profile of the XACML 2.0 standard to specify our policies, some modifications were necessary in order to evaluate these XACML policies.

As described above, the policies in form of an RBAC model are stored into three different categories, namely: *Role Policy Set (RPS)*, *Permission Policy Set (PPS)* and *Role Authorization (RA)*. The RPS and the PPS contain the roles definition and their corresponding permissions, respectively. Therefore, each RPS has a reference to the corresponding PPS. However, the implementation of SunXACML version 1.2 does not support references in policy. Therefore, we have extended this implementation accordingly to actually support policy references.

**Policy Enforcement.** The policy enforcement component assures that all security relevant tasks can only be fulfilled if they are in accordance with the underlying security policy. The policy enforcement component detects security relevant tasks, consults the policy decision component in order to decide upon a task, and enforces the policy decisions, i. e. allows a platform entity to access a platform resource or not.

**SicAri Services.** Interaction between applications and services and between one service and another as well as access of local resources within services is implicitly controlled by the policy enforcement component. Except the situation that a service wants to explicitly request the policy decision component, policy processing is done transparently during service execution. That is, SicAri services do not have to be aware of the existence of a security policy. As consequence, there is no need to modify or adapt existing or upcoming services to be compatible with the policy integration concept. The only thing that needs to be done by a security administrator is to configure resp. re-generate the security policy according to the security requirements in the context of new services integrated into the service infrastructure.

**Platform Authentication.** Finally and as another precondition for policy enforcement e. g. by means of access authorization, every acting entity in the service infrastructure has to be successfully authenticated. Thus, several authentication modules are provided locally on a platform instance to allow different user login procedures according to the specific use case and characteristic of the local platform instance.

## 4 Policy Distribution with COPS and XACML

This sections takes on the policy distribution issue from the introduction. The protocol framework for *Policy Based Network Management (PBN)* which has been defined by the *IETF Resource Allocation Protocol (RAP)* work group offers a good solution to those questions.

### *4.1 Policy Distribution with COPS*

The core of the RAP framework is the COPS [5] protocol. It provides a means to communicate policies and policy decisions in a distributed system. The main characteristics are

(i) the logical and architectural separation of policy enforcement and policy decision components, (ii) a client/server model of PEP and PDP, (iii) reliable transport of messages between PEP and PDP via TCP, (iv) a flexible and extensible framework through self-identifying objects that allow to define arbitrary protocol payload, and (v) a stateful communication between PEP and PDP which share request/decision states that allow the PDP to asynchronously update decisions and configuration information at the PEP.

COPS is designed to be used in two basic scenarios – outsourcing and configuration. In the *configuration scenario* a *local Policy Decision Point (LPDP)* is available and in the *outsourcing scenario* there is none. In the first case the PEP asks the LPDP for local policy decisions and in the latter case the PEP delegates all policy decisions to the remote PDP. Since the configuration scenario has already been implemented in SicAri its concept is described in more detail in the next paragraph. Section 4.2 explains how both scenarios integrate into the platform architecture.

#### COPS in Configuration Mode

In configuration mode the PEP requisitions a whole configuration for a component. Because COPS is policy independent the configured component can be such different things as router hardware or Web services.

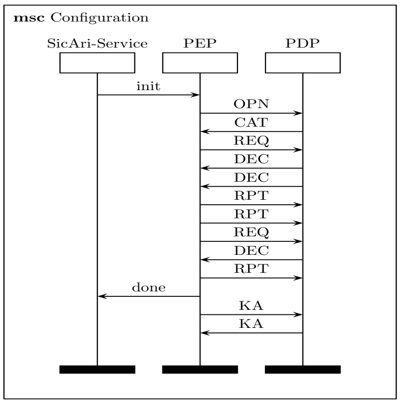


Fig. 4. Configuration request

Figure 4 shows the schematic sequence of the COPS configuration procedure in form of a Message Sequence Chart (MSC). When a service is started for the first time it contacts the PEP. The PEP sends a client open message (`OPN`) to the corresponding PDP. This message contains a unique ID that identifies the PEP to the PDP and it also contains a client specific information (`ClientSI`) object. This object is necessary to enable the PDP to relay the `OPN` message to a PDP module that can handle the requests for the incoming type of policy.

When the PDP is capable to serve the client type it answers with a client accept (`CAT`) message and expects incoming requests. In the configuration scenario the PEP sends one or more request messages (`REQ`) that contain context objects which identify the message as configuration requests. The request messages also comprise ClientSI objects that carry client specific information on the requested configuration data. Each configuration request may be answered with a single decision message (`DEC`) or a stream thereof. On reception and successful installation of the configuration data the PEP acknowledges this to the PDP with a report state message (`RPT`) for each of the `DEC` messages. When the PEP finally has received all configuration data from the PDP it signals the installation back to the SicAri service which now can rely on the LPDP to decide access requests.

From now on the PDP proactively keeps the policy at the PEP side up to date. Whenever a change to the master policy at the PDP side is made, it passes it on to all PEPs that make use of this policy. For that purpose both parties regularly exchange keep alive (`KA`) messages to assure that the PEP always uses a policy that is up to date.

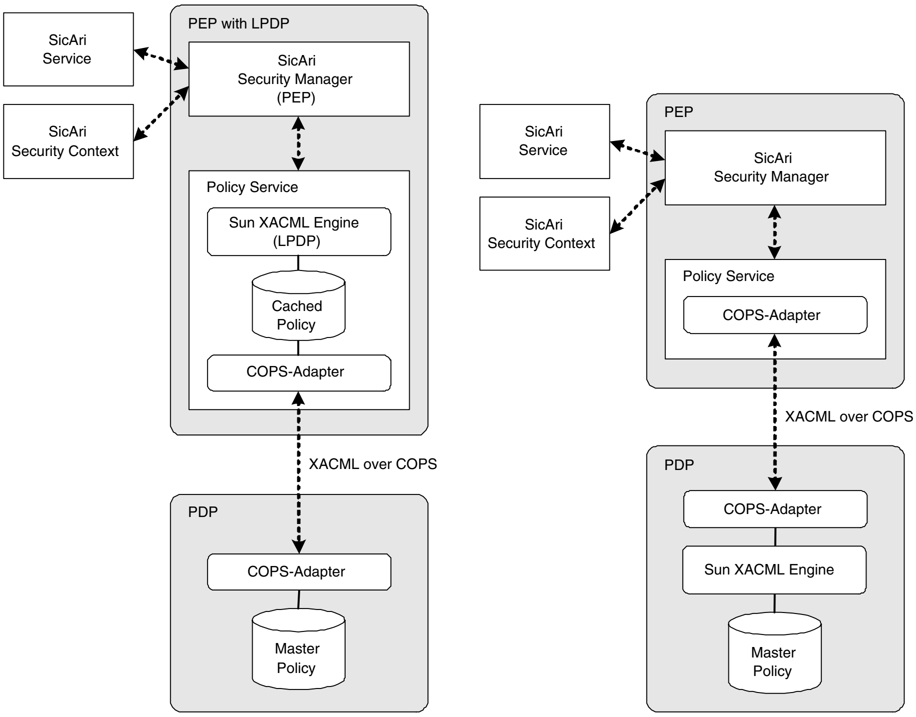
### *4.2 Platform Integration*

This section describes how the two policy distribution approaches can be integrated into the platform architecture.

In contrast to the COPS specification, the definition of PEP and LPDP in SicAri are slightly different: Whereas the PEP in COPS is defined as a local client component communicating with a global PDP, in the COPS policy configuration scenario this component corresponds best to the LPDP, as defined in SicAri (cf. Figure 4 vs. Figure 5 (a)).

#### COPS Policy Configuration

The main characteristics of this scenario are the local PEP and LPDP (cf. Figure 5 (a)). The PEP interacts with the local policy service, which mainly consists of the following components: LPDP, cached policy, and COPS adapter. The LPDP is responsible for making policy decisions based on the input from the SicAri security manager (PEP) and a locally cached version of the master security policy. The LPDP is realized by an extended version of Sun's XACML reference implementation (see below). The PEP uses the Java-API of Sun's XACML engine in order to communicate with the LPDP. A (potentially remote) policy provisioning component provides a copy of the latest master policy to the LPDP using the COPS protocol.



(a) Configuration Scenario

(b) Outsourcing Scenario

Fig. 5. Policy provisioning

### COPS Policy Outsourcing

The main characteristic of the second scenario is that a local PEP delegates all policy decisions to a remote PDP. This scenario is not yet implemented.

The policy service mainly consists of a COPS adapter which transforms the policy decision request of the PEP into an XACML policy request. The COPS adapter sends this request to the remote PDP which is responsible for providing the policy decision based on the master security policy. The XACML policy decision response is sent back from the remote PDP via COPS to the local PEP which enforces the policy decision.

## *4.3 XACML over COPS*

We extended the COPS framework with an XACML client type. All COPS messages start with a common header that determines the message type and the payload type. Figure 6 shows the schema of this header whose relevant fields are described below.

<table>
<tr><th colspan="2">0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>Version</td><td>Flags</td><td>Op Code</td><td colspan="2">Client Type</td></tr>
<tr><td colspan="5">Message Length</td></tr>
</table>

Fig. 6. Common COPS header

The `Op Code` indicates the type of the message, e. g. `REQ` or `KA`. The `Client Type` field provides a code that uniquely identifies the payload carried in the message. For example client-type number 1 is a published *Internet Assigned Numbers Authority* (IANA) number assigned to RSVP policy data [9].

Each COPS message may consist of different COPS objects. The message content is wrapped with the help of 16 different predefined COPS objects. Some of these objects provide fields to carry client-type specific data.

The most important object is the afore mentioned `ClientSI` object that has variable length and transports the client-type data. Figure 7 depicts the generic COPS object structure. Depending on the type of COPS message that is signaled zero, one or more COPS objects may follow the COPS header.

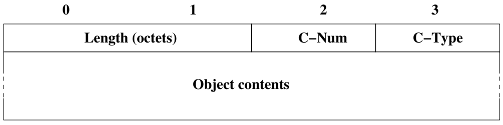


Fig. 7. Generic COPS object

The `C-Num` and `C-Type` fields determine the class and the characteristics of the object. For `ClientSI` objects the `C-Num` field is 9 and the variable length field for the object content carries the policy data. This data has to be processed by special COPS modules that can interpret the corresponding client type specific information. Our implementation bases on an open source implementation of the COPS protocol from the University of Waterloo [1,3]. We extended their PBN code with several classes to multiplex incoming COPS messages at the PDP to modules which handle client-specific content like XACML.

### *4.4 XACML Client Type for the COPS Protocol*

As the next step a concept to extend the COPS protocol to transport XACML polices as payload has been developed. For any extension to the COPS protocol one has to take the peculiarity of the target policy language into account. The structure of the client-type specific objects and the protocol extensions should be specified in a supplementary document that defines how the PEP and the PDP interpret and handle the policy specific payload.

Any XACML policy document is structured according to the respectively effective XACML schema. The XACML data flow model defines that a *Policy Administration Point (PAP)* provides the PDP with XACML documents that contain sets of `policy` and `policyset` elements. It is specified in the XACML schema that `policy` and `policyset` elements can be nested inside a `policyset`. The distribution of XACML client-type data in the configuration mode will base on this tree structure of XACML documents. Any leaf and any node of such a document will be encapsulated in a single COPS object and send alone or in a group in a `decision` message from the PDP to PEP. Our concept considers the `policy` and `policyset` elements as such leaf and node objects since they define logical building blocks of a policy. The PEP acknowledges any such COPS message with XACML content with a COPS `report` message. On COPS' PEP side the `policy` and `policyset` building blocks are assembled back into a copy of the XACML master policy which is passed on to the LPDP as defined in SicAri.

The XML structure provides another advantage with respect to the proactive update and delete mechanism of COPS PDPs. A PDP can address any `policy` element in the XACML document using XPATH. Any administrative task modifying the master policy triggers a COPS message for the corresponding `policy` and `policyset` element that has been changed. This COPS message will transport one or more COPS `decision` objects containing `replacement data` that uniquely identifies the processed policy elements using XPATH objects. This way the PDP can generate fine-granular updates at the XML element level. The COPS protocol assures that both, PEP and PDP, always work on the same XACML document. Any policy document is definitely identified by the TCP connection between PEP and PDP together with the client handle that the PEP uniquely assigns to each request that it sends out.

## 5 Related Work

In [17], Ponnappan *et al.* describe a policy based QoS management system for IntServ/DiffServ networks. This design uses COPS for interfacing with the network devices and CORBA as middleware for component interaction.

An approach presented in [20] by Toktar *et al.* proposes an XACML-based framework for distributing and enforcing access control policies to RSVP-aware application servers. Access control policies are represented in an extension of XACML (based on Sun XACML) which is an alternative to the IETF Policy Core Information Model (PCIM) [10] based approach. The authors use COPS in outsorcing mode to distribute policy requests and decisions between the policy server and the RSVP server that is responsible to enforce the QoS measures.

In his dissertation [16], K. Phanse proposes a management framework for policy based ad hoc network management. He builds on a distributed, hybrid architecture that combines the outsourcing and provisioning models of COPS and COPS-PR to provide an efficient and flexible solution for policy distribution in wireless ad hoc networks. To translate the policy specification into device-specific configuration, the management framework must be aware of the various resources available in the system. Policy provisioning occurs after policies are distributed, and consists of installing and implementing the policies using device specific mechanisms.

Since COPS seems to be the only open service for policy distribution of notable propagation we find it hard to compare our approach to others. While it is difficult to make a quantitative statement on the efficiency to transport XACML policies via COPS it is easier to give a qualitative predication. COPS realizes some design aspects that improve efficient distribution and maintenance of distributed policies. After provisioning the initial policy in configuration mode to the LPDPs the central PDP keeps them up-to-date with unsolicited decision messages whenever some part of the policy changes. Since COPS allows to transport policy parts of arbitrary size it is up to the developer of the payload extension to optimize the communication overhead. COPS only demands that the transported pieces are uniquely addressable. This perfectly fits to XACML because XPATH and unique element identifiers allow to address and transport only those pieces of the master policy that have really changed. Furthermore, COPS in configuration mode promises to economically use network resources because access control policies are not very likely to be changed frequently. This enables the administrator to choose a higher value for the KA timeout thus reducing the communication overhead when there are no updates. The only drawback lurks in the fact that XACML is an XML-based language which are inherently wordy.

## 6 Outlook

It is planned to implement the mechanism to encapsulate XACML payload in COPS messages as described in Section 4.4. The open source JDOM (`http://www.jdom.org/`) package for parsing and representing XACML documents as objects seems a viable basis to build a solution upon.

We will furthermore develop a concept to use the policy administration, policy validation, and policy provisioning mechanisms implemented in the platform, to manage other policy domains such as network security policies (e. g. to configure external PEPs such as firewalls).

An additional research aspect with respect to a holistic policy approach in distributed environments will be policy negotiation in case services from two different security domains, enforcing security upon two different security policies, have to interact with each other.

## Acknowledgement

This paper was written while the authors were working within SicAri, a project funded by the German Ministry of Education and Research

## References

[1] Boutaba, R., A. Polyrakis and A. Casani, *Active Networks as a Developing and Testing Environment for Networks Protocols*, Annals of Telecommunications **59**, 2004, pp. 495–514.

[2] Bryans, J., *Reasoning about XACML policies using CSP*, in: *SWS'05: Proceedings of the 2005 workshop on Secure Web Services* (2005), pp. 28–35.

[3] Casani, A., *Implementation of a Policy Based Network Framework using Metapolicies* (2001).

[4] Consortium, S., *SicAri – A security architecture and its tools for ubiquitous Internet usage* (2003), `http://www.sicari.de/`.

[5] Durham, D., J. Boyle, R. Cohen, S. Herzog, R. Rajan and A. Sastry, *The COPS (Common Open Policy Service) Protocol*, RFC 2748 (Proposed Standard) (2000), updated by RFC 4261.

[6] Ferraiolo, D. F., D. R. Kuhn and R. Chandramouli, "Role-Based Access Control," Computer Security Series, Artech House, Boston, 2003.

[7] Fisler, K., S. Krishnamurthi, L. A. Meyerovich and M. C. Tschantz, *Verification and change-impact analysis of access-control policies*, in: *ICSE'05: Proceedings of the 27th International Conference on Software engineering* (2005), pp. 196–205.

[8] Gong, L., "Java$^{TM}$ 2 Platform Security Architecture, Version 1.2," Sun Microsystems Inc., 2002.

[9] Herzog, S., J. Boyle, R. Cohen, D. Durham, R. Rajan and A. Sastry, *COPS usage for RSVP*, RFC 2749 (Proposed Standard) (2000).

[10] Moore, B., E. Ellesson, J. Strassner and A. Westerinen, *Policy Core Information Model, Version 1* (2001).

[11] Moses, T., *eXtensible Access Control Markup Language (XACML), Version 2.0*, Technical report, OASIS Standard (2005).

[12] National Institute of Standards and Technology (NIST), *Role-Based Access Control*, http://csrc.nist.gov/rbac/.

[13] OASIS Open, *Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML v2.0* (2005), http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.

[14] Ochsenschläger, P., J. Repp and R. Rieke, *The SH-Verification Tool*, in: *Proc. 13th International Florida Artificial Intelligence Research Society Conference (FLAIRS'00)* (2000), pp. 18–22.

[15] Ochsenschläger, P., J. Repp, R. Rieke and U. Nitsche, *The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems*, Formal Aspects of Computing, The International Journal of Formal Methods **11** (1999), pp. 1–24.

[16] Phanse, K. S., *Policy-Based Quality of Service Management in Wireless Ad Hoc Networks*, Dissertation, Virginia Polytechnic Institute and State University (2003).

[17] Ponnappan, A., L. Yang, R. Pillai and P. Braun, *A Policy Based QoS Management System for the IntServ/DiffServ Based Internet*, in: *Proc. of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'02)* (2002), p. 159ff.

[18] Sandhu, R., *Role activation hierarchies*, in: *Proceedings of the 3rd ACM workshop on Role-based access control* (1998).

[19] Sun Microsystems, Inc., *Sun's XACML Implementation, Version 1.2.* URL http://sunxacml.sourceforge.net/

[20] Toktar, E., E. Jamhour and C. Maziero, *RSVP Policy Control using XACML*, in: *Proc. of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)* (2004), p. 87ff.

[21] World Wide Web Consortium, *OWL Web Ontology Language – Overview* (2004), http://www.w3.org/TR/owl-features/.

# P13

## PREDICTIVE SECURITY ANALYSIS FOR EVENT-DRIVEN PROCESSES

| | |
|---|---|
| **Title** | Predictive Security Analysis for Event-Driven Processes |
| **Authors** | Roland Rieke and Zaharina Stoynova |
| **Publication** | In Igor Kotenko and Victor Skormin, editors, *Computer Network Security – 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2010 St. Petersburg, Russia, September 2010, Proceedings*, volume 6258 of *Lecture Notes in Computer Science*, pages 321–328, 2010. |
| **ISBN/ISSN** | ISBN 978-3-642-14705-0 |
| **DOI** | `http://dx.doi.org/10.1007/978-3-642-14706-7_25` |
| **Status** | Published |
| **Publisher** | Springer Berlin / Heidelberg |
| **Publication Type** | Conference Proceedings (LNCS, Vol. 6258) |
| **Copyright** | 2010, Springer |
| **Contribution of Roland Rieke** | Main Author, editor, and presenter at the MMM-ACNS conference 2010.<br>Specific contributions are: (a) design of the architecture for security event processing and predictive security monitoring, and (b) the operational model for security event prediction. |

Table 18: Fact Sheet Publication *P13*

Publication P13 [Rieke & Stoynova, 2010] addresses the following research question:

RQ9A *How can operational models reflect the state of observed systems and thus capture abstractions of runtime behaviour?*

The main constraint of current systems is the restriction of Security Information and Event Management (SIEM) [Nicolett & Kavanagh, 2009] to network infrastructure, and the inability to interpret events and incidents from other layers such as the service view, or the business impact view, or on a viewpoint of the service itself. This paper presents an approach for predictive security analysis in a business process execution environment. It is based on operational process models and leverages process and threat analysis and simulation techniques in order to be able to dynamically relate events from different processes and architectural layers and evaluate them with respect to security requirements. Based on this, a blueprint of an architecture is presented which can provide decision support by performing dynamic simulation and analysis while considering real-time process changes. It allows for the identification of close-future security-threatening process states and will output a predictive alert for the corresponding violation.

With kind permission of Springer Science+Business Media.
This is an author-created version of: Computer Network Security; Lecture Notes in Computer Science Volume 6258, 2010, pp 321-328; Predictive Security Analysis for Event-Driven Processes; Roland Rieke, Zaharina Stoynova; © Springer-Verlag Berlin Heidelberg 2010; DOI: 10.1007/978-3-642-14706-7_25; Print ISBN: 978-3-642-14705-0; Online ISBN: 978-3-642-14706-7.
The original publication is available at www.springerlink.com.
http://link.springer.com/chapter/10.1007%2F978-3-642-14706-7_25

# Predictive Security Analysis for Event-driven Processes

Roland Rieke and Zaharina Stoynova

Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany
{roland.rieke,zaharina.stoynova}@sit.fraunhofer.de

**Abstract.** This paper presents an approach for predictive security analysis in a business process execution environment. It is based on operational formal models and leverages process and threat analysis and simulation techniques in order to be able to dynamically relate events from different processes and architectural layers and evaluate them with respect to security requirements. Based on this, we present a blueprint of an architecture which can provide decision support by performing dynamic simulation and analysis while considering real-time process changes. It allows for the identification of close-future security-threatening process states and will output a predictive alert for the corresponding violation.

**Keywords:** predictive security analysis, analysis of business process behaviour, security modelling and simulation, complex event processing

## 1 Introduction

With the increased adoption of service oriented infrastructures and architectures, organisations are starting to face the need for an accurate management of cross-process and cross-layer security information and events. The main constraint of current systems is the restriction of Security Information and Event Management (SIEM) [8] to network infrastructure, and the inability to interpret events and incidents from other layers such as the service view, or the business impact view, or on a viewpoint of the service itself. Conversely, specific service or process oriented security mechanisms are usually not aware of attacks that exploit complex interrelations between events on different layers such as physical events (e.g. access to buildings), application level events (e.g. financial transactions), business application monitoring, events in service oriented architectures or events on interfaces to cloud computing applications. Nevertheless, next generation systems should be able to interpret such security-related events with respect to specific security properties required in different processes. On the base of these events, the system should be able to analyse upcoming security threats and violations in order to trigger remediation actions even before the occurrence of possible security incidences.

In this paper we propose to combine process models with security policies and a security model in order to identify potential cross-cutting security issues. We furthermore suggest a blueprint of an architecture for predictive security analysis that leverages process and threat analysis and simulation techniques in order to be able to dynamically relate events from different execution levels, define specific level abstractions and evaluate them with respect to security issues.

## 2 Related Work

Our work combines aspects of process monitoring, simulation, and analysis. Some of the most relevant contributions from these broad areas are reviewed below.

**Business Activity Monitoring (BAM).** The goal of BAM applications, as defined by Gartner Inc., is to process events, which are generated from multiple application systems, enterprise service busses or other inter-enterprise sources in real time in order to identify critical business key performance indicators and get a better insight into the business activities and thereby improve the effectiveness of business operations [6]. Recently, runtime monitoring of concurrent distributed systems based on LTL, state-charts, and related formalisms has also received a lot of attention [5, 3]. However these works are mainly focused on error detection, e.g. concurrency related bugs. In the context of BAM applications, in addition to these features we propose a *close-future* security analysis which provides information about possible security risks and threats reinforcing the security-related decision support system components.

**Complex Event Processing (CEP)**. CEP provides a powerful analytic computing engine for BAM applications which monitor raw events as well as the real-time decisions made by event scenarios. David Luckham [4] provides us with a framework for thinking about complex events and for designing systems that use such events. A framework for detecting complex event patterns can be found e.g. in [10]. However such frameworks concentrate on detecting events important for statistical aspects, redesign and commercial optimisation of the business process. Here we want to broaden the scope of the analysed event types by introducing *complex security events* in the CEP alphabet.

**Simulation.** Different categories of tools that are applicable for simulation of event-driven processes including process modelling tools based on different semi-formal or formal methods such as Petri Nets [2] or Event-driven Process Chains (EPC) [1]. Some process managements tools, such as FileNet [7] offer a simulation tool to support the design phase. Also some general purpose simulation tools such as CPNTools [11] were proven to be suitable for simulating business processes. However, independently from the tools and methods used, such simulation tools concentrate on statistical aspects, redesign and commercial optimization of the business process. On the contrary, we propose an approach for *on-the-fly* intensive dynamic simulation and analysis considering the current process state and the event information combined with the corresponding steps in the process model.

**Security Information Management (SIM).** SIM systems generally represent a centralized server acting as a ”security console”, sending it information about security-related events, which displays reports, charts, and graphs of that information, often in real time. Commercial SIEM products include

Cisco Security Monitoring Analysis and Response System (`http://www.cisco.com/en/US/products/ps6241/index.html`), EventTracker by Prism Microsystems (`http://www.prismmicrosys.com/EventTrackerSIEM/index.php`), SenSage (`http://www.sensage.com/products/sensage-40.php`) and others. All these products monitor the low-level events (such as network events) and perform event correlation only on the base of event patterns and rules. Our approach additionally considers the business process level events combined with the current process state and business process information provided by a process specification.

## 3 Blueprint of Architecture for Security Event Processing and Predictive Security Monitoring

In this section we introduce our approach for security evaluation of event-driven processes. Figure 1 depicts the core components which we consider necessary in order to be able to perform a security event processing and monitoring analysis in the context of a running event-driven business process.

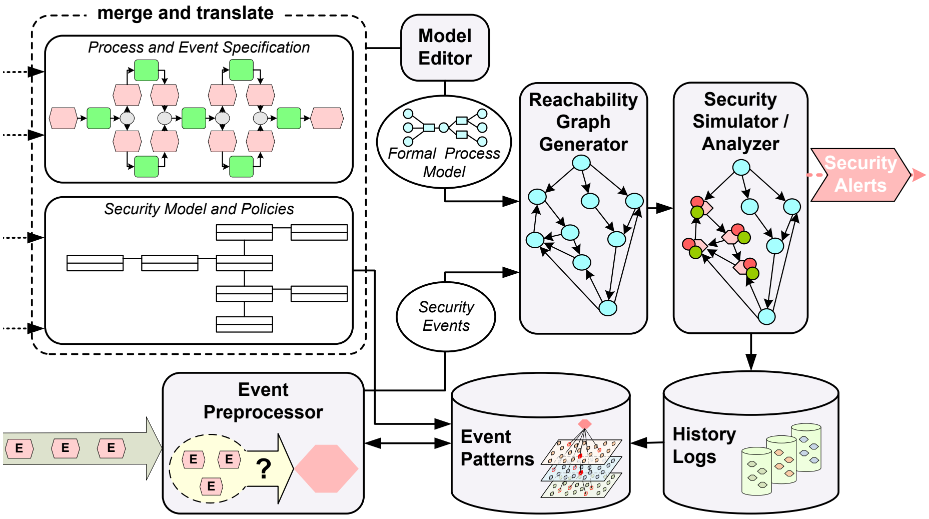


**Fig. 1.** Predictive Security Analyser

The input elements which we need comprise, (1) a *process model* given in a notation such as EPC, BPEL, YAWL or BPMN that contains a specification of the events which can be triggered during runtime, (2) *security policies* which contain information about the relations between the users involved in the process, their roles and the relations between the roles and resources deployed by the process, (3) a *security model* that should provide information about the process's predefined security requirements which will be used to construct the security events patterns, and, (4) *real-time events* which will be triggered during runtime.

**Model Editor.** In order to analyse the system behaviour with tool support, an appropriate formal representation has to be chosen because semi-formal languages such as BPMN allow to create models with semantic errors [2]. In our approach, we use an operational finite state model based on *Asynchronous Product Automata (APA)* [9]. An APA consists of a family of so called *elementary automata* communicating by common components of their state (shared memory). The process model, the organisational model and the security model should be imported and merged in a high-level model of the process and then this model is translated into an APA, which will enable the computation of the possible system behaviour. In general, we could also use other descriptions of processes with unambiguous formal semantics here such as the approaches in [2] for BPMN or [1] for EPC that allow for computation of possible system's behaviour.

**Reachability Graph Generator.** Formally, the behaviour of an APA can be given by a reachability graph which represents all possible coherent sequences of state transitions starting with the initial state. In the context of on-the-fly security analysis the reachability graph will represent the path given by the already triggered events, forwarded by the Event Preprocessor. The computation will be automatically paused each time when the current state (according to the triggered events) of the process is reached. In the context of predictive simulation analysis the Reachability Graph Generator computes all possible near-future paths according to the given process specification, (e.g. sequences of at most 2-3 plausible events). This will allow exhaustive analysis of all near-future states to be performed in order to compute whether there exist possible security-threatening states of the process which can compromise the process security and match some of the event patterns saved in the Event Patterns database.

**Security Simulator/Analyser.** During the computation of the graph this component will check for each state, whether the specified security properties are fulfilled and trigger security alarms when possible security violations are found. Furthermore, it is possible to detect new security violations that were not predicted by the available security patterns. In order to include them in the analysis of future process instances, they will be logged in the History Logs database and then they will be transformed into security event patterns and saved in the Event Patterns database. The simulator will also enable security analysis by performing intensive simulation which inspects the behaviour of complex/parallel processes under given hypotheses (*what-if analysis*) concerning changes in the organisational model/security policies or the process model.

**Security Event Patterns.** These patterns which are relevant for the corresponding process are kept in the Event Patterns database and they should be extracted from the provided security model. In order to be able to reason about potential security problems, based on real life events, specific abstractions are included in this extraction process so that the abstraction levels for the various types of security-related events can be interrelated. Solutions for these kind of security analysis are already available but usually limited to a narrow field of application such as IDS where e.g. the detection of a number of abnormal connections could lead to a “worm detection” alarm. We propose a generic approach leveraging these ideas and incorporating other types of security related events.

**Event Preprocessor.** In the context of on-the-fly security analysis the Event Preprocessor is responsible for receiving the real-life events triggered during runtime, matching them against the available security event patterns and forwarding them to the Reachability Graph Generator. During predictive security analysis the Event Preprocessor will generate all possible events according to the process specification and will match them against the event patterns. Then it will forward them to the Reachability Graph Generator in oder to enable the computation of the process graph.

**History Logs.** In the History Logs database newly detected security-violating sequences of events will be logged. These will be used to create new security event patterns.

## 4 An Application Scenario

For illustrating how our architecture components, described in the previous section, collaborate we will refer to a common example scenario for online credit application.

### 4.1 Process Model

In an EPC graph events are represented as hexagons and functions that describe state transitions are represented as rounded rectangles. Now consider the online credit application process expressed in EPC notation in Fig. 2. The process starts when an applicant submits an application form. Upon receiving a new application form a credit clerk performs checks in order to validate the applicant’s income and other relevant information. Depending on the requested loan amount different checks are performed. Then the validated application passed on to a manager to decide whether to accept or reject it. In both cases the applicant is notified of the decision and the process ends.

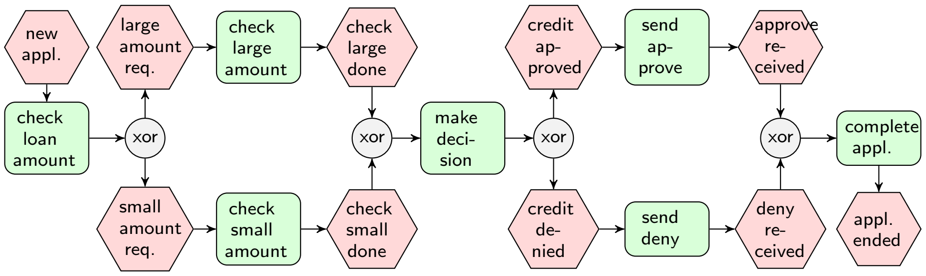
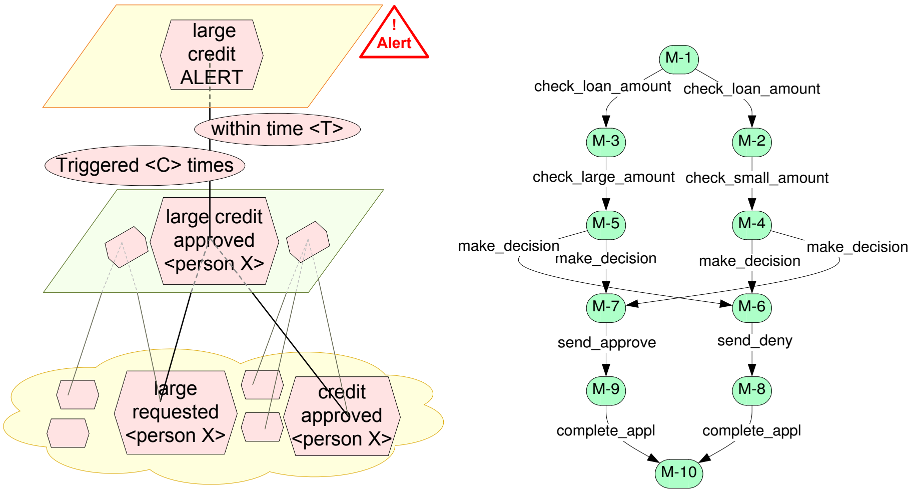


**Fig. 2.** Business Process Model

### 4.2 Predicting Security Events

In our example scenario we consider the security event "*large credit ALERT*" which is raised when too many large credits are approved for one customer (see Fig. 3(a)). This is an example of an event abstraction or complex event generated by a certain sequence of simple events, triggered in the process. Such complex events are generated by CEP engines whenever certain predefined sequences of events have been triggered.

Additionally, we apply such complex event patterns in a predictive way. This means that whenever an event pattern is *probably* going to match by taking into account a current partial match and a possible continuation of the current state, these abstractions can be generated prior to the real-time triggering of the simple events. In our example we generate an abstraction of the atomic events "*large amount requested*" and "*credit approved*" triggered by the same customer, namely the complex event "*large credit approved*". Then if this complex event is generated e.g. two times within a certain time and according to security regulations only two large credits can be given to one customer we can generate the alert "*large credit ALERT*" in the upper abstraction level prior to the next approval in order to ensure that the security regulations will not be overseen by taking the credit decision.



(a) On-the-fly security violations (b) Reachability graph of business process

**Fig. 3.** Predict near future security violations

### 4.3 Operational Model for Security Event Prediction

A computation of the possible system behaviour of a formal APA model of the business process in Fig. 2 results in the reachability-graph depicted in Fig. 3(b).

The state $M$-3 e.g. represents the situation where an event of type "*large amount requested*" is available and can be processed by the action "*check_large_amount*" which in turn will trigger an event of type "*check large done*". After this, the process is in state $M$-5, where the action "*make_decision*" can be executed and lead to one of the two possible followup states $M$-6 or $M$-7. $M$-7 is reached iff the decision results in an event "*credit approved*".

From this we now conclude that a predictive alert "*large credit ALERT*" can be generated if, (1) the system is in a state where the number of large credits allowed for one customer is exhausted, (2) an event "*large amount requested*" for the same customer is received, and, (3) an evaluation of possible continuations of the process's behaviour based on the operational model shows that an additional event of type "*large credit approved*" is possible within the forecast window.

The method described in this paper addresses security properties that can be stated as safety properties. Possible violations of these properties are identified by reachable states in the predicted system behaviour. Some examples of security related event types that can be analysed by the method given in this paper are:

*Confidentiality.* Consider an event sending a cleartext password. Predict that in one possible continuation of a process, an event about processing a cleartext password locally may lead to an event sending that password.

*Authenticity.* Consider the physical presentation of a token which is known to be unique such as a credit card or passport as parameter of two different events with very close time and very different location.

*Authorisation.* Consider two events with persons with the same biometric parameters in different locations at the same time.

*Integrity/Product counterfeiting.* Consider RFIDs being scanned in places where they are not expected.

*Integrity/Safety.* Consider two trains on the same railtrack. Predict that a specific constellation of switches leads to a crash in one possible continuation.

## 5 Conclusions and Further Work

In this paper we proposed a blueprint of an architecture for predictive security analysis of event-driven processes that enables exhaustive process analysis during runtime based on the triggered real-life events. Our approach is based on the specification of an operational finite state model of the process behaviour We have demonstrated how our methods can be applied in order to ensure certain security regulations in the process of online credit application and how we can construct event abstractions on different levels in order to detect current and near-future threats.

Currently our components are prototypically implemented without automated merging and translation mechanisms for the input models and specifications, automated event pattern extraction and new event pattern composition. We used the *SH verification tool* [9] to analyse an exemplary business process model for different concrete instantiations (numbers of clients, and time-horizon) of the model. In the future, we will further develop such techniques in order to automate the security analysis and simulation and extend the method to cover liveness properties.

Furthermore, alerts in today's monitoring systems by themselves bring little value in the process security management if they cannot be acted upon. Therefore, we have to provide additionally to the alerts alternative counter-measure scenarios that can be quantifiable evaluated thanks to simulation. In this way our analysis can be extended to provide feedback to the operators on feasibility and impacts of both attacks and counter-measures.

**Acknowledgments.** The work presented in this paper was developed in the context of the project Alliance Digital Product Flow (ADiWa) that is funded by the German Federal Ministry of Education and Research. Support code: 01IA08006F.

## References

1. Dijkman, R.M.: Diagnosing Differences Between Business Process Models. In: Dumas, M., Reichert, M., Shan, M.C. (eds.) BPM. Lecture Notes in Computer Science, vol. 5240, pp. 261–277. Springer (2008)
2. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. 50(12), 1281–1294 (2008)
3. Kazhamiakin, R., Pistore, M., Santuari, L.: Analysis of communication models in web service compositions. In: WWW'06: Proc. of the 15th international conference on World Wide Web. pp. 267–276. ACM, New York (2006)
4. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2002)
5. Massart, T., Meuter, C.: Efficient online monitoring of LTL properties for asynchronous distributed systems. Tech. rep., Université Libre de Bruxelles (2006)
6. McCoy, D.W.: Business Activity Monitoring: Calm Before the Storm. Gartner Research (2002)
7. Netjes, M., Reijers, H., Aalst, W.P.v.d.: Supporting the BPM life-cycle with FileNet. In: Proceedings of the Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'06), held in conjunction with the 18th Conference on Advanced Information Systems (CAiSE'06), Luxembourg, Luxembourg, EU. pp. 497–508. Namur University Press, Namur, Belgium, EU (2006)
8. Nicolett, M., Kavanagh, K.M.: Magic Quadrant for Security Information and Event Management. Gartner RAS Core Reasearch Note (May 2009)
9. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing, The International Journal of Formal Method 11, 1–24 (1999)
10. Pietzuch, P.R., Shand, B., Bacon, J.: A framework for event composition in distributed systems. In: Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware. pp. 62–82. Springer-Verlag New York, Inc., New York, NY, USA (2003)
11. Rozinat, A., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.J.: Workflow simulation for operational decision support. Data Knowl. Eng. 68(9), 834–850 (2009)

# P14

# MODEL-BASED SITUATIONAL SECURITY ANALYSIS

| | |
|---|---|
| **Title** | Model-based Situational Security Analysis |
| **Authors** | Jörn Eichler and Roland Rieke |
| **Publication** | In *Proceedings of the 6th International Workshop on Models@run.time at the ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (MODELS 2011), Wellington, New Zealand*, pages 25–36. 2011. |
| **ISBN/ISSN** | ISSN 1613-0073 |
| **URL** | `http://ceur-ws.org/Vol-794/paper_1.pdf` |
| **Status** | Published |
| **Publisher** | RWTH Aachen |
| **Publication Type** | CEUR Workshop Proceedings, Vol. 794 |
| **Copyright** | 2011, Papers' Authors |
| **Contribution of Roland Rieke** | Main Author, editor, and presenter at the 6th MRT workshop 2011.<br>Specific contributions are: (a) the process and event model, (b) the security requirements elicitation, (c) the formal model, and (d) the security reasoning during runtime operation. |

Table 19: Fact Sheet Publication *P14*

Publication P14 [Eichler & Rieke, 2011] addresses the following research questions:

RQ10 *How can security analysis at runtime exploit process models to identify current and close-future violations of security requirements?*

Security analysis is growing in complexity with the increase in functionality, connectivity, and dynamics of current electronic business processes. To tackle this complexity, the application of models in pre-operational phases is becoming standard practice. Runtime models are also increasingly applied to analyse and validate the actual security status of business process instances. This paper presents an approach to support not only model-based evaluation of the current security status of business process instances, but also to allow for decision support by analysing close-future process states. The approach is based on operational formal models derived from development-time process and security models. This paper exemplifies the approach utilising real world processes from the logistics domain and demonstrates the systematic development and application of runtime models for situational security analysis.

# Model-based Situational Security Analysis

Jörn Eichler and Roland Rieke

Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany
{joern.eichler,roland.rieke}@sit.fraunhofer.de

**Abstract.** Security analysis is growing in complexity with the increase in functionality, connectivity, and dynamics of current electronic business processes. To tackle this complexity, the application of models in pre-operational phases is becoming standard practice. Runtime models are also increasingly applied to analyze and validate the actual security status of business process instances. In this paper we present an approach to support not only model-based evaluation of the current security status of business process instances, but also to allow for decision support by analyzing close-future process states. Our approach is based on operational formal models derived from development-time process and security models. This paper exemplifies our approach utilizing real world processes from the logistics domain and demonstrates the systematic development and application of runtime models for situational security analysis.

**Keywords:** security requirements elicitation, predictive security analysis, analysis of business process behavior, security modeling and simulation, security monitoring

## 1 Introduction

Electronic business processes connect many systems and applications. This leads to an increasing complexity when analyzing distinctive properties of those business processes. Additionally, frequent changes to business process models are applied to address changing business needs. Current approaches apply changes to those models at runtime [4]. This situation challenges operators and participants in electronic business processes as the assessment of the status of business process instances at runtime becomes difficult. An example for these difficulties is the assessment whether instances of business processes violate security policies or might violate them in the near future.

Traditionally, approaches to security analysis of electronic business processes are executed at development-time. In this perspective, the analysis of possible violations of security policies is part of the requirements engineering process [13]. To cope with the growing complexity of the electronic business processes, the application of security models in the course of the requirements engineering process is becoming a common strategy [5]. Nevertheless, the requirements engineering process is generally limited to development-time.

**Contributions.** To support security analysis at runtime we utilize formal models based on development-time process and security models. On the basis of sound methods for the elicitation and modeling of security requirements provided in [7] and an architectural blueprint described in [18], we document in this paper our approach to analyze the security status of electronic business processes. The security analysis consumes events from the runtime environment, maps those events to security events and feeds them to our runtime model, an operational finite state model. This allows to match and synchronize the state of the real process with the state of the model. Annotations of security requirements to the states of the model can now be used to check for security violations and possibly generate alarms. These alarms are then in turn converted to events and sent to the running business process. Furthermore, a computation of possible close-future behavior, which is enabled by the model of the business process, is used to evaluate possible security critical states in the near future at runtime. This knowledge about possibly upcoming critical situations can be used to raise respective predictive alarms.

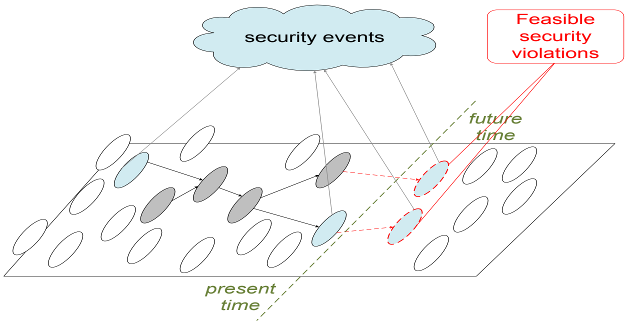


**Fig. 1.** Predict feasible security violations

In section 2 we provide an application scenario from the logistics domain and elicit security requirements. The formalization of the scenario is given in section 3. Section 4 analyzes the runtime operation and exemplifies generated security alerts. Section 5 reviews shortly related work to our approach. Concluding remarks and further research directions are given in section 6.

## 2 Application Scenario

In order to demonstrate what kind of security requirements we are able to consider and how our model-based runtime analysis is applied, we have chosen a small part of a "*Pickup*" target process which is analysed in the project Alliance Digital Product Flow (`http://www.adiwa.net/`).

### 2.1 Process and Event Model

The "Pickup" process is initiated when the truck driver is notified about new pickup orders. He accepts the received list of orders and the system calculates a route plan based on the addresses. When the driver arrives at a pickup address, he checks visually the packages for deviations with regard to the description in the order. In case of deviations he consults with the sender whether this package is to be transported. If the truck driver accepts the new package, the package description in the list of orders is updated accordingly. For each accepted package the system receives a confirmation that it has been loaded. The system links each loaded package and its transporting truck using the corresponding radio-frequency identification (RFID) tag identifiers. An Event-driven Process Chain (EPC) flowchart of the considered subprocess is depicted in Figure 2. Rectangles with rounded corners denote actions and chamfered rectangles denote events.

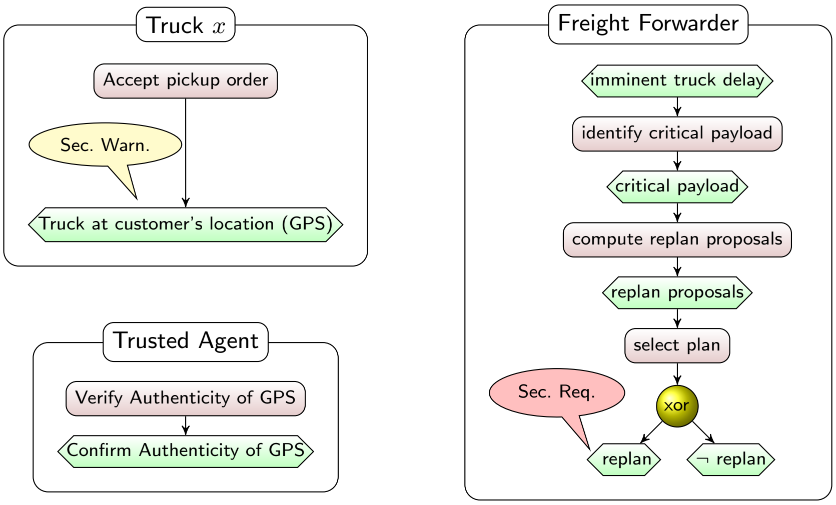


**Fig. 2.** Model of a part of the Pickup Process (EPC notation)

As an example of a security threatening misuse case, we consider a situation where the system performs a rescheduling because of a delay of one or more trucks on the basis of not confirmed Global Positioning System (GPS) locations. In this case there is a possibility for an attacker to send false GPS data to the system, which may result in ineffective rescheduling and possible time loss in completing the orders.

### 2.2 Security Requirements Elicitation

In order to derive the security requirements in the given scenario, we follow the scheme described in [7]. We assume that the functional dependencies between the actions in our scenario are given by Fig. 3.

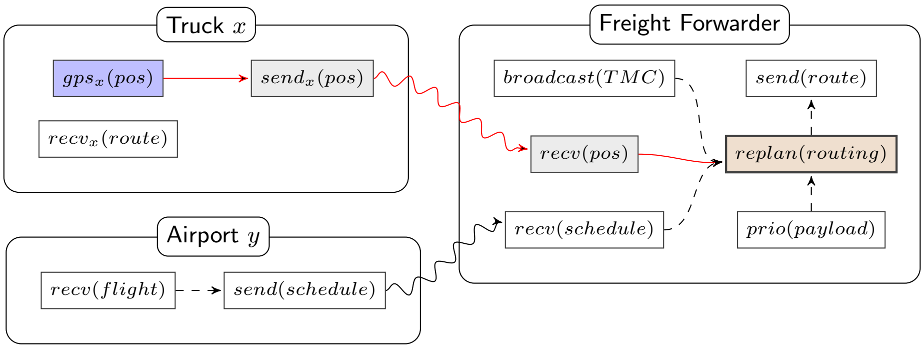


**Fig. 3.** Functional dependencies

We apply a general security goal: *Whenever a certain output action happens, the input actions that presumably led to it must actually have happened.* As an example for a specific security goal, in the following we will use the authenticity requirement: *Whenever a rescheduling action is performed, the GPS coordinates of each truck should be authentic for the dispatcher in terms of origin, content and time.* The formal syntax to describe these requirements in parameterized form is defined as (see [8]):

**Definition 1.** *$auth(a, b, P)$: Whenever an action $b$ happens, it must be authentic for an Agent $P$ that in any course of events that seem possible to him, a certain action $a$ has happened.*

Therefore, our selected authenticity requirement can be written as:

$$auth(gps_x(pos), replan(routing), dispatcher). \tag{Auth 1}$$

We will use the authenticity requirement (Auth 1) to describe the reasoning process with the help of an appropriate operational model.

There are of course many other security requirements necessary in this scenario. For example, *while loading a package on the truck the RFID data and the truck driver should be authentic in terms of content and identification number.* Analysis and application in our situational security analysis follow the same procedure as for (Auth 1). Therefore, we will exemplify only (Auth 1) in the following.

## 3 Formal Model

In order to analyze the system behavior with tool support, an appropriate formal representation has to be chosen. In our approach, we use an operational finite state model of the behavior of the given process which is based on *Asynchronous Product Automata (APA)*, a flexible operational specification concept for cooperating systems [16]. An APA consists of a family of so called *elementary automata* communicating by common components of their state (shared memory). We now introduce the formal modeling techniques used, and illustrate the usage by our application example.

**Definition 2 (Asynchronous Product Automaton (APA)).** *An* Asynchronous Product Automaton $\mathbb{A} = ((Z_s)_{s\in\mathbb{S}}, (\Phi_t, \Delta_t)_{t\in\mathbb{T}}, N, q_0)$ *consists of a family of* state sets $Z_s, s \in \mathbb{S}$, *a family of* elementary automata $(\Phi_t, \Delta_t)$, *with* $t \in \mathbb{T}$, *a* neighborhood relation $N : \mathbb{T} \rightarrow \mathfrak{P}(\mathbb{S})$ *and an* initial state $q_0 = (q_{0s})_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$. *$\mathbb{S}$ and $\mathbb{T}$ are index sets with the names of state components and of elementary automata and $\mathfrak{P}(\mathbb{S})$ is the power set of $\mathbb{S}$. For each elementary automaton $(\Phi_t, \Delta_t)$ with* Alphabet $\Phi_t$, *its* state transition relation *is $\Delta_t \subseteq \bigtimes_{s\in N(t)}(Z_s) \times \Phi_t \times \bigtimes_{s\in N(t)}(Z_s)$. For each element of $\Phi_t$ the state transition relation $\Delta_t$ defines state transitions that change only the state components in $N(t)$. An APA's (global)* states *are elements of* $\bigtimes_{s\in\mathbb{S}}(Z_s)$. *To avoid pathological cases it is generally assumed that $N(t) \neq \emptyset$ for all $t \in \mathbb{T}$. An elementary automaton $(\Phi_t, \Delta_t)$ is* activated *in a state $p = (p_s)_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$ as to an* interpretation $i \in \Phi_t$, *if there are $(q_s)_{s\in N(t)} \in \bigtimes_{s\in N(t)}(Z_s)$ with $((p_s)_{s\in N(t)}, i, (q_s)_{s\in N(t)}) \in \Delta_t$. An activated elementary automaton $(\Phi_t, \Delta_t)$ can execute a state transition and produce a* successor state $q = (q_r)_{r\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$, *if $q_r = p_r$ for $r \in \mathbb{S} \setminus N(t)$ and $((p_s)_{s\in N(t)}, i, (q_s)_{s\in N(t)}) \in \Delta_t$. The corresponding state transition is $(p, (t, i), q)$.*

A simplified model of the part of the "Freight Forwarder" business process shown in Fig 2 contains the APA state components *pstate* and *event* representing the current process state and event. Formally, $\mathbb{S} = \{pstate, event\}$, with $Z_{event} = \{imminent\_truck\_delay, \ldots, replan, \neg replan\}$, $Z_{pstate} = \ldots$.

The *elementary automata* $\mathbb{T} = \{identify\_critical\_payload, \ldots, select\_plan\}$ represent the possible actions that the systems can take. The neighborhood relation between elementary automata and state components of the APA model is depicted by the edges in Fig. 4.

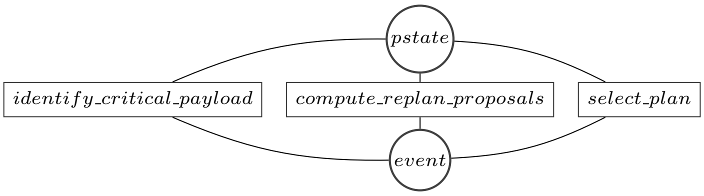


**Fig. 4.** Elementary automata and state components in the APA process model

Formally, the behavior of our operational APA model of the business process is described by a reachability graph. In the literature this is sometimes also referred to as labeled transition system (LTS).

**Definition 3 (Reachability graph).** *The behavior of an APA is represented by all possible coherent sequences of state transitions starting with initial state* $q_0$*. The sequence* $(q_0, (t_1, i_1), q_1)(q_1, (t_2, i_2), q_2) \dots (q_{n-1}, (t_n, i_n), q_n)$ *with* $i_k \in \Phi_{t_k}$ *represents one possible sequence of actions of an APA. State transitions* $(p, (t, i), q)$ *may be interpreted as labeled edges of a directed graph whose nodes are the states of an APA:* $(p, (t, i), q)$ *is the edge leading from* $p$ *to* $q$ *and labeled by* $(t, i)$*. The subgraph reachable from* $q_0$ *is called* reachability graph *of an APA.*

We use the *SH verification tool* [16] to analyse the process model. This tool provides components for the complete cycle from formal specification to exhaustive validation as well as visualisation and inspection of computed reachability graphs and minimal automata. The applied specification method based on APA is supported. The tool manages the components of the model, allows to select alternative parts of the specification and automatically *glues* together the selected components to generate a combined model of the APA specification.

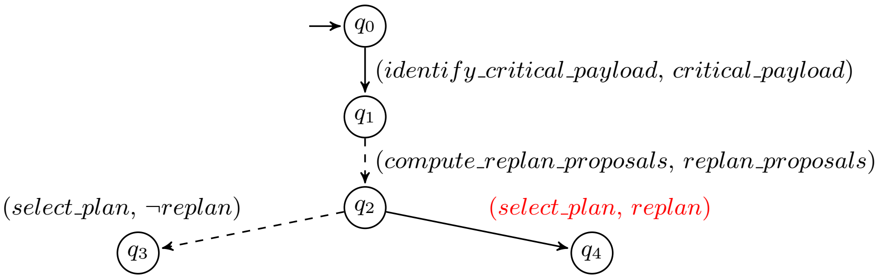

$q_0$
$(identify\_critical\_payload, critical\_payload)$
$q_1$
$(compute\_replan\_proposals, replan\_proposals)$
$(select\_plan, \neg replan)$
$q_2$
$(select\_plan, replan)$
$q_3$
$q_4$

**Fig. 5.** Close-future (3 Steps) Reachability Analysis

Figure 5 shows the initial part of the reachability graph resulting from the analysis of the model when reaching the part of the business process of the freight forwarder shown in Fig. 2. An example for a state transition of the model in this situation is: ($q_0$,($identify\_critical\_payload$, $critical\_payload$),$q_1$). Please note that there are two different transitions from the state $q_2$ because the interpretation of a variable can have the values *replan* or ¬ *replan*, respectively.

## 4 Runtime Operation and Generated Alerts

During runtime, the events from the business process are used to synchronize the state of the model with the real process. In our exemplary setup, the events are produced by a Complex Event Processing (CEP) engine which is provided by one of the project partners. The events are described by an XML schema and communicated by the Java Message Service (JMS). The events from the event bus are used to provide the information about the state and input to the business process. In our finite state model, this information is represented in the state components *pstate* and *event* (cf. Fig. 4). This constitutes the initial state of the model from which a simulation is then started. In addition to the predicted system behavior, we also need the information on the security requirements in order to identify critical situations. In [18] we proposed to use APA to specify meta-events, which match security critical situations, to generate alerts. However, since this is slow and not easily usable by end-users, we decided to build the matching algorithm directly into the SH verification tool. We use monitor automata [22] to specify the security requirements graphically. These automata monitor the behaviour of the abstract system during the run of the simulation and provide interfaces to trigger alerts. This concept could be further extended to make use of the built-in temporal logic based reasoning component if more complex reasoning is necessary.

### 4.1 Security Reasoning – No Authenticity Approval of GPS Event

In order to demonstrate the use of process models at runtime, let us assume the following situation. We are currently at logical time 0 as depicted on the timeline in Figures, 7, 8, 9, 10. We further assume that the trusted agent inspects the events generated by GPS units of the trucks and sends additional events which attest to the authenticity of each GPS event within a timeframe of 2 logical time units. Please note that it is also a possibility that the trusted agent would filter the events and only let authentic events pass the filter. We furthermore assume that we know from the analysis of dependencies of actions and specifically from the requirement (Auth 1) that whenever a rescheduling action is performed, the GPS coordinates of each truck should be authentic for the process planner in terms of origin, content and time.

We now describe the reasoning process where the authenticity of the GPS event is not approved by the trusted agent. In the diagrams we use pentagon symbols to depict events on the event bus such as GPS information and we use triangles to depict Security Warnings (SW), Predictive Security Alerts (PSA) and Security Alerts (SA) generated by the reasoning process.

**Step 1: A GPS event is received**

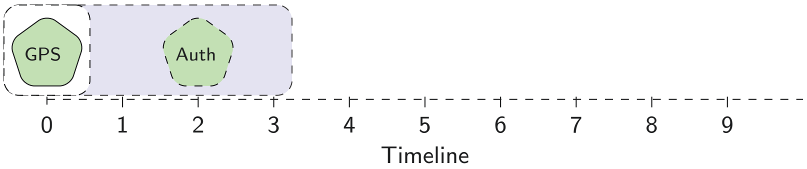


**Fig. 6.** Security Reasoning - Authenticity Approval of GPS Event - step 1

Figure 6 shows the situation when a GPS event is received. This event is matching a precondition in the requirement pattern: *GPS needs confirmation* *in 2 steps*. This requirement (warn-level) is triggered by the GPS event. The reachability analysis reveals no critical actions within the scope (3 steps) of the analysis. We conclude from Fig. 6 that everything is OK at this point. A future event might confirm authenticity of the GPS location received.

### Step 2: Confirmation of GPS event not received

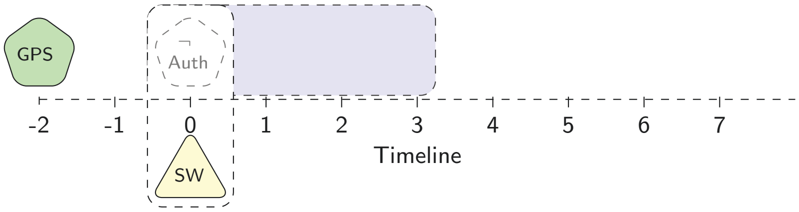


**Fig. 7.** Security Reasoning - No Authenticity Approval of GPS Event - step 2

Figure 7 shows the situation when an expected event from the trusted agent, namely the authenticity approval of this GPS event is recognized as missing. The missing event indicates a broken security requirement: *GPS needs confirmation in 2 steps*. The reachability analysis in this situation shows that no other security requirement will be triggered within the scope of the analysis. However, some forthcoming security relevant action might require authenticity of this GPS event. Therefore, an alert action associated with a broken warn-level requirement, such as issuing a security warning (SW), is now triggered.

### Step 3: Replan event in analysis scope

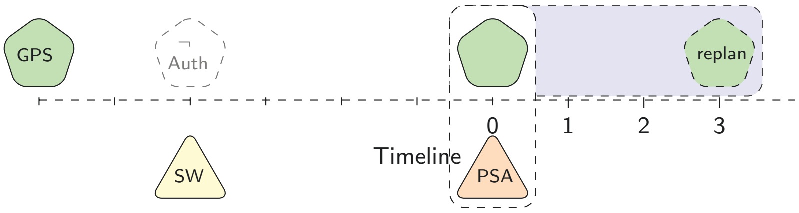


**Fig. 8.** Security Reasoning - No Authenticity Approval of GPS Event - step 3

In Fig. 8, an arbitrary event is received from which, in one possible execution sequence of the business process, a *replan* event is reachable within the scope of the analysis. In our scenario *imminent_ truck_delay* is such an event. The reachability graph is similar to the one depicted in Fig. 5. It shows that the *select_plan* action may happen in the future if *replan* is chosen. But there is another possible path in the graph where *replan* is not chosen. The *replan* event in the prediction scope is matching a precondition in a requirement pattern: $auth(GPS, replan, dispatcher)$, but the GPS event is not approved to be authentic. Therefore, a *replan* event with broken security requirement is possible. An action associated with this (possibly) broken alert-level requirement, such as issuing a predictive security alert (PSA), is now executed.

### Step 4a: Expected *replan* event received

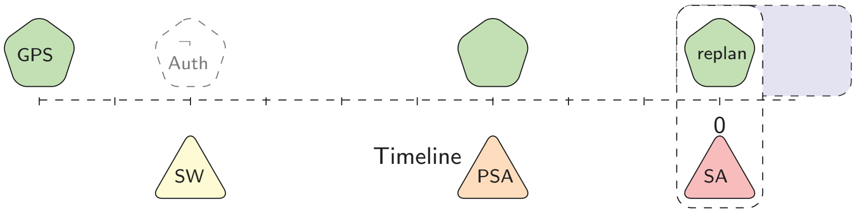


**Fig. 9.** Security Reasoning - No Authenticity Approval of GPS Event - step 4a

Figure 9 shows the situation when a *replan* event is received as predicted (cf. Fig. 5 transition $q_2 \rightarrow q_4$). At this time we know that the security requirement (Auth 1) is broken. Therefore, an action associated with a broken alert-level requirement, such as issuing a security alert (SA), is now executed.

### Step 4b: Predicted *replan* event not received after step 3

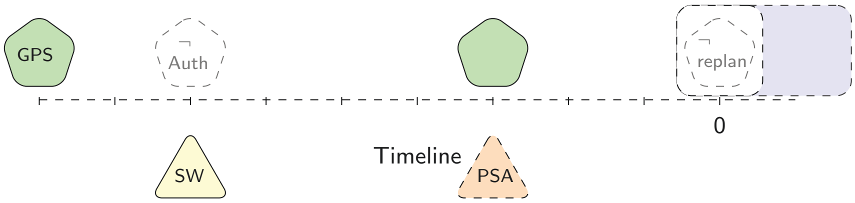


**Fig. 10.** Security Reasoning - No Authenticity Approval of GPS Event - step 4b

Figure 10 shows the situation when a *replan* event is not received as expected (cf. Fig. 5 transition $q_2 \rightarrow q_5$). In this case, we know that the issued predictive security alert (PSA) was a “False Positive”, so a corrective action may be necessary. Corrective actions might be the reduction of a general security warning level or lifting of restrictions on the business process depending on the operating environment. However, the security warning issued in step 2 is still valid because some future event might require authenticity of the GPS event.

## 5 Related Work

The work presented here combines specific aspects of security analysis with generic aspects of process monitoring, simulation, and analysis. The background of those aspects is given by the utilization of models at runtime [6]. A blueprint for our architecture of predictive security analysis is given in [18].

Security analysis *at development-time* to identify violations of security policies is usually integrated in the security requirements engineering process. An overview of current security requirements engineering processes is given in [5,13]. The security requirements elicitation methods developed in [7] are used in section 2 to derive the requirements which are needed to assess possible security policy violations at runtime. A formalized approach for security risk modeling in the context of electronic business processes is given in [21]. It touches also the aspect of simulation, but does not incorporate the utilization of runtime models. Approaches that focus security models at runtime are given in [14] or in [12]. Morin et. al [14] propose a novel methodology to synchronize an architectural model reflecting access control policies with the running system. Therefore, the methodology emphasizes policy enforcement rather than security analysis. The integration of runtime and development-time information on the basis of an ontology to engineer industrial automation systems is discussed in [12].

Process monitoring has gained some popularity recently in the industrial context prominently accompanied with the term Business Activity Monitoring (BAM). The goal of BAM applications, as defined by Gartner Inc., is to process events, which are generated from multiple application systems, enterprise service buses or other inter-enterprise sources in real-time in order to identify critical business key performance indicators and get a better insight into the business activities and thereby improve the effectiveness of business operations [11]. Recently, runtime monitoring of concurrent distributed systems based on linear temporal logic (LTL), state-charts, and related formalisms has also received a lot of attention [9,10]. However, these works are mainly focused on error detection, e.g., concurrency related bugs. A classification for runtime monitoring of software faults is given in [1]. Patterns to allow for monitoring security properties are developed in [20]. In the context of BAM applications, in addition to these features we propose a *close-future* security analysis as it is detailed in section 4. Our analysis provides information about possible security policy violations reinforcing the security-related decision support system components.

Different categories of tools applicable for simulation of business processes including process modeling tools are based on different semi-formal or formal methods such as Petri Nets [3] or Event-driven Process Chains (EPC) [2]. Some process management tools such as FileNet [15] offer a simulation tool to support the design phase. Also, some general-purpose simulation tools such as CPNTools [19] were proven to be suitable for simulating business processes. However, independently from the tools and methods used, such simulation tools concentrate on statistical aspects, redesign and commercial optimization of the business process. On the contrary, we propose an approach for *on-the-fly* dynamic simulation and analysis on the basis of operational APA models detailed in section 3. This includes consideration of the current process state and the event information combined with the corresponding steps in the process model.

## 6 Conclusions and Further Work

In this paper we demonstrated the application of runtime models to analyze the security status of business processes and to identify possible violations of the security policy in the near future. Therefore, we started with a business process model from the logistics domain and analyzed corresponding security requirements. Utilizing both development-time models we derived a runtime model. The runtime model consumes events from the runtime environment, evaluates current violations of the security policy, and identifies close-future violations of the security policy. Within the logistics domain we applied our approach to identify situations in which an attacker might try to disrupt or degrade the process performance. By issuing predictive security alerts, users or operators (in this case: the dispatcher in the logistic process) are able to act securely without the need to understand the security policy or infrastructure in detail.

Other novel uses of such models at runtime can enable anticipatory impact analysis, decision support and impact mitigation by adaptive configuration of countermeasures. The project MASSIF (`http://www.massif-project.eu/`), a large-scale integrating project co-funded by the European Commission, addresses these challenges within the management of security information and events in service infrastructures. In MASSIF [17] we will apply the presented modeling concept in four industrial domains: (i) the management of the Olympic Games IT infrastructure; (ii) a mobile phone based money transfer service, facing high-level threats such as money laundering; (iii) managed IT outsource services for large distributed enterprises; and (iv) an IT system supporting a critical infrastructure (dam).

**Acknowledgments.** The work presented here was developed in the context of the project MASSIF (ID 257475) being co-funded by the European Commission within the Seventh Framework Programme and the project Alliance Digital Product Flow (ADiWa) (ID 01IA08006F) which is funded by the German Federal Ministry of Education and Research.

## References

1. Delgado, N., Gates, A., Roach, S.: A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Transactions on Software Engineering 30(12), 859–872 (2004)
2. Dijkman, R.M.: Diagnosing differences between business process models. In: Business Process Management (BPM 2008). LNCS, vol. 5240, pp. 261–277. Springer (2008)
3. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology 50(12), 1281–1294 (2008)

4. Döhring, M., Zimmermann, B., Karg, L.: Flexible workflows at design- and runtime using BPMN2 adaptation patterns. In: Business Information Systems (BIS 2011), LNBIP, vol. 87, pp. 25–36. Springer (2011)
5. Fabian, B., Gürses, S., Heisel, M., Santen, T., Schmidt, H.: A comparison of security requirements engineering methods. Requirements engineering 15(1), 7–40 (2010)
6. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: Future of Software Engineering. pp. 37–54. IEEE (2007)
7. Fuchs, A., Rieke, R.: Identification of Security Requirements in Systems of Systems by Functional Security Analysis. In: Architecting Dependable Systems VII, LNCS, vol. 6420, pp. 74–96. Springer (2010)
8. Gürgens, S., Ochsenschläger, P., Rudolph, C.: On a formal framework for security properties. Computer Standards & Interfaces 27, 457–466 (2005)
9. Kazhamiakin, R., Pistore, M., Santuari, L.: Analysis of communication models in web service compositions. In: World Wide Web (WWW 2006). pp. 267–276. ACM (2006)
10. Massart, T., Meuter, C.: Efficient online monitoring of LTL properties for asynchronous distributed systems. Tech. rep., Université Libre de Bruxelles (2006)
11. McCoy, D.W.: Business Activity Monitoring: Calm Before the Storm. Gartner Research (2002)
12. Melik-Merkumians, M., Moser, T., Schatten, A., Zoitl, A., Biffl, S.: Knowledge-based runtime failure detection for industrial automation systems. In: Workshop Models@run.time. pp. 108–119. CEUR (2010)
13. Mellado, D., Blanco, C., Snchez, L.E., Fernndez-Medina, E.: A systematic review of security requirements engineering. Computer Standards & Interfaces 32(4), 153–165 (2010)
14. Morin, B., Mouelhi, T., Fleurey, F., Le Traon, Y., Barais, O., Jézéquel, J.M.: Security-driven model-based dynamic adaptation. In: Automated Software Engineering (ASE 2010). pp. 205–214. ACM (2010)
15. Netjes, M., Reijers, H., Aalst, W.P.v.d.: Supporting the BPM life-cycle with FileNet. In: Exploring Modeling Methods for Systems Analysis and Design (EMM-SAD 2006). pp. 497–508. Namur University Press (2006)
16. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing 10(4), 381–404 (1998)
17. Prieto, E., Diaz, R., Romano, L., Rieke, R., Achemlal, M.: MASSIF: A promising solution to enhance olympic games IT security. In: International Conference on Global Security, Safety and Sustainability (ICGS3 2011) (2011)
18. Rieke, R., Stoynova, Z.: Predictive security analysis for event-driven processes. In: Computer Network Security, LNCS, vol. 6258, pp. 321–328. Springer (2010)
19. Rozinat, A., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.J.: Workflow simulation for operational decision support. Data & Knowledge Engineering 68(9), 834–850 (2009)
20. Spanoudakis, G., Kloukinas, C., Androutsopoulos, K.: Towards security monitoring patterns. In: Symposium on Applied computing (SAC 2007). pp. 1518–1525. ACM (2007)
21. Tjoa, S., Jakoubi, S., Goluch, G., Kitzler, G., Goluch, S., Quirchmayr, G.: A formal approach enabling risk-aware business process modeling and simulation. IEEE Transactions on Services Computing 4(2), 153–166 (2011)
22. Winkelvos, T., Rudolph, C., Repp, J.: A Property Based Security Risk Analysis Through Weighted Simulation. In: Information Security South Africa (ISSA 2011). IEEE (2011)

# P15

# ARCHITECTING A SECURITY STRATEGY MEASUREMENT AND MANAGEMENT SYSTEM

| | |
|---|---|
| **Title** | Architecting a Security Strategy Measurement and Management System |
| **Authors** | Roland Rieke, Julian Schütte, and Andrew Hutchison |
| **Publication** | In *Proceedings of the Workshop on Model-Driven Security*, MDsec'12, pages 2:1–2:6. |
| **ISBN/ISSN** | ISBN 978-1-4503-1806-8 |
| **DOI** | `http://dx.doi.org/10.1145/2422498.2422500` |
| **Publisher** | ACM, New York, NY, USA |
| **Publication Type** | ACM digital library |
| **Status** | Published |
| **Copyright** | 2012, ACM |
| **Contribution of Roland Rieke** | Main Author, editor, and presenter at the workshop on Model-Driven Security 2012.<br>Specific contributions are: (a) Security Information Meta Model (SIMM), (b) contributions to Security Strategy Meta Model (SSMM), and (c) the design of the conceptual security strategy management framework.<br>Related contributions: Roland Rieke also contributed to the closely related paper "Model-Based Security Event Management" [Schütte et al., 2012] and gave a related talk together with Co-Author Andrew Hutchison about "Measuring Progress in Cyber-Security: An Open Architecture for Security Measurement Consolidation" at the 2012 Workshop on Cyber Security and Global Affairs and Global Security Forum [Hutchison & Rieke, 2012]. |

Table 20: Fact Sheet Publication *P15*

Publication P15 [Rieke, Schütte & Hutchison, 2012] addresses the following research question:

RQ11 *How can security analysis at runtime be integrated in a security management strategy?*

This paper presents a model driven approach for architecting a *security strategy measurement and management system*. Concretely, it describes the definition of security objectives for a particular system, and a mechanism for collecting information from operational systems in a manner which enables assessment and measurement of how well the system is fulfilling the security objectives. Existing Security Information and Event Management (SIEM) solutions are limited, while this approach overcomes these contextual restrictions (typically predefined, closed models) offering an extensible and open model, encompassing all parts of the security monitoring and decision support process, namely: (i) detecting threatening events; (ii) putting them in context of the current system state; (iii) explaining their potential impact with respect to some security- or compliance model; and (iv) taking appropriate actions. The proposed deployment model brings together all parts of security runtime management, namely, detection, reporting, handling, and explanation of security incidents, which are to date covered by different systems, such as intrusion detection systems, Complex Event Processing (CEP) engines [Esper contributors and EsperTech Inc., 2012], SIEM systems [AlienVault, 2012; Prelude - CS Group, 2014; Araknos, 2012], intrusion response systems [Shameli-Sendi et al., 2012], cyber attack information systems [Skopik et al., 2012], and governance, risk management, and compliance systems [Racz et al., 2010]. So, the model supports an integration of functionalities of these existing systems into one coherent security strategy management framework.

# Architecting a Security Strategy Measurement and Management System

Roland Rieke
Fraunhofer Institute SIT
Rheinstrasse 75
Darmstadt, Germany
roland.rieke@sit.fraunhofer.de

Julian Schütte
Fraunhofer Institution AISEC
Parkring 4
Garching, Germany
julian.schuette@aisec.fraunhofer.de

Andrew Hutchison
T-Systems South Africa
4 Churchill Close, Bellville
Cape Town, South Africa
andrew.hutchison@t-systems.co.za

## ABSTRACT

The use of formal models to guide security design is appealing. This paper presents a model driven approach whereby security systems in operation can be assessed and measured against various requirements that are defined when the system is created. By aligning with organisational policy, and business requirements of a specific system, design and operation can proceed in a way that allows measurement of how successfully security objectives are being achieved. This paper describes a model driven approach which overcomes the contextual restrictions of existing solutions. In particular, where models have been used previously these have tended to be predefined and closed models, whereas the approach described here is an extensible model that comprises all parts of the security monitoring and decision support process. By means of interlinked semantic concepts, the proposed security strategy meta model provides a way to model security directives at an abstract level, which can be automatically compiled into specific rules for an underlying framework of monitoring, decision support, and enforcement engines.

## Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.2.1 [**Software Engineering**]: Requirements/Specifications; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management,Network monitoring*

## General Terms

Security,Management,Measurement

## Keywords

Security strategy, security monitoring, decision support, security information and event management, information security measurement model, governance and compliance

*MDSec'12* September 30 2012, Innsbruck, Austria
Copyright 2012 ACM 978-1-4503-1806-8/12/09 ...$15.00.

## 1. INTRODUCTION

Cyber Security is an area of great global focus, yet it is both hard to manage and – arguably – even harder to measure. But the two concepts go together: if some sort of *measurement* approach can be implemented, it should at least be possible to *manage* systems better and assess whether they are meeting the security objectives that they were designed with. In spite of the fact that technical security solutions are deployed, there are numerous instances of processes or transactions being compromised. Part of the challenge with security implementations is that they are made in isolation from any formal specification or model of what the security profile should look like. In the absence of a holistic view that extends from business process to logical and technical security realisation, there is high potential for gaps or mismatches to occur. Fueling this situation is the fact that life-cycle approaches to security are not easily applied – or measured.

In this paper we argue for a *meta model* approach to drive security from design to implementation, through an *analysis and refinement* approach, and also through a *security measurement* approach which would enable assessment of the system's performance against the security requirements it was designed for. To achieve a meta model approach for security, several phases are required and in this paper we present a *Security Information Meta Model* (SIMM) consisting of: (a) high level goal setting, (b) security requirements, (c) measurement requirements, and (d) objects of measurement. Through applying this model, high level goals for security can be established and defined. Importantly, measurement objectives can also be developed and stated at this point. By proceeding in this way, security can be designed in such a way that it can be measured (and managed). Activities of analysis and refinement are required to move from security requirements to measurement requirements. In this process, objects of measurement also need to be identified.

In order to cover the operational aspects of this concept, we make use of a *Security Strategy Meta Model* (SSMM) [23] that describes the control flow at runtime, independent from the underlying event description language. A specific rule from a *Security Strategy Model* (SSM) that adheres to the SSMM is called *Security Directive* (SD). A distinguished *Security Strategy Component* controls the execution of the SDs. It can execute a SD or parts of it directly or delegate workload to a specialised *Security Strategy Processing Component* (SSPC). The overall aim is to overcome the contextual restrictions of existing solutions, with their predefined and closed models, and rather to provide an extensible model that spans all parts of the security monitoring and decision support process, namely: (i) detecting threatening events; (ii) putting them in context of the current system state; (iii) explaining their potential impact with respect to some security- or compliance model; and (iv) taking appropriate actions. Depending on the outcome of the analysis of these components, other components that implement decision support and enforcement will be triggered. The proposed SSMM together with the framework of SSPCs could be used as a core of a technology platform for an integrated concept for governance, risk and compliance [19]. Furthermore, we consider the proposed approach to be applicable within the design of a cyber attack information system [24], which uses collaborative detection and response mechanisms for high-level situational awareness and coordination of local incident response.

The structure of the paper is as follows: first the use of *Security Information and Event Management* (SIEM) techniques for information security management in general is discussed; next integration into a system architecture is presented and this is then also contrasted with existing work, positioning how this approach differs from other similar work.

## 2. USING SIEM FOR INFORMATION SECURITY MANAGEMENT

*Information security management* is needed to protect information and information infrastructure assets of an organisation against the risks of loss, misuse, disclosure or damage. It specifically describes controls that an organisation needs to implement to ensure that it is sensibly managing the risks. The ISO/IEC 27000-series comprises information security standards in the context of an *Information Security Management System* (ISMS). Specifically, the ISO/IEC 27004 standard [11] provides guidance on the development and operation of measures and measurement, and reporting of the results, with the aim to help organisations to systematically improve the effectiveness of their ISMSs.

### 2.1 Information Security Measurement

The ISO/IEC 27004 standard defines an *Information Security Measurement Model* (ISMM) that provides a structure linking an *information need* to the relevant *object of measurement*. Furthermore, it describes how the attributes of an object of measurement are quantified and converted to indicators, thus providing a basis for decision making. Figure 1 depicts an abstract view of the ISMM. Specific objects of measurement relevant for the work presented here include the status of information assets protected by the controls and the measurement of process behaviour. This standard further provides a template for an *information security measurement construct* and several examples of concrete measurement constructs. Further examples are given in [15]. The frequency of reporting measurement results is in most of the given examples “monthly”, “quarterly” or “yearly”. The design of measurement constructs as described in the ISO/IEC 27004 standard covers in detail the steps needed to derive the measurement results from a given object of measurement [15]. However, the method for identification of the objects of measurement from the information needs is not specified in detail (cf. the dashed arrow in Fig. 1).

In the following, we show how we map relevant parts of the information security measurement process to a SIEM information flow. While the former usually takes months or years to be updated, through manual inspection and creation of checklists, the latter allows a near-realtime observation of incidents and the mapping of them back to the information security management requirements. Thus, by the application of our model, we expect a semi-automatic and significantly faster update cycle of compliance checks.

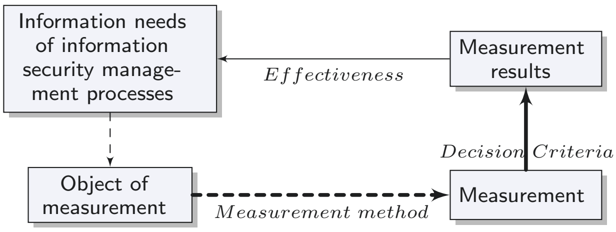


**Figure 1: Abstract view of the ISMM**

### 2.2 Security Information and Event Management

SIEM systems provide important security services. They collect and analyse data from different sources, such as sensors, firewalls, routers or servers, and provide decision support based on anticipated impact analysis. This enables timeous response to (or prevention of) attacks as well as impact mitigation by adaptive configuration of countermeasures. The frequency of reporting measurement results is in most cases very high. In [18], e.g., it is reported that for the Beijing 2008 Olympic Games, more than 12 million IT security events were collected and filtered each day to detect any potential security risk for the Olympic Games IT systems. From these, less than 100 were identified as real issues. All were resolved, with no impact at all on the Olympic Games.

The rules for measurements and correlation are usually a mixture of predefined rules from the SIEM system provider and specific rules from the SIEM system user. Compared to the ISMM, a SIEM based approach presents several key advantages:

1. the measurement frequency is much higher,
2. the system is tool based and most actions are executed automatically, and
3. a decision support system or intrusion response system [25] may offer automatic countermeasures.

There are also disadvantages though:

1. the rules are written in vendor specific notation,
2. back-traceability across layers of derived measures from base measures is not always possible,
3. rules don’t necessarily use contextual information,
4. the effects of countermeasures are not clear,
5. there is no clear derivation of the measurement rules from the information needs,
6. therefore, there is no traceback possibility from measurement results to information needs, and

7. because there are also – best practice – rules used from external sources, there is no clear way to express how these rules are related to the company goals.

In order to combine the advantages of both ISMM and SIEM concepts, we propose a Security Strategy Meta Model (SSMM) that addresses the above mentioned disadvantages.

## 2.3 Interlinked Semantic Concepts

The SIEM information flow is based on rules specifying which system behaviours to observe. However, simply reacting to individual rules is of little help for users who need to understand the actual incident that has been detected and its implications in terms of the security model. For this purpose, we propose firstly, to refine the left side of the ISMM by an *Security Information Meta Model* (SIMM), which should be derived in an measurement requirements elicitation process [20]. The most important objectives of this process are:

**Coverage of Security Goals.** The requirements elicitation method should ensure that uncovered aspects of high-level security goals are revealed.

**Information Needs.** A lack of SIEM monitoring capabilities would prevent the derivation of assumptions necessary for the reasoning process. This should be detected in the requirements elicitation process.

**Sufficiency of Monitoring Capabilities.** Assumptions can be derived from the monitoring capabilities for reasoning whether the given requirements are fulfilled under these assumptions. This reasoning process can't be successful if monitoring capabilities are insufficient or can't be assigned to entities used in the current abstraction level of the system model.

**Traceability.** The derived relations between security event measurements, the associated security requirements and corresponding assumptions, and the security goals can be used to identify the concrete high-level goals affected by the measurements.

Secondly, we propose to use an SSMM that comprises all aspects of SIEM functionality as well as countermeasure configuration support in order to cover the operational aspects of the overall security management goal. By means of interlinked semantic concepts, the SSMM provides a way for users to model incident detection rules at an abstract level, which are automatically compiled into specific rules for an underlying *Complex Event Processing* (CEP) engine. Thus, the SSMM serves as a generic and extensible model on top of a specific rule language used for actual event evaluation. At the model level, these rules are linked to environmental conditions, countermeasures, and explanations based on an external security model. The SSMM is constructed of four parts : *on*, : *if*, : *do*, and : *why*, which are derived from the measurement requirements on the one hand, and which refer back to the SIMM on the other hand (cf. Fig. 2).

The : *on* part specifies the incident detection by means of an *event stream property*. This addresses the first of the "disadvantages" mentioned previously, by abstracting from specific event processing languages. Once specified, event stream properties can be reused across different CEP engines and do not require users to be expert in such systems. This supports a separation of duty, where security engineers can concentrate on modelling security information measurement and do not necessarily need to be knowledgeable regarding all the technical details of a CEP engine.

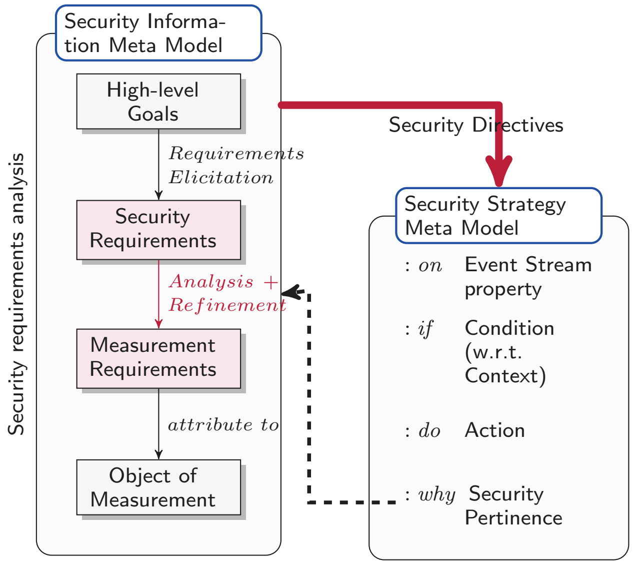

Security Information Meta Model
High-level Goals
Requirements Elicitation
Security Requirements
Analysis + Refinement
Measurement Requirements
attribute to
Object of Measurement
Security requirements analysis
Security Directives
Security Strategy Meta Model
: on Event Stream property
: if Condition (w.r.t. Context)
: do Action
: why Security Pertinence

**Figure 2: Security Information Meta Model and Security Strategy Meta Model**

Addressing the second stated "disadvantage", the model is able to express correlations of incoming events "horizontally" (i.e., as steps in a workflow), as well as "vertically". While most SIEMs focus on *horizontal* correlation, *vertical* correlation is an interesting feature, because it allows the linking of information across different levels of abstraction, such as events from an intrusion detection system with the currently threatened protection goal. To address the third "disadvantage", the : *if* part of the model allows for inclusion of context information. This is of special importance for stateful incident detection, as encountered in the monitoring of ongoing processes, and also to increase the likelihood of discovery of a targeted attack. Addressing the fourth "disadvantage", our model allows combination of SIEM functionality – for detecting incidents – with an actual handling of these. This is modelled by the : *do* part of the model that refers to countermeasures to be taken, ranging from a simple reporting, to autonomous re-configurations of the system. In [22], we have shown how such an autonomous and goal-driven re-configuration can be realised. In order to close the traditional *plan-do-check-act* [8] cycle of Information Security Measurement, addressing the fifth, sixth and seventh "disadvantages", we finally need to link security incident detection to the high-level security requirement. This is achieved by the : *why* part of the SSMM, represented by the dotted arrow in Fig. 2. The : *why* part should contain information similar to the *information security measurement construct* defined in the ISO/IEC 27004 standard.

In [23], we have defined a language from which to form a model that satisfies these requirements.

## 3. INTEGRATION INTO SYSTEM ARCHITECTURE

We now describe a mapping of the SSMM to the components of a proposed monitoring infrastructure. The goal is to enable the inclusion of existing engines, which need not know about the overall security strategy but only receive specific tasks in their respective language. We first describe the concept and then continue with a prototypical implementation.

## 3.1 Security Strategy Processing Components

Conceptually, the implementation of the processing of the SSMM is composed of SSPCs. The main components and some optional components of the proposed system architecture are illustrated in Fig. 3. A distinguished *Security Strategy Component* controls the execution of the SDs. It can execute a SD, or parts of it, directly or delegate the workload to specialised components. The Security Strategy Component initially gets the SSM from the *Security Information Modeller*. It parses the SDs of the SSM, identifies the responsible SSPCs for each subtask, and distributes a respective configuration to the relevant SSPC. The CEP engine normally processes the : *on* part of the SD. The security monitoring probes, which are described at an abstract level in the SSM, have to be compiled to the configuration language of the actual CEP engine, if an engine specific specification is not given in the : *on* part of the SD. Optionally, the events could be processed directly. Furthermore, other event processing components such as intrusion visualisation could be triggered. The : *if* part of the SD can be processed by several different components, responsible for different aspects of the domain or several domains. One component, which will be needed in most implementations, is that responsible for the provisioning of the network state information. Other components could, e.g., provide cyber-physical models, workflow specifications, business process information or process visualisation. For example, in the project MASSIF [3] we are currently implementing an advanced SIEM architecture that comprises an *Attack Modeling and Security Evaluation Component* (AMSEC) [13] and a *Predictive Security Analyser* (PSA) [9]. The AMSEC component provides attack scenario recognition by real-time event analysis and prognosis of future attack steps by recognition of the attacker model. The PSA component provides advanced, application aware security monitoring capabilities. Specifically, it supports close-future process behaviour simulation and prediction of possible security violations. Prior to the start of the engine, the process description and security goals/events are transformed into "PSA understandable" models, which are then used for the continuous real-time analysis and close-future simulation. Thus, : *if* components such as AMSEC and PSA provide situational awareness with regard to network state, attack state, and application state.

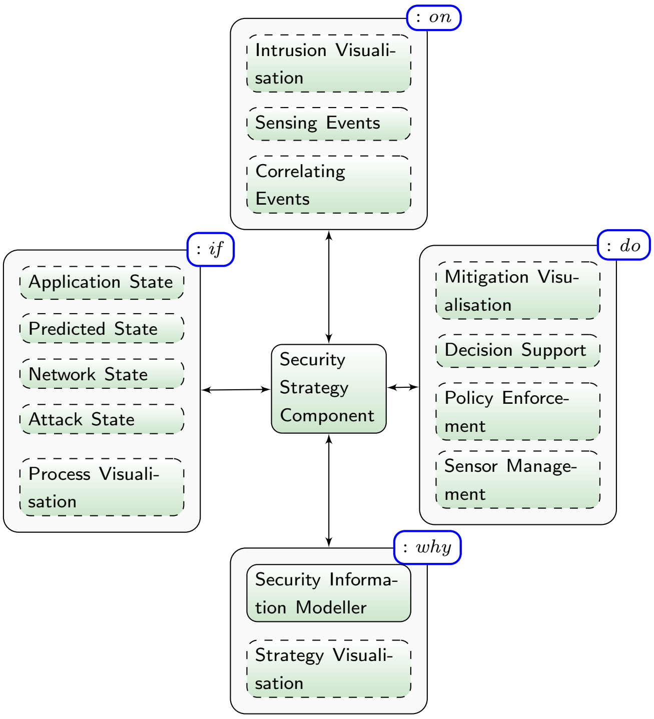


**Figure 3: Conceptual components of the framework**

Depending on how the : *if* condition evaluates, the respective : *do* components will be triggered. These components can implement, e.g., *simulative mitigation visualisation*, *decision support*, *policy enforcement* or *sensor management*. A sensor management component can control the configuration of sensors in a monitored system, e.g., the (de-)activation and the adaptation of the sampling rate to an optimal level [5].

A *security information modeller component* is responsible for maintaining the security strategy and a *strategy visualisation component* can help to assist in the : *why* determination, thus resolving the traceability requirement.

As a special case, the functionalities of some or all components could also be implemented within one engine. In this case no translation into the specific configuration languages is needed and the SDs could be interpreted directly.

## 3.2 Prototype Implementation

We have implemented the model in a prototype in order to test whether it can be used, as intended, for detecting security incidents. Additionally, the prototype implementation should confirm that our component based system architecture is applicable and the idea of SSPC extensions is practical. For this purpose, we implemented a simple Security Strategy Component which processes a given SSM and coordinates the different SSPCs for evaluating it. When the SSM is loaded, the Security Strategy Component first parses the : *on* part of a SD and transforms it into a query for the registered CEP engine. Then, the query is registered at the respective engine and the Security Strategy Component receives a callback whenever the query is triggered, i.e., the : *on* part of the SD has been met. In that case, the Security Strategy Component creates an *evaluation context* object, which acts as container during the evaluation process, and writes the attributes received from the event stream to it. The evaluation context is then passed on to subsequent components for evaluating the condition in the : *if* part of the SD. If the condition has evaluated to "true", the Security Strategy Component loads the action components indicated by the : *do* part and invokes them, passing the evaluation context object as parameter. Because in the prototype, the : *why* part provides merely explanatory reasons, it is not involved in the evaluation process and can rather be explored by users to investigate security implications of the detected incident.

Each of the components, i.e., the Security Strategy Component, as well as the SSPCs, has been created in the form of OSGi bundles and communicates over R-OSGi, a binary RPC protocol. This allows us to dynamically load and unload components, even from a remote repository, so that it is possible to support additional actions by providing respective components in the repository. As an event correlation engine, we have used Esper [2], which comes with its own EPL query language. Listing 1 shows an example of an event description (written in Turtle notation), referring to an anomaly in the traffic to the syslog service, and in Listing 2 its translation into EPL. While the EPL representation is more compact, it is only applicable to the specific CEP engine and does not bear any semantics which could be linked to external models of security requirements and controls.

**Listing 1: Modelled Event Condition**

```
:historyEvent
      :hasName "HistoryDBServerAnomaly" ;
      :hasCriterion [ :hasParam1 "avgTraffic" ;
                      :hasParam2 42^^float ;
                      :hasBooleanOp :gt ] ;
      :hasExtractor [
            :hasEventChannel [ rdf:type :SyslogChannel
                  :hasFields "sourceIP" ;
                  :hasName "IPStreamToIPX" ;
                  :hasFields "FIXEDdestIP","traffic" ]
            :hasFunction [
                  :hasParam1 "traffic" ;
                  :providesVariable "avgTraffic" ;
                  :hasScope "30 sec" ;
                  :hasOp :avg ] .
```

**Listing 2: Generated EPL Query**

```
SELECT   sourceIP?,
         avg(cast(traffic?,float)) AS avgTraffic
FROM     SyslogChannel.win:time(30 sec)
HAVING   cast(avgTraffic?,float)>42
```

The combination of our semantic model with a CEP engine allows us to efficiently evaluate incoming event patterns, while still being able to refer to their semantic description, once the pattern has been found. While an extensive evaluation of our prototype is still outstanding, first results are encouraging and make us confident that it is practically applicable.

## 4. RELATED WORK

Work related to ours is on the one hand concerned with modelling security-relevant information in a way that creates the possibility to reason about it and link it to the ISMM [11]. On the other hand, we review current SIEM systems to highlight how they could be improved by adding a model-based SIMM. In this paper, we rely on the overall plan-do-check-act cycle [8] and the information flow described by the ISO27004 standard [11].

In [10], an approach to create ISO27001-based metrics based on a security ontology is proposed. While it lacks the automatic gathering of measurements, it could serve as a later extension to our SIMM, which is more focused on measurable technical events. Further, linking semantic attacker models to the : *why* part of the SSMM could be promising (cf. the AMSEC model [13]). Another example for a potential information source is the *Engineering Knowledge Base* (EKB) [16], which is an ontology relating to sensor values and combining run-time with development-time models. It is focused on the analysis of industrial automation systems, and is used to define SPARQL or SWRL queries over sensor definitions. As we have a similar goal of finding inconsistencies, we believe that an approach like the EKB could help defining which inconsistencies to look for in event streams, and thus which measurement points might indicate violations of the security requirements. Other approaches of interest to this end are the modelling concepts in [12], where business, application, physical, and technical information is merged and related, as well as concepts to use event-triggered rules for sensing and responding to business situations in [21]. In this paper, we focused on a model to bridge the gap between high-level security measurements and data gathered by SIEM engines, like OSSIM [4], Prelude [17], or Akab [1]. OSSIM detects events at the network layer and stores respective attributes like *IP address* or *port number* in a relational database. Thus, while it is possible to link these attributes to our model, OSSIM itself does not support reasoning over gathered data, nor extending its model. Similarly, Akab [1] is a SIEM appliance for monitoring network events. It uses a proprietary event format and also stores collected events persistently in a database. Prelude [17] is an open source SIEM framework which relies on the open IDMEF [7] event format. Also related to the model-based security information measurement that we envisage are commercial tools RSA Archer, ArcSight ESM, or IBM Tivoli Security Information and Event Manager [6]. Although they also aim at relating incidents to compliance catalogues and corporate policies, they rely on predefined event structures, comprising specific technical attributes [14]. The RSA Archer Threat Monitor manages an assets catalogue and links it to security-relevant information, such as known vulnerabilities and patch levels. It does not however feature an extensible and semantic model which would enable automatic reasoning regarding the implications of a detected incident as it relates to the affected security requirements. It could also make amendments based on information from external sources like our PSA.

## 5. CONCLUSION

This paper has presented a model driven approach for architecting a security strategy measurement and management system. Concretely, it has described the definition of security objectives for a particular system, and a mechanism for collecting information from operational systems in a manner which enables assessment and measurement of how well the system is fulfilling the security objectives. Existing SIEM solutions are limited, while this approach overcomes these contextual restrictions (typically predefined, closed models) offering an extensible and open model, encompassing all parts of the security monitoring and decision support process, namely: (i) detecting threatening events; (ii) putting them in context of the current system state; (iii) explaining their potential impact with respect to some security- or compliance model; and (iv) taking appropriate actions. The proposed deployment model brings together all parts of security runtime management, namely, detection, reporting, handling, and explanation of security incidents, which are to date covered by different systems, such as intrusion detection systems, CEP engines [2], SIEM systems [4, 17, 1], intrusion response systems, cyber attack information systems [24], and governance, risk management, and compliance systems [19]. So, the model supports an integration of functionalities of these existing systems into one coherent security strategy management framework.

## 6. ACKNOWLEDGMENTS

The work presented here was developed in the context of the projects MASSIF (ID 257475) being co-funded by the European Commission within the Seventh Framework Programme, the project SealedCloud being funded by the German Federal Ministry of Economics and Technology (BMWi), and the project ACCEPT (ID 01BY1206D) being funded by the German Federal Ministry of Education and Research.

## 7. REFERENCES

[1] Araknos website. http://www.araknos.it/en.html, 2012. [Online; accessed 16-Sep-2012].

[2] Esper – Complex Event Processing. http://esper.codehaus.org/, 2012. [Online; accessed 16-Sep-2012].

[3] Project MASSIF website. http://www.massif-project.eu/, 2012. [Online; accessed 16-Sep-2012].

[4] AlienValult. AlienVault Unified SIEM. http://alienvault.com/, 2012. [Online; accessed 16-Sep-2012].

[5] L. Baumgärtner, P. Graubner, M. Leinweber, R. Schwarzkopf, M. Schmidt, B. Seeger, and B. Freisleben. Mastering Security Anomalies in Virtualized Computing Environments via Complex Event Processing. In *Proceedings of the The Fourth International Conference on Information, Process, and Knowledge Management (eKNOW 2011)*, pages 76–81. XPS, 2012.

[6] A. Buecker, J. Amado, D. Druker, C. Lorenz, F. Muehlenbrock, and R. Tan. *IT Security Compliance Management Design Guide with IBM Tivoli Security Information and Event Manager*. IBM Redbooks, July 2010. ISBN 0-7384-3446-9.

[7] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765 (Experimental), March 2007.

[8] W. E. Deming. *The new economics for industry, government, education*. Massachusetts Institute of Technology, Center for Advanced Engineering Study, Cambridge, MA, 1993.

[9] J. Eichler and R. Rieke. Model-based Situational Security Analysis. In *Proceedings of the 6th International Workshop on Models@run.time at the ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (MODELS 2011)*, volume 794 of *CEUR Workshop Proceedings*, pages 25–36. 2011.

[10] S. Fenz. Ontology-based generation of it-security metrics. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1833–1839, New York, NY, USA, 2010. ACM.

[11] I. Iec. ISO/IEC 27004:2009 - Information technology - Security techniques - Information security management - Measurement. *ISOIEC*, 2009.

[12] F. Innerhofer-Oberperfler and R. Breu. Using an enterprise architecture for it risk management. In J. H. P. Eloff, L. Labuschagne, M. M. Eloff, and H. S. Venter, editors, *ISSA*, pages 1–12. ISSA, Pretoria, South Africa, 2006.

[13] I. Kotenko, A. Chechulin, and E. Novikova. Attack Modelling and Security Evaluation for Security Information and Event Management. In P. Samarati, W. Lou, and J. Zhou, editors, *SECRYPT*, pages 391–394. SciTePress, 2012.

[14] Lieberman Software. Common event format configuration guide, Jan. 2010.

[15] K. Lundholm, J. Hallberg, H. Granlund, and T. forskningsinstitut. Informationssystem. *Design and Use of Information Security Metrics: Application of the ISO/IEC 27004 Standard*. FOI-R. Information systems, Swedish Defence Research Agency, 2011.

[16] M. Melik-Merkumians, T. Moser, A. Schatten, A. Zoitl, and S. Biffl. Knowledge-based runtime failure detection for industrial automation systems. In *Workshop Models@run.time*, pages 108–119. CEUR, 2010.

[17] PreludeIDS Technologies. Prelude PRO 1.0. http://www.prelude-technologies.com/, 2010.

[18] E. Prieto, R. Diaz, L. Romano, R. Rieke, and M. Achemlal. Massif: A promising solution to enhance olympic games it security. In C. K. Georgiadis et al., editors, *Global Security, Safety and Sustainability & e-Democracy*, volume 99 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 139–147. Springer, 2012.

[19] N. Racz, E. Weippl, and A. Seufert. A Frame of Reference for Research of Integrated Governance, Risk and Compliance (GRC). In B. D. Decker and I. Schaumüller-Bichl, editors, *Communications and Multimedia Security*, volume 6109 of *Lecture Notes in Computer Science*, pages 106–117. Springer, 2010.

[20] J. Repp and R. Rieke. Formal Specification of Security Properties. Deliverable D4.2.1, MASSIF Project, September 2011. http://www.massif-project.eu/sites/default/files/deliverables/D4.2.1%20-%20Formal%20Speci%EF%AC%81cation%20of%20Security_v1.0_final.pdf.

[21] J. Schiefer, S. Rozsnyai, C. Rauscher, and G. Saurer. Event-driven rules for sensing and responding to business situations. In H.-A. Jacobsen, G. Mühl, and M. A. Jaeger, editors, *DEBS*, volume 233 of *ACM International Conference Proceeding Series*, pages 198–205. ACM, 2007.

[22] J. Schütte. Goal-based policies for self-protecting systems. In *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA)*. IEEE Computer Society, 2012.

[23] J. Schütte, R. Rieke, and T. Winkelvos. Model-based security event management. In *International Conference "Mathematical Methods, Models and Architectures for Computer Networks Security" (MMM-ACNS-12)*. Springer, 2012.

[24] F. Skopik, Z. Ma, P. Smith, and T. Bleier. Designing a cyber attack information system for national situational awareness. In N. Aschenbruck, P. Martini, M. Meier, and J. Tölle, editors, *Future Security*, volume 318 of *Communications in Computer and Information Science*, pages 277–288. Springer, 2012.

[25] N. Stakhanova, S. Basu, and J. Wong. A taxonomy of intrusion response systems. *Int. J. Inf. Comput. Secur.*, 1(1/2):169–184, Jan. 2007.

# P16

## MASSIF: A PROMISING SOLUTION TO ENHANCE OLYMPIC GAMES IT SECURITY

| | |
|---|---|
| **Title** | MASSIF: A Promising Solution to Enhance Olympic Games IT Security |
| **Authors** | Elsa Prieto, Rodrigo Diaz, Luigi Romano, Roland Rieke, and Mohammed Achemlal |
| **Publication** | In Christos K. Georgiadis et al., editors, *Global Security, Safety and Sustainability & e-Democracy*, pages 139–147. |
| **ISBN/ISSN** | ISBN 978-3-642-33448-1 |
| **DOI** | `http://dx.doi.org//10.1007/978-3-642-33448-1_20` |
| **Status** | Published |
| **Publisher** | Springer Berlin Heidelberg |
| **Publication Type** | Book Chapter: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 99 |
| **Copyright** | 2012, Springer |
| **Contribution of Roland Rieke** | Co-Author with significant contribution with respect to specific challenges and overall MASSIF concepts.<br>Related contributions: (a) presentation of the topic titled "Advanced Security Monitoring: Challenges, Advances, and Foundations - The MASSIF project" at the the Cyber Security & Privacy EU Forum 2012 [Rieke, 2012a], and (b) invited talk titled "Enhancing Situational Awareness, Security and Trustworthiness of Processes in Systems of Systems" at the Second International Workshop 'Scientific Analysis and Policy Support for Cyber Security' [Rieke, 2012b]. |

Table 21: Fact Sheet Publication *P16*

Publication P16 [Prieto, Diaz, Romano, Rieke & Achemlal, 2012] addresses the following research question:

RQ12A *Can the developed methods and tools be successfully adapted to large scale industrial scenarios?*

This paper addresses the security management challenges that arise in the cyber-security of Olympic Games and how advanced SIEM can help to improve it. Nowadays, Olympic Games have become one of the most profitable global media events, becoming at the same way more and more attractive target from the terrorist perspective due to their media diffusion and international dimension. Critical for the success of such a highly visible event is protecting and securing the business and the supporting cyber-infrastructure enabling it. In this context, the MASSIF project aims to provide a new generation SIEM framework for service infrastructures supporting intelligent, scalable, and multilevel/multi-domain security event processing and predictive security monitoring.

# MASSIF: A Promising Solution to Enhance Olympic Games IT Security

Elsa Prieto[1], Rodrigo Diaz[1], Luigi Romano[2], Roland Rieke[3] and Mohammed Achemlal[4]

[1] Atos Research and Innovation (ARI), Atos Origin
{elsa.prieto, rodrigo.diaz}@atosresearch.eu
[2] Consorzio Interuniversitario Nazionale per l'Informatica (CINI)
{lrom}@uniparthenope.it
[3] Fraunhofer - Institut für Sichere (SIT)
{roland.rieke}@sit.fraunhofer.de
[4] Orange - France Telecom SA
{mohammed.achemlal}@orange-ftgroup.com

**Abstract:** Nowadays, Olympic Games have become one of the most profitable global media events, becoming at the same way more and more attractive target from the terrorist perspective due to their media diffusion and international dimension. Critical for the success of such a highly visible event is protecting and securing the business and the supporting cyber-infrastructure enabling it. In this context, the MASSIF project aims to provide a new generation SIEM framework for service infrastructures supporting intelligent, scalable, and multi-level/multi-domain security event processing and predictive security monitoring.

**Keywords:** Information Security Management, Security Event Management, Systems Safety, Data Security, Software Protection, Secure Architecture Design.

## 1 Introduction

Recent terrorist attacks across the world indicated that terrorists continue to target crowded places and show how vulnerable high profile venues and events can be used to perpetrate such incidents for maximum impact across the globe.

Terrorism attacks can adopt many forms, not just a physical attack on life and limb. It can include interference with vital information or communication systems, causing disruption and economic damage. For this reason, in addition to the physical security of the event venues, the cyber-security of the IT event infrastructure should be protected in the same way.

Nowadays, Olympic Games have become one of the most profitable global media events. From the terrorist perspective, Olympics can be seen as one of the most attractive events to commit actions due to their media diffusion, international dimension and symbolic representation. As a consequence of this, security has become a top focus and budget priority. Surpassing all before it in size and scope, security at the Vancouver Olympic Games of 2010 cost an estimated USD $1 billion and included a 15,000-person force of Canadian military, Vancouver police, U.S. security forces, and private contractors to guard the city by air, land, and sea[1]. Vancouver marked a transition into an unparalleled era of Olympic security in terms of cross-national cooperation, planning, and spending. The scope, however, was limited—the majority of funds and efforts aimed to maintain calm during the two-week event and did not address longer-term security concerns.

As the Worldwide IT Partner for the Olympic Games and Top sponsor, Atos Origin integrates, manages and secures the vast IT system that relays results, events and athlete information to spectators and media around the world. The Atos Origin contract with the International Olympic Committee (IOC) is the world's largest sports related IT contract and was recently extended to cover the Sochi Olympic Winter Games in 2014 in Russia and the Rio Olympic Summer Games in 2016 in Brazil. The major challenge is to create an IT solution for each Olympic Games that allows the capture and reporting of every moment of the action and brings it to the world via television and the Internet. Critical for the success of such a highly visible event is protecting and securing the business and the supporting cyber-infrastructure enabling it and naturally, security is a top priority.

With innovation as the cornerstone of its business and strategy, Atos Origin is coordinating the European research project MASSIF (MAnagement of Security information and events in Service Infrastructures, http://www.massif-project.eu/). The MASSIF Consortium consists of 12 project partners from 6 different European countries (France, Germany, Italy, Portugal, Russia, Spain) and South Africa including three different groups of business roles: scenario providers (Atos Origin, Epsilon, France Telecom and T-Systems), scientific partners (Fraunhofer SIT, Institut Telecom, SPIIRAS, C.I.N.I., Universidad Politécnica de Madrid and Universidade de Lisboa) and SIEM developers (Alienvault and 6cure). This paper addresses the challenges that arise in the cyber-security of Olympic Games and how the results of the MASSIF project can help to improve it.

The paper is structured as follows: Section 2 provides an overview of the existing IT infrastructure while Section 3 includes the challenges addressed by the MASSIF project. Section 4 gives an overview of the related work. Finally, Section 5 concludes this paper and provides pointers for future work.

## 2 Olympic Games IT Infrastructure

Olympic Games are getting more and more huge events, numbers in this context are gigantic. For instance, in the Beijing games 10.708 athletes were competing, 5.600 written press & photographers were accredited, 12.000 rights holding broadcaster staff, 70.000 volunteers, more than 60 competition and non-competition venues (http://en.beijing2008.cn/media/usefulinfo/).

The Olympic Games, must successfully issue and activate more than 200,000 accreditations for Games that comprise around 300 events representing over 4,500 hours of live competition. Live commentator services are delivered for around 20 sports. More than 15 million information pages are viewed, with peaks of 1 million pages viewed on specific days. Over 3Gb of live results are provided in around 800,000 messages to the Olympic website, broadcasters and sports federations.

The complex, massive IT infrastructure of the Olympic Games is deployed by large teams of people into different environments every other year. Such a major task could potentially pose significant risks, but these can be offset through preserving and sharing the knowledge gained from previous Games.

The Olympic Games have 3 core systems that support the operations of the Games. These systems are summarised below:

**Core Games System (CGS).** CGS is a set of applications for assisting in the capture and management of data about people who will be attending the Games events and the staff supporting them. Among others, this includes Accreditation and Workforce management (including Volunteers)

**Information Diffusion System (INFO).** INFO comprises of a set of applications that retrieve and distribute information related to, and supporting, of the Games. The information is provided by different sources e.g. Results system, interfaces with CGS, Weather provider etc. The information is processed and distributed to internal clients e.g. broadcasters, journalists, and other members of the Olympic and Paralympics Families. IT is also sent to external clients e.g. World News Press Agencies (WNPA), sports federations and governing bodies, and Internet Service Providers (ISPs).

**Results Systems,** are grouped into two sets of systems:

**Timing & Scoring Systems (T&S)** capture real-time data during the competition. Through electronic feeds to other systems, this data is made available for use on the scoreboard, in TV graphics and other related outputs, by OVR.

**On Venue Results Systems (OVR**) running at each of the competition venues receives both data from T&S and manually entered data to calculate results of each Olympic event. OVR Systems then distribute the results to INFO.

Concerning the security of the IT infrastructure, for the Beijing 2008 Olympic Games, more than 12 million IT security events were collected and filtered events each day to detect any potential security risk for the Olympic Games IT systems. From these, less than 100 were identified as real issues. All were resolved, with no impact at all on the Olympic Games (http://www.atosorigin.com/olympic_games).

For an event of this magnitude, deadlines are not negotiable, when world-class athletes are ready to compete for gold after years of rigorous training and qualification and viewers are anxious to enjoy such a show, there are no second chances. System disruption or failure is not acceptable. In this context, the main challenge of the SIEM infrastructure in Olympic Games is to protect the games IT infrastructure from any undesired and/or uncontrolled phenomena which can impact any part of the result chain and associated services.

## 3 Security Management Challenges

Security Information Event Management (SIEM) solutions have become the backbone of the all Service Security systems. They collect data on events from different security elements, such as sensors, firewalls, routers or servers, analyze the data, and provide a suitable response to threats and attacks based on predefined security rules and policies. Despite the existence of highly regarded commercial products, their technical capabilities show a number of constraints in terms of scalability, resilience and interoperability.

The MASSIF project aims at achieving a significant advance in the area of SIEMs by integrating and relating events from different system layers and various domains into one more comprehensive view of security-aware processes and by increasing the scalability of the underlying event processing technology. The main challenge that MASSIF will face is to bring its enhancements and extensions into the business layer with a minimum impact on the end-user operation.

A further goal of the MASSIF project is to integrate these results in two existing Open Source SIEM solutions, namely OSSIM (http://alienvault.com/community) and Prelude (http://www.prelude-technologies.com/) and to apply them to four industrial scenarios, including the Olympic Games IT infrastructure.

Aligned with the security needs of these scenarios, MASSIF challenges can be arranged according to the following dimensions:

### 3.1 Collection

The data gathering must have the ability to deal with a large number of highly heterogeneous data feeds. The capabilities of the SIEM will be improved by the integration of new types of security tools/probes. This implies that the parsing/processing logic (and code) should be as much as possible decoupled from the specific characteristics of the data format and related technologies. Additionally, the parsing logic and related languages must allow effective processing of virtually any type of security relevant event in cyber-environment, including, in the future, possible extensions to capture and process security events from physical security equipment.

Moreover, the volume of events to be collected and processed per unit of time can occasionally increase resulting in load peaks. The data collection layer should be able to handle such peaks and to propagate relevant events to the SIEM core platform without loss of information.

These concepts are implemented in MASSIF by the Generic Event Translation (GET) framework. The GET framework relies on grammar-based parsing [2], [3] and compiler-compiler technology to implement effective processing of security-relevant events. The main components of the Generic Event Translation Framework are represented in Figure 1. A brief description of each component is provided in the following:

**Generic Event Translation (GET) Manager.** This component is responsible for the activation of all the modules which belong to the Generic Event Translation framework. In particular, it is in charge of the generation of new Adaptable Parser modules, as new grammars are added to the system.

**Event Dispatcher.** This component connects each source of sensor events to the appropriate GET Access Point (GAP), in order to provide it with an Adaptable Parser which is capable of processing the specific event format.

**GET Access Point (GAP).** It is responsible for orchestrating the translation process of the GET. It is in charge of extracting the content of source messages in the source specific format, using the event parsing capabilities of the Adaptable Parsers and requesting the final conversion to the MASSIF Event Format by the MASSIF Event Manager (MEM).

**Format-specific Grammars.** These contain semantic description of the different event formats that are used for the creation of the Adaptable Parsers.

**Adaptable Parsers.** These components provide the parsing capabilities for the different types of events used in MASSIF. They allow for extraction of the relevant information for the event to be inserted in the MASSIF Event Format.

**MASSIF Event Manager (MEM).** It translates the event content, extracted by the Adaptable Parser to the MASSIF Event Format, thus allowing the event to be sent to the reliable event bus. It also attaches to each MASSIF Event a timestamp, which is made available by the synchronized time source of the Resilient Architecture.

**Sender Agent**. It is the component that finally sends MASSIF-formatted events to the reliable event bus.

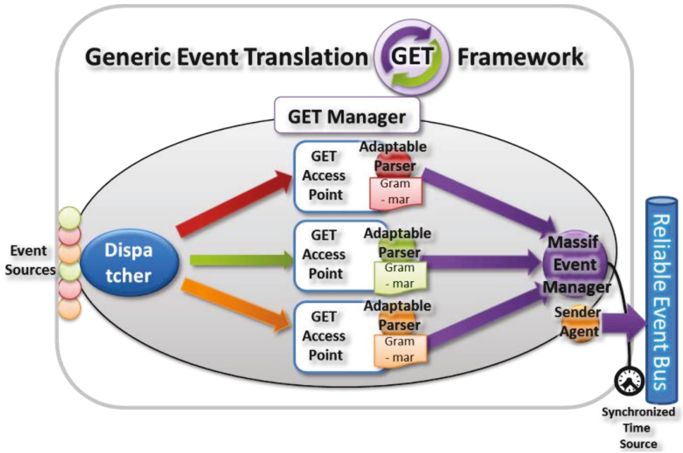


**Figure 1: Generic Event Translation main components**

### 3.2 Processing

The core of MASSIF is an event processing engine capable of handling high input rates and of optimizing the amount of resources based on the actual load [4]. In other words, the system should monitor both input loads and vital parameters, such as CPU utilization, in order to adjust the amount of resources, i.e., provision more resources during peak load times and decommission them during valley load periods. The system must process input data at high rate and provide meaningful results with soft real-time requirements. The engine should be able to aggregate, abstract and correlate heterogeneous events from multiple sources at different levels of the system stack.

### 3.3 Correlation

MASSIF targets at correlation capabilities across layers of security events, from network and security devices as well as from the service infrastructure such as correlation of physical and logical event sources. The engine should be shipped with a set of predefined correlation rules to identify well-known attacks. However, it should also support easy and intuitive creation of user-defined rules.

### 3.4 Resilience

Special emphasis will be placed on providing a highly resilient architecture against attacks, concurrent component failures, and unpredictable network operation conditions. The event flows should be protected, from the collection points through their distribution, processing and archival. The designed mechanisms should offer flexible and incremental solutions for node resilience, providing for seamless deployment of necessary functions and protocols. These mechanisms should take into consideration particular aspects of the infrastructure, such as edge-side and core-side node implementations.

### 3.5 Timeliness

The infrastructure should provide for (near) real-time collection, transmission and processing of events, and ensure the corresponding reliable and timeliness generation of alarms and countermeasures when needed.

### 3.6 Sensitive information

MASSIF features for forensic support will satisfy the following requirements:

**Data authenticity**. Security event data contents, as well as additional/added information related to data origin and destination, must be the reliably stored.

**Fault and intrusion-tolerant stable storage**. The stable storage system on which data for forensic use will be persisted must be tolerant both to faults and to intrusions.

**Least persistence principle**. With respect to sensitive data, only information which is actually needed should be persisted to stable storage (most of the data should be processed in real-time and thrown away).

**Privacy of forensic records**. Forensic evidence related to security breaches will be made available only to authorized parties.

## 4 Related Work

The research in MASSIF combines aspects of process monitoring, simulation, and analysis as well as trustworthiness and scalability of the complex event processing architecture itself. Relevant contributions from these broad areas are:

**Attack modelling, simulation and risk evaluation.** The technology most relevant to the modelling and simulation methods to be developed for MASSIF is commonly called attack-graph analysis, an approach presented by Phillips and Swiler [5] in. Two participants of the MASSIF team namely Fraunhofer SIT and SPIIRAS are actively researching in that area [6], [7].

**Predictive Security Analysis.** The predictive security analysis in MASSIF will use the method given in [8] to analyse the security requirements. Based on this, the attack models together with the SIEM's information about the current attack state and the process models together with the SIEM's information about the current process state can be used to derive a near future view of possible upcoming security problems [9]. This information can now be used in an ontology-driven approach to select appropriate countermeasures [10].

**SIEM Scalability and Trustworthiness.** Complex Event Processing (CEP) is a promising technology to improve current SIEM systems. It allows processing of large amounts of streaming data in real time and provides information abstraction and correlation, similarly to SIEM correlation engines. MASSIF will develop new parallel distributed CEP technology that overcomes scalability limitations due to single node bottlenecks or high distribution overhead [4]. The trustworthiness of the SIEM architecture will be improved by utilising secure digital chains of evidence [11].

## 5 Conclusions and Outlook

The MASSIF project is still at an early stage. However, the challenges that the project aims to achieve will provide a significant advance in the area of Security Information and Event Management (SIEM) by integrating and relating events from different system layers and various domains into one more comprehensive view of security-aware processes and by increasing the scalability of the underlying event processing technology. To address the challenges the MASSIF partners plan to develop a novel SIEM system with the following solutions and implied research and development needs.

In order to enable a highly scalable security situation assessment, the MASSIF event engine will provide a flexible language to express filtering, transformation, abstraction, aggregation, intra-layer and cross-layer correlation as well as storage of security events. The event engine will be able to process with the same language both the real-time event flow as well as stored events for forensic analysis. Additionally, specific collectors to translate from the external languages into the event engine language will be provided.

Ideally, the MASSIF system should be able to analyze upcoming security threats and violations in order to trigger remediation actions even before the occurrence of possible security incidences. Therefore, new process and attack analysis and simulation techniques will be developed in order to be able to relate events dynamically from different execution levels, define specific level abstractions, evaluate them with respect to security issues and during runtime interpret them in context of specific security properties. Novel adaptive response technologies will enable anticipatory impact analysis, decision support and support impact mitigation by adaptive configuration of countermeasures such as policies.

Due to the highly distributed and heterogeneous nature of the various components, and the hostile and unpredictable operational environment, it becomes a challenge to design an integrated solution for the protection of the SIEM framework itself. Therefore, the MASSIF system will be based on a resilient, trust-enabling architecture with trusted collection of security-relevant data from highly heterogeneous trusted networked devices in order to ensure unforgeability of stored security events and to support criminal/civil prosecution of attackers.

## Acknowledgements

The work in this paper has been sponsored by the EC Framework Programme as part of the ICT MASSIF project (grant agreement no. 257644).

## References

1. S. R. McRoskey. Security and the Olympic Games: Making Rio an Example. Yale Journal of International Affairs. 2010.
2. Turmo, J., Ageno, A., and Catala, N.: Adaptive Information Extraction. ACM Computing Surveys, vol. 38, no. 2, 2006
3. F. Campanile, A. Cilardo, L. Coppolino, L. Romano: Adaptable Parsing of Real-Time Data Streams. In proc. of The Fifteen Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP07), February 7-9, 2007, Naples, Italy, pp. 412-418, IEEE Computer Society Press, Los Alamitos, CA (USA).
4. V. Gulisano, R. Jimenez-Peris, M. Patiño-Martínez, P. Valduriez: A Large Scale Data Streaming System. 30th IEEE Int. Conf. on Distributed Systems (ICDCS), Genoa, Italy, 2010.
5. Cynthia A. Phillips and Laura Painton Swiler: A graph-based system for network-vulnerability analysis. In NSPW '98, Proceedings of the 1998 Workshop on New Security Paradigms, pages 71-79. ACM Press, 1998.
6. Roland Rieke: Abstraction-based analysis of known and unknown vulnerabilities of critical information infrastructures. International Journal of System of Systems Engineering (IJSSE), 1:59-77, 2008.
7. I.Kotenko, M.Stepashkin, E.Doynikova: Security Analysis of Computer-aided Systems taking into account Social Engineering Attacks. 19th Euromicro International Conference on Parallel, Distributed and network-based Processing (PDP 2011). Ayia Napa, Cyprus.
8. Andreas Fuchs and Roland Rieke: Identification of Security Requirements in Systems of Systems by Functional Security Analysis. In Antonio Casimiro, Rogério de Lemos, and Cristina Gacek, editors, Architecting Dependable Systems VII, volume 6420 of Lecture Notes in Computer Science, pages 74-96. Springer, 2010.
9. Roland Rieke and Zaharina Stoynova: Predictive Security Analysis for Event-Driven Processes. In Igor Kotenko and Victor Skormin, editors, Computer Network Security, volume 6258 of LNCS, pages 321-328. Springer Berlin / Heidelberg, 2010.
10. N. Cuppens-Boulahia, F. Cuppens, J. Lopez, E. Vasquez, J. Guerra, and H. Debar: An ontology-based approach to react to network attacks. International Journal of Information and Computer Security, 3:280-305, 2009.
11. Nicolai Kuntze, Carsten Rudolph: Secure digital chains of evidence. Sixth International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE) 2011. Oakland (USA)

# P17

# SECURITY AND RELIABILITY REQUIREMENTS FOR ADVANCED SECURITY EVENT MANAGEMENT

| | |
|---|---|
| **Title** | Security and Reliability Requirements for Advanced Security Event Management |
| **Authors** | Roland Rieke, Luigi Coppolino, Andrew Hutchison, Elsa Prieto, and Chrystel Gaber |
| **Publication** | In Igor Kotenko and Victor Skormin, editors, *Computer Network Security – 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2012 St. Petersburg, Russia, October 2012, Proceedings*, volume 7531 of *Lecture Notes in Computer Science*, pages 171–180, 2012. |
| **ISBN/ISSN** | ISBN 978-3-642-33703-1 |
| **DOI** | `http://dx.doi.org/10.1007/978-3-642-33704-8_15` |
| **Status** | Published |
| **Publisher** | Springer Berlin Heidelberg |
| **Publication Type** | Conference Proceedings (LNCS, Vol. 7531) |
| **Copyright** | 2012, Springer |
| **Contribution of Roland Rieke** | Main Author and editor.<br>Related contributions: (a) main authorship of the description of work for the MASSIF project which provided the base for the guidelines for next generation SIEM described here, (b) presentation of this topic titled "Challenges for Systems of Systems Security Information and Event Management" at the 2010 Workshop on Cyber Security and Global Affairs [Rieke, 2010a], and (c) joined talk with Co-Author Andrew Hutchison titled "Management of Security Information and Events in Future Internet" at the 2011 Workshop on Cyber Security and Global Affairs [Hutchison & Rieke, 2011]. |

Table 22: Fact Sheet Publication *P17*

Publication P17 [Rieke, Coppolino, Hutchison, Prieto & Gaber, 2012] addresses the following research question:

RQ12a *Can the developed methods and tools be successfully adapted to large scale industrial scenarios?*

This paper addresses security information management in complex application scenarios. SIEM systems collect and examine security related events, with the goal of providing a unified view of the monitored systems' security status. While various SIEMs are in production, there is scope to extend the capability and resilience of these systems. The use of SIEM technology in four disparate scenario areas is used in this paper as a catalyst for the development and articulation of Security and Reliability requirements for advanced security event management. The scenarios relate to infrastructure management for a large real-time sporting event, a mobile money payment system, a managed services environment and a cyber-physical dam control system. The diversity of the scenarios enables elaboration of a comprehensive set of security and reliability requirements which can be used in the development of future SIEM systems.

# Security and Reliability Requirements for Advanced Security Event Management

Roland Rieke[1], Luigi Coppolino[2], Andrew Hutchison[3], Elsa Prieto[4], Chrystel Gaber[5]

[1] Fraunhofer Institute SIT, Darmstadt, Germany
[2] Epsilon S.r.l., Naples, Italy
[3] T-Systems, South Africa
[4] Atos Research & Innovation
[5] Orange Labs - France Telecom

**Abstract.** This paper addresses security information management in complex application scenarios. Security Information and Event Management (SIEM) systems collect and examine security related events, with the goal of providing a unified view of the monitored systems' security status. While various SIEMs are in production, there is scope to extend the capability and resilience of these systems. The use of SIEM technology in four disparate scenario areas is used in this paper as a catalyst for the development and articulation of Security and Reliability requirements for advanced security event management. The scenarios relate to infrastructure management for a large real-time sporting event, a mobile money payment system, a managed services environment and a cyber-physical dam control system. The diversity of the scenarios enables elaboration of a comprehensive set of Security and Reliability requirements which can be used in the development of future SIEM systems.

**Keywords:** security requirements, security information and event management, SIEM, architecting trustworthy systems

## 1 Introduction

Security information and event management (SIEM) systems provide important security services. They collect and analyse data from different sources, such as sensors, firewalls, routers or servers, and provide decision support based on anticipated impact analysis. This enables timeous response to (or prevention of) attacks as well as impact mitigation by adaptive configuration of countermeasures. However, there are also a number of constraints for current commercial solutions. These constraints include the inability of systems to consider events from a multiple organisations or the ability to provide high degree of trustworthiness or resilience in the event collection environment.

The project MASSIF [3], a large-scale integrating project co-funded by the European Commission, addresses these challenges with respect to four industrial domains: (i) the management of the Olympic Games information technology (IT) infrastructure [12]; (ii) a mobile phone based money transfer service, facing high-level threats such as money laundering; (iii) managed IT outsource services for large distributed enterprises; and (iv) an IT system supporting a critical infrastructure (dam) [4].

In undertaking the development of next-generation SIEM concepts and constructs, it became clear that the Security and Reliability of the SIEM itself are critical to the successful deployment of SIEM in a particular environment. With this in mind, we set about analysing each of the mentioned scenarios in some detail, to create an explicit list of Security and Reliability requirements. The intention is that these requirements can be used to guide and assess SIEM development, and ensure that these important attributes are incorporated.

## 2 Large Scale Scenarios in four Industrial Domains

In this section, four deployment scenarios for SIEM technology are introduced. The elements of the scenario which can benefit from further SIEM development are also outlined in each case. From the introduction of the scenarios and their unique characteristics, a set of consolidated requirements for a next-generation SIEM can be compiled.

### 2.1 Scenario 1: SIEM technologies used in The Olympic Games

The Olympic Games is one of the largest and most high profile sporting events that takes place, and there is a large technical infrastructure to support many aspects of the games both asynchronously and in real-time. SIEM infrastructure is used with the Olympic Games systems, to protect the games IT infrastructure from any undesired and/or uncontrolled phenomena which could impact any part of the result chain and associated services. The nature of this kind of event presents a big challenge to SIEM infrastructures, for example the next London 2012 games cater for 79 days of competition, 26 sports, 94 venues, 17.000 athletes, 20.000 journalists, 70.000 volunteers, 4.000 IT team members, 900 servers, 1,000 network and security devices and more than 10,000 computers deployed. One of the new challenges will be the amount of data generated from the results systems, representing 30% more than in the Beijing Olympics in order to provide real-time information to fans, commentators and broadcasters world wide. The intensity and complexity of this kind of sporting event presents a big challenge to SIEM infrastructure, mainly, due to two very characteristic features: the number of security event types (about 20,000), and the volume of generated events to be handled (around 11,000,000 alerts per day). However, the most critical aspect that a SIEM system faces in the Olympic Games is that those security events must be processed and reacted upon in real-time.

**Advanced SIEM system contribution.** The Olympic Games scenario is valuable to demonstrate the enhancement on scalability, processing enormous amounts of generated data events in real-time. Furthermore the scenario can contribute to validate the cross-layer correlation of events (service, application, infrastructure) from multiple sources.

### 2.2 Scenario 2: Mobile money transfer service

Use of Mobile Phones to effect payment is a widely used service, particularly in developing markets where banking systems may not be as dense or available as in developed countries. Characteristics and challenges of authentication, confidentiality, integrity and mobility all have to be considered in this scenario.

From a SIEM perspective, mobile money transfer is an interesting and challenging scenario for the unique attributes that the scenario presents. Indeed, this scenario is quite complex because it requires to analyze past and present data and to extract information from raw events. It is also very sensitive to the performance of detection as the rate of false positives and true negatives should be optimised. Finally, all this should be done while keeping the service scalable and secure. The service allows end users to convert cash to "electronic money" (and vice versa) at merchants, who act as distributors and act as channel users. The electronic money can be used to pay purchases at the merchants' or for bills such as electricity. Furthermore the electronic money can also be transferred between the end users. End users access the service with their mobile phones and distributors can access the service either via mobile phone or directly on the Internet. Both means of access are handled by front-end servers that then access the back-end servers containing the transactions etc.

**Advanced SIEM system contribution.** Like any other money transfer service, the service is exposed to the risk of money laundering and other types of fraud. The money laundering risk implies misuse through disguising illegally obtained funds to make them seem legal, and more generally the fraud risk implies any intentional deception made for financial gain. In addition, any money transfer service that has part of its infrastructure exposed via the Internet and/or the end user can access the service using electronic means (a mobile device such as a phone or a pad in this use case), has an increased exposure to fraud, via both attacks against the service infrastructure itself and the abuse of normal service functionality. The objective of including this scenario is to achieve greater protection and transactional integration of SIEM protection through next generation SIEM services. The ultimate intention is to protect the money transfer service against fraud both by detection and application of relevant counter-measures.

### 2.3 Scenario 3: Managed Enterprise Service Infrastructures

The use of *managed services* by businesses is an increasingly used model, whereby elements of IT and infrastructure are "outsourced" to specialist service providers. In some instances, services are provided by an outsourcer via shared platforms, giving customers economies of scale. In other instances, managed services are performed by a provider on the infrastructure belonging to a customer. Mixed approaches are also possible, and an extrapolation of this can be viewed as occurring when such services are provided in a "cloud based" mode. Provision of Security Information and Event Management services for customers is a valuable complement to the management which an outsourcer or service provider can deliver. The purpose of including a managed enterprise service infrastructure scenario was to consider just such cases: where the services of large enterprises are managed, and a SIEM service is used to collect, inspect and react to large scale security events from member systems and devices.

**Advanced SIEM system contribution.** There are a number of limitations of SIEM systems, encountered by managed security service providers, that are not adequately addressed by current SIEM solutions. For this reason, such a SIEM deployment is interesting to consider when looking at next-generation SIEM requirements. Some of the issues that can be identified in particular are: (i) insufficient resilience of the SIEM infrastructure itself to withstand large scale attacks; (ii) inadequate trustworthiness of source data within the SIEM; and (iii) inadequate disaster recovery capabilities of SIEM systems. Solutions to the limitations that current SIEM systems present will improve the resilience and business continuity capabilities of large companies, through enabling managed service providers to detect and address security events more proactively. It is considered that work on next-generation SIEM systems could address some of the identified problems through the following focus areas:

1. Providing guidelines on the minimum requirements for event data to enable successful event correlation.
2. Providing guidelines on the impact of the unavailability of certain event data on successful event correlation and management.
3. Guaranteeing the trustworthiness of event sources.
4. Improving correlation modelling for better analysis of complex environments (and for better automated correlation processing in complex environments).
5. Improving the resilience and business continuity capabilities for large enterprises.

### 2.4 Scenario 4: Critical Infrastructure Process Control (Dam)

The features of dam infrastructures are strictly related to the aims they are conceived for. Dams are mostly used for water supply, hydroelectric power generation, irrigation, water activities and wildlife habitat granting. Dams represent fundamental assets for the economy and the safety of a country, such as they are counted among critical infrastructures. So, monitoring of a dam is essential since an accident would have dramatic consequences. The amount of parameters to be monitored to assess the safety of a dam and foresee possible failures or anomalies is enormous, and this huge data flow must be analyzed under real time constraints. Each of these parameters is measured using different sensors, such as inclinometers and tiltmeters, crackmeters, jointmeters, earth pressure cells, turbidimeters, and thermometers. In addition to the above mentioned parameters measured by the sensors, other components are necessary for the full control of dam. Some essential elements are: data collectors, human machine interaction interfaces, data storing units, command and data gateways and signal buses. In other cases there is the need also to integrate different subsystems existing.

**Advanced SIEM system contribution.** The current SIEM solutions hardly facilitate the introduction of new technologies to improve the efficiency of the security event detection. At the same time, they usually lack in the capability to support heterogeneous systems and technologies. Introducing SIEMs to jointly manage all different aspects related to the security in the monitoring of a dam can be a very powerful mechanism to increase the overall security of such critical infrastructures. However, currently available SIEMs solutions are focused on the management of digital and information security related events and are designed specifically for this type of applications. This may make complex or even impossible the development of applications targeting security of critical infrastructures in a wider sense. For instance, creating an application capable of correlating network and host events that may indicate a cyber-attack with suspicious activities detected by the dam surveillance system may greatly improve the security of the whole monitoring process but may introduce some implementation difficulties. SIEMs are not designed to deal with this kind of scenarios and so, encompassing security events coming from different application domains within the same application may be troublesome. In particular, the current technologies usually neglect the possibility to correlate physical and logical events, which can improve the effectiveness of the detection process.

In order to secure the dam control system, today recognized as a critical infrastructure and hence of public interest, regulations must be considered. Indeed, any activity of the dam operators strictly follows well-known rigid procedures. For example, the opening of a gate without alerting the control center is not admitted. Unfortunately, the current SIEM technologies insufficiently exploit regularities characterizing dam systems. In particular, procedures could be encoded in patterns and they could be exploited for detecting anomalies in control system. All this information could contribute to make the security system aware of the context in order to correctly interpret the meaning of some evidences. Introducing such features in a SIEM solution moves the focus of the analysis from a system level view to the business process model of the system.

## 3 Consolidated Guidelines for Next Generation SIEM

Based on the four scenarios described, and the diverse set of circumstances that they cover between them, a set of consolidated recommendations, to guide the design and development of next generation SIEM platforms, is identified and grouped in five topics.

### 3.1 Guidelines Concerning Advanced Security Services

Besides issues like dependability, redundancy and fault tolerance, analysis of the four scenarios considered reveals the need for enhanced *security-related* features of future SIEM platforms. These features go beyond what is currently supported by existing solutions. Overall a lack of capability to model incidents at an abstract level is perceived. From the scenarios investigated, and the current SIEM limitations observed, the following guidelines have been identified for next-generation SIEMs with respect to security:

**Correlation across layers of security events.** Advanced SIEM systems needs to support enhanced correlation across layers, from network and security devices as well as from the service infrastructure such as correlation of physical and logical event sources. This is due to the variety of systems issuing inputs that can give insights to security only when combined. An example is the off-site monitoring and the on-site management of the dam's configuration.

**Multi-level security event modelling.** Multi-level security event modelling will enable provision of more holistic solutions to protect the respective infrastructures. The Olympic Games Scenario stipulates that it would be of interest to understand the effects of technical events on the user or process level of the system.

**Analysis of malicious behaviour using attack graphs.** Many of the security issues mentioned in this document originate from complex malicious actions or patterns of actions (e.g., the laundering of money in the mobile money transfer scenario or the misuse case of *Low and Slow* attacks in the Olympic Games infrastructure).

**Predictive security monitoring.** Predictive security monitoring allows to counter negative future actions, proactively. There is a crucial demand for early warning capabilities. Moreover, the limitations with regards to the Managed Enterprise Service point to the fact that dealing with unknown or unpredictable behaviour patterns is not sufficient in current SIEM solutions.

**Modeling of the events and their relation to other, possibly external, knowledge.** A basic precondition of prediction and simulation as well as of attack analysis is the proper representation of the security requirements and any relevant information about the system as well as any knowledge about the actual and possible behaviour. When reasoning under incomplete information it is not only decisive to properly gather and describe the information available, but it is also required to develop novel methods based on discernibility, probability or plausibility in order to reason about uncertainty.

**Securing the evidence progressed by the SIEM components.** The misuse case of a sensor compromise, showing that it is vital to be able to trust the information that is received, when using events from sensors like those deployed to monitor the dam or other critical infrastructures.

### 3.2 Guidelines Concerning Event Processing

Similar to the limitations noted for security, recommendations for *event processing* are also made, based on limitations in current SIEM implementations. The guidelines for a next generation event correlation engine are as follows:

**Real-time.** The system must process input data at a high rate and provide meaningful results with soft real-time requirements.

**Scalability and elasticity.** The engine should be capable of handling high input rate and should optimize the quantity of resources required based on the actual load. In other words, the system should monitor both input loads and vital parameters, such as CPU utilization, in order to adjust the amount of resources, i.e., provision more resources during peak load times and decommission them during valley load periods.

**Handling streaming and stored data.** The engine should allow processing and correlation both of streams of events and stored relations (i.e., information stored in a database).

**Multiple-sources.** The engine should be able to aggregate, abstract and correlate heterogeneous events from multiple sources at different levels of the system stack.

**Pre-defined correlation rules and rule augmentation capability.** The engine should be shipped with a set of predefined correlation rules to identify well-known attacks. However, it should also support easy and intuitive creation of user-defined rules.

### 3.3 Guidelines Concerning Advanced SIEM Trustworthiness

*Trustworthiness* is the ability to provide a service in a way it is expected in terms of safety, security, reliability, availability, and timeliness. The analysis of the input scenarios has resulted in the following guidelines, to improve the general resilience and trustworthiness aspects of a next generation SIEM:

**Resilience of the infrastructure.** The infrastructure should be highly resilient under attack, concurrent component failures, and unpredictable network operation conditions.

**Security of event flows.** The event flows should be protected, from the collection points through their distribution, processing and archival.

**Protection of the nodes.** The designed mechanisms should offer flexible and incremental solutions for node resilience, providing for seamless deployment of necessary functions and protocols. These mechanisms should take into consideration particular aspects of the infrastructure, such as edge-side and core-side node implementations.

**Timeliness of the infrastructure.** The infrastructure should provide for (near) real-time collection, transmission and processing of events, and ensure the corresponding reliable and timeliness generation of alarms and countermeasures when needed. Similarly, features for forensic support should adhere to the following guidelines:

**Data authenticity.** Security event data contents, as well as additional/added information related to data origin and destination, must be the reliably stored.

**Fault and intrusion-tolerant stable storage.** The stable storage system on which data for forensic use will persist must be tolerant both to faults and to intrusions.

**Least persistence principle.** With respect to sensitive data, only information which is actually needed should be retained to stable storage (much of the data could be processed in real-time and potentially discarded).

**Privacy of forensic records.** Forensic evidence related to security breaches should be made available only to authorized parties.

### 3.4 Guidelines Concerning Compiler Technologies

In terms of *data acquisition* functionality, it has been noted that next generation SIEM systems should exhibit efficient implementation and/or support for various Features relating to data collection and parsing. Specific guidelines are as follows:

**Heterogeneity support.** The data acquisition element must have the ability to deal with a large number of highly heterogeneous data feeds.

**High degree of adaptability.** Seamless integration of new types of security tools/probes should be possible, to improve the capabilities of the SIEM on an ongoing basis.

**Peak handling.** The volume of events, to be collected and processed per unit of time, can occasionally increase, resulting in load peaks. The data collection layer should be able to handle such peaks and propagate relevant events to the SIEM core platform without loss of information.

**High degree of expressiveness.** The parsing logic, and related Languages, must allow effective processing of virtually any type of security relevant event.

**Support for fast and reliable development.** Simple Development and configuration techniques and tools must be available. These will make it possible to implement, deploy, and integrate new parsers and collectors in a relatively short time and at a relatively low cost.

**Generality and platform independence.** The parsing/processing logic (and code) should as far as possible be decoupled from the specific characteristics of the data format and related technologies.

**Distributed processing.** Whenever possible (and feasible), the data collection and parsing layer should implement parsing, filtering, and correlation functions at the edges and/or at intermediate nodes, i.e. nodes located along the path to the core SIEM correlation engine.

### 3.5 Guidelines Concerning Legal Aspects

In terms of *legal* considerations, SIEM systems themselves need to be viewed as data processing entities with consideration being given to issues like data retention, data privacy and so on. From the scenarios considered, the following guidelines in terms of legal aspects have been identified:

**Data Retention.** Data must be retained for a period of time not more than that necessary to the activities for which they were collected. If the data are required for detection and suppression of crime they can be stored for a longer period of time.

**Cross-Border Data Transmission.** It must be possible to limit the transmission of data outside of certain borders. It should be possible to process data within such a border. If personal data must be transferred to another country, it must be ensured that the level of data protection in the country of destination is adequate.

**Minimum and Appropriate Security Measures.** Considering state of the art technology, a minimum (but sufficient) set of measures must be taken to preserve integrity, confidentiality, and availability of personal data. More sensitive data require increased security measures.

**Data Minimization and Anonymization.** Only data strictly needed for security guarantee must be kept, while unnecessary details must be deleted or made anonymous.

## 4 Related Work

The development of new security relevant systems requires the integration of a security engineering process in the earliest stages of the development life-cycle. This is specifically important in the development of systems, where security is the enabling technology, as in advanced SIEM systems. There are several common approaches to security requirements engineering that may be taken. An overview of such processes is given in [5] and also in [9]. A comprehensive concept for an overall security requirements engineering process is described in detail in [8]. The authors propose a 9 step approach called SQUARE (Security Quality Engineering Methodology). A similar approach based on the integration of Common Criteria (ISO/IEC 15408) called SREP (Security Requirements Engineering Process) is described in [10]. In [6], different kinds of security requirements are identified and informal guidelines are listed that have proven useful when eliciting concrete security requirements. The author emphasises that there has to be a clear distinction between security requirements and security mechanisms. In [7], Hatebur et al. describe a security engineering process based on security problem frames and concretised security problem frames. The two kinds of frames constitute patterns for analysing security problems and associated solution approaches. [7] specifically addresses accountability by logging.

Though all of the above mentioned approaches may lead to a sufficient level of security for the designed architecture, there is no obvious means by which they can be compared regarding the security requirements that they fulfil. In this paper, we address the first step in every security engineering procces, namely the identification of artifacts, such as functional descriptions, dependencies and information flows, the identification of use cases and misuse cases, and stakeholders' information on assets, safety and security requirements. Additionally, we consider state-of-the-art information on existing SIEM systems and challenges identified by other work such as the following:

Security information and event management technology provides log management and compliance reporting as well as real-time monitoring and incident management for security events from networks, systems, and applications. A concise overview of current SIEM systems functionalities is presented in [11]. In [1], current threats are identified and advanced monitoring techniques such as file integrity monitoring, database activity monitoring, application monitoring, identity monitoring, and user activity monitoring are discussed. In [2], some challenges with respect to collecting and analyzing a multi-gigabit network stream are outlined. SIEM systems manage security events but are not primarily concerned with the trustworthiness of the event sources. Compared to traditional IT systems, securing SCADA systems (e.g., in the dam scenario) poses unique challenges. In order to understand these challenges and potential dangers, [13] provides a taxonomy of possible cyber attacks – including cyber-induced cyber-physical attacks on SCADA systems.

## 5 Conclusion and Future Work

This paper has described requirements in terms of security and reliability for advanced security information and event management. The approach used to identify requirements is scenario-driven: scenarios relating to a real-time, high profile sporting event infrastructure; a mobile payment system; an enterprise service provider deployment and a cyber-physical environment has been used as catalyst for requirements identification and elaboration.

Based on the key elements and attributes of each scenario, guidelines for security, event processing, trustworthiness, and compiler technologies in next-generation SIEM systems have been elaborated. To consolidate the approach, a conceptual model showing the progression from business process / application / infrastructure to elements of SIEM design and implementation has been introduced. It is considered to be quite unique and beneficial to have such a comprehensive and rigorous set of scenarios to draw upon, and studying and analysing the scenarios presented provides a sound foundation from which to make recommendations for next-generation SIEM systems.

We cannot necessarily claim that the set of recommendations is "complete", but by developing (and ultimately testing) the proposed items against such a diverse set of scenarios, there is a high probability of addressing a wide range of SIEM requirements. The benefit of multiple scenarios is that associated characteristics which include diverse requirements including mobility, scalability, real-time processing, potentially hostile device environments and so on. In this light, the security and reliability requirements are considered to be applicable to a wide range of advanced security event management contexts.

**Acknowledgements.** The authors developed this work in the context of the project MASSIF (ID 257475) being co-funded by the European Commission within FP7.

## References

1. Monitoring up the Stack: Adding Value to SIEM. White paper, Securosis L.L.C., Phoenix, AZ (2010), https://securosis.com/research/publication/monitoring-up-the-stack-adding-value-to-siem
2. Applied Network Security Analysis: Moving from Data to Information. White paper, Securosis L.L.C., Phoenix, AZ (2011), https://securosis.com/research/publication/applied-network-security-analysis-moving-from-data-to-information
3. Project MASSIF website (2012), http://www.massif-project.eu/
4. Coppolino, L., D'Antonio, S., Formicola, V., Romano, L.: Integration of a System for Critical Infrastructure Protection with the OSSIM SIEM Platform: A dam case study. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) SAFECOMP. Lecture Notes in Computer Science, vol. 6894, pp. 199–212. Springer (2011)
5. Fabian, B., Gürses, S., Heisel, M., Santen, T., Schmidt, H.: A comparison of security requirements engineering methods. Requirements engineering 15(1), 7–40 (2010)
6. Firesmith, D.: Engineering security requirements. Journal of Object Technology 2(1), 53–68 (2003)
7. Hatebur, D., Heisel, M., Schmidt, H.: Analysis and component-based realization of security requirements. In: Proceedings of the International Conference on Availability, Reliability and Security (AReS). pp. 195–203. IEEE Computer Society (2008), http://www.ieee.org/
8. Mead, N.R., Hough, E.D.: Security requirements engineering for software systems: Case studies in support of software engineering education. In: CSEET '06: Proceedings of the 19th Conference on Software Engineering Education & Training. pp. 149–158. IEEE Computer Society, Washington, DC, USA (2006)
9. Mellado, D., Blanco, C., SÃ¡nchez, L.E., FernÃ¡ndez-Medina, E.: A systematic review of security requirements engineering. Computer Standards & Interfaces 32(4), 153–165 (2010)
10. Mellado, D., Fernández-Medina, E., Piattini, M.: A common criteria based security requirements engineering process for the development of secure information systems. Comput. Stand. Interfaces 29(2), 244–253 (2007)
11. Nicolett, M., Kavanagh, K.M.: Magic Quadrant for Security Information and Event Management. Gartner Reasearch (May 2010)
12. Prieto, E., Diaz, R., Romano, L., Rieke, R., Achemlal, M.: MASSIF: A promising solution to enhance olympic games IT security. In: International Conference on Global Security, Safety and Sustainability (ICGS3 2011) (2011)
13. Zhu, B., Joseph, A., Sastry, S.: Taxonomy of Cyber Attacks on SCADA Systems. In: Proceedings of CPSCom 2011: The 4th IEEE International Conference on Cyber, Physical and Social Computing, Dalian, China (2011)

# P18

## FRAUD DETECTION IN MOBILE PAYMENT UTILIZING PROCESS BEHAVIOR ANALYSIS

| | |
|---|---|
| **Title** | Fraud Detection in Mobile Payment Utilizing Process Behavior Analysis |
| **Authors** | Roland Rieke, Maria Zhdanova, Jürgen Repp, Romain Giot, and Chystel Gaber |
| **Publication** | Proceedings of 2013 International Conference on Availability, Reliability and Security, ARES 2013, pages 662–669. |
| **ISBN/ISSN** | ISBN-13: 978-0-7695-5008-4 |
| **DOI** | `http://dx.doi.org/10.1109/ARES.2013.87` |
| **Status** | Published |
| **Publisher** | IEEE |
| **Publication Type** | Conference Proceedings |
| **Copyright** | 2013, IEEE |
| **Contribution of Roland Rieke** | Co-Author with significant contribution, editor, and presenter at the 2nd international workshop on Recent Advances in Security Information and Event Management (RaSIEM 2013).<br>Specific contributions are: (a) development of the overall concept of Predictive Security Analysis at Runtime (PSA@R), and (b) lead of the MASSIF workpackages which developed the Predictive Security Analyser (PSA).<br>Related contributions: Roland Rieke is Co-Chair of this workshop. He also presented the same topic in a joined talk with Co-Author Romain Giot titled "Predictive Security Analysis - Concepts, Implementation, first Results in Industrial Scenario" at the Cyber Security & Privacy EU Forum 2013 [Rieke & Giot, 2013]. |

Table 23: Fact Sheet Publication *P18*

Publication P18 [Rieke, Zhdanova, Repp, Giot & Gaber, 2013] addresses the following research questions:

RQ9B *How can operational process models be used for early detection of and reaction to deviations of process execution from its specification?*

This paper describes the control flow of *model learning* and *uncertainty reasoning* for *anomaly detection* in PSA@R. This is exemplified by detection of money laundering patterns in synthetic process behaviour composed of simulated logs based on properties captured from real world money transaction events.

In the initial learning phase, the normal behaviour pattern with regard to the transaction characteristics is learned by processing an event log without malicious content. A mapping to classify the transactions with regard to the amount of money transferred and the order of such abstract events in the event log is used for this purpose. This mapping is created empirically using real operational logs of the Mobile Money Transfer (MMT) system and can change if different training sets are used. An overview of the event processing steps of PSA@R in the learning phase is given.

In the anomaly detection phase PSA@R is used to identify deviations from the normal characteristics based on the values from a transaction monitor. In an experimental setup, the transaction monitor has been replaced by synthetic process behaviour composed of simulated logs based on properties captured from real logs.

RQ12A *Can the developed methods and tools be successfully adapted to large scale industrial scenarios?*

RQ12B *What are the performance effects of the number of events, processes, security requirements, predicted steps, and of event abstraction?*

In this paper the applicability of the PSA@R approach is exemplified by a MMT scenario. MMT systems are systems where electronic money is issued to different roles in order to perform various types of transactions. As with any payment system, this service can be an attractive target for attackers and fraudsters. For legal and service security issues it is mandatory to observe the transactions for potential abnormal activities. The work presented in this paper utilises alerts generated by the uncertainty reasoning component of the PSA prototype to detect money laundering patterns in synthetic process behaviour composed of simulated logs based on properties captured from real world transaction events. In particular, it is shown that the PSA is able to raise alerts in a simulated scenario of fraud with mules. For this simulated scenario, the detection is efficient, but show that such system could be sensitive to noise in a real world system. It would be necessary to improve the resistance to noise through a correlation of the generated alerts or by an application of specific evaluation of the process states when an alert is generated. The applicability of the proposed approach is evaluated and provides measurements on computational and recognition performance of the tool.

# Fraud Detection in Mobile Payments Utilizing Process Behavior Analysis

Roland Rieke[*][†], Maria Zhdanova[†], Jürgen Repp[†], Romain Giot[‡] and Chrystel Gaber[‡]
[*]Philipps-Universität Marburg, Germany
[†]Fraunhofer SIT, Darmstadt, Germany
Email:{roland.rieke,maria.zhdanova,juergen.repp}@sit.fraunhofer.de
[‡]France Télécom-Orange Labs,
Caen, France
Email:{romain.giot,chrystel.gaber}@orange.com

***Abstract*—Generally, fraud risk implies any intentional deception made for financial gain. In this paper, we consider this risk in the field of services which support transactions with electronic money. Specifically, we apply a tool for predictive security analysis at runtime which observes process behavior with respect to transactions within a money transfer service and tries to match it with expected behavior given by a process model. We analyze deviations from the given behavior specification for anomalies that indicate a possible misuse of the service related to money laundering activities. We evaluate the applicability of the proposed approach and provide measurements on computational and recognition performance of the tool – Predictive Security Analyzer – produced using real operational and simulated logs. The goal of the experiments is to detect misuse patterns reflecting a given money laundering scheme in synthetic process behavior based on properties captured from real world transaction events.**

***Keywords*—money laundering; predictive security analysis; analysis of business process behavior; security modeling and simulation; security monitoring.**

## I. Introduction

The field of Mobile Money Transfer (MMT) is a growing market segment, particularly in developing countries where banking systems may not be as dense or available as in developed countries. For example, M-Pesa, which was launched in 2007 in Kenya, displayed in December 2011 about 19 million subscribers, namely 70% of all mobile subscribers in Kenya [1]. Orange Money is deployed in 10 countries and gathers around 14% of the mobile subscribers of these countries [2]. In such services, transactions are made with electronic money, called mMoney. The users can convert cash to mMoney through distributors and use it to purchase goods at merchants, pay bills or transfer it to other users [3]. Like any other money transfer service, this service is exposed to the risk of money laundering, i.e., misuse through disguising illegally obtained funds to make them seem legal, and more generally fraud risk that implies any intentional deception made for financial gain [3].

In this paper, we apply a tool – the Predictive Security Analyzer (PSA) – that implements predictive security analysis at runtime [4], [5] in order to identify misuse patterns in event streams of MMT transactions that can be concerned with money laundering activity. Predictive security analysis at runtime is an advanced method for the evaluation of security-related events and their interpretation with respect to: (1) the known control-flow of the processes involved, and (2) the required security properties. With respect to (1), deviations of observed process behavior from the given process specification are analyzed. These deviations can be the result of an evolution in the process specification, problems with the measurement (e.g., lost events), or anomalies caused by attacker's interventions. Regarding (2), continuous monitoring of security properties specified for the process in question is performed to detect potential security violations.

The PSA takes as an input real-time events from the process execution environment, a process model and its security requirements. For informal modeling of MMT processes (at the business logic level) we employ the Event-driven Process Chain (EPC) language [6]. Security analysis of event-driven processes uses a formal process model encompassing the incoming events. If a critical state, i.e., a process anomaly or a security requirement violation, is detected the PSA provides a semi-automatic treatment by visualization and inspection of the problem (uncertainty management) or performs automatic generation and dissemination of alerts for further processing depending on the selected option. Due to the advanced security analysis method the PSA can enhance evaluation and correlation capabilities of Security Information and Event Management (SIEM) systems, as shown for MASSIF SIEM [7].

In this paper we evaluate the applicability of the approach and the performance of the PSA in this context. In particular, we show that: (1) the PSA is able to manipulate an event stream of operational systems in real time; (2) the PSA is able to raise alerts on most fraudulent transactions related to money laundering. The results of the experiments on real and simulated events for several money laundering scenarios will allow us to determine sensitivity and specificity of the PSA and refine the detection scheme.

The paper is organized as follows: Section II introduces the MMT scenario and a misuse case related to money laundering and Section III gives an overview of the PSA and its application in the experiment. Section IV introduces the experimental setup, while Section V discusses the results of the experiments. Section VI reviews related work and Section VII presents conclusions and directions for further research.

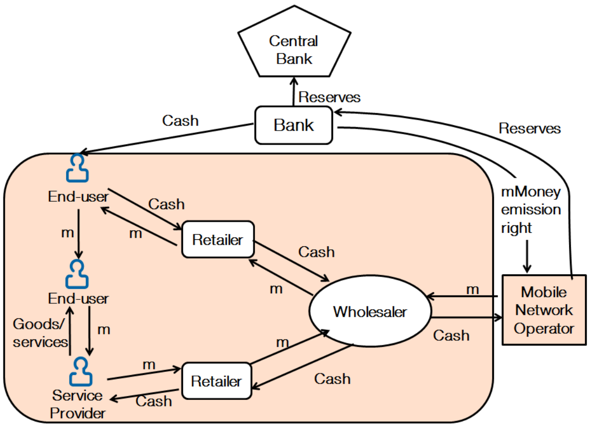


Figure 1. Economical environment of Mobile Money Transfer services

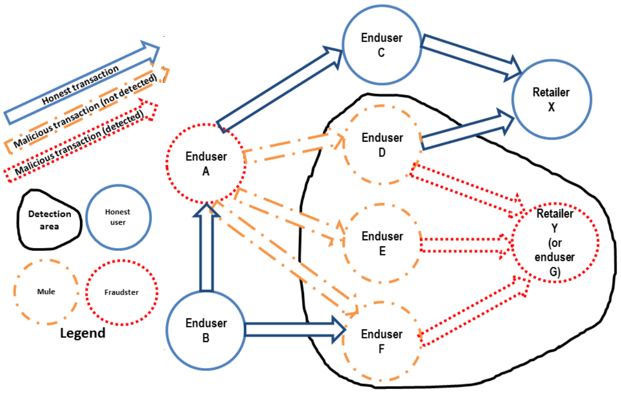


Figure 2. Description of the money laundering scenario: several mules receive mMoney from a malicious user and transfer a high percentage of this mMoney to another malicious user. The first malicious user does not want to be directly linked with the second malicious user

## II. Fraud Detection in Mobile Money Transfer System

### A. Mobile Money Transfer

This article is based on the MMT use case detailed in [8]. This section sums up the major points to understand the use case. MMT systems are systems where electronic money, called mMoney (or $m$), is issued to different roles (e.g., regular users, retailer, merchant, etc), in order to perform various types of transactions (e.g., mobile recharge, cash in, cash out, national or international transfer, bill payments, etc). Figure 1, which is adapted from [9], shows the economic principle of mMoney and the roles of the various actors. As depicted, the Mobile Network Operator (MNO) emits mMoney in partnership with a private bank. The MNO regularly produces compliancy reports to the Central Bank, responsible for the country's monetary policy. In particular, it is the MNO's responsibility to detect suspicious money laundering activities and to report them to the Central Bank, hence the importance of SIEM systems for MMT systems. The emitted mMoney can only be used among the MNO's clients who subscribe to the MMT service. The subscribers are end-users, service providers or retailers. They hold a prepaid account stored on a platform and accessible via the MNO's network and an application on their mobile device. Some users, such as retailers or service providers, can use computers to access their account. This account contains mMoney which can be acquired from the retailers. End-users can either transfer money to other end-users or purchase goods.

### B. Misuse case

We are interested in a misuse case related to money laundering using MMT (see Figure 2). A malicious user (fraudster) transfers small amounts of money to several mules. These mules can optionally receive the money at the end of a chain of mules. Then the mules of the final chain make a final transfer to a second malicious user. Each mule may keep a small percentage of the transfer as a salary. Also, these mule transfers can be manually operated by a mule or automatically transfered by a malicious software installed on the mobile phone and exploiting a flaw in the mobile money application. This way, the first malicious user has transfered money to the second malicious user, but there is no direct transfer traces between them.

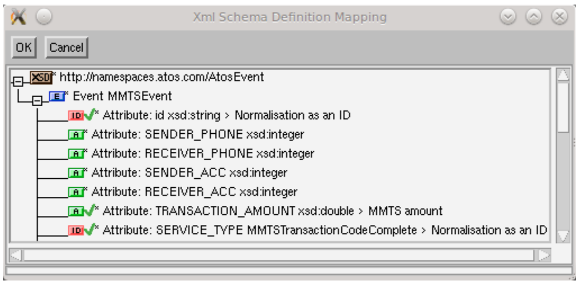
Many schemes of money laundering are known and criminals struggle to invent new ones [10], [11]. In this paper we consider a money laundering scheme involving the following assumptions: (i) there is only one mule in the chain of mules; (ii) the amount of a fraudulent transaction is much smaller then the average on the system; (iii) along with fraudulent actions the mules perform regular mMoney transfers; (iv) normal behavior of a MMT user (as observed in the operational logs) is persistent in regard to transaction amounts used, i.e. sudden changes in transferred mMoney amounts indicate an anomaly. These limitations do not restrict the proposed approach to fraud detection as far as one is able to model a process workflow related to a chosen (any other) money laundering scheme.

## III. Predictive Security Analysis at Runtime

The PSA provides support for the whole cycle of event-driven process security analysis.

*a) Event pre-processing:* Events come from an Extensible Markup Language (XML) stream or a database based on a pre-defined event schema, which is necessary in order to filter out data containing information not relevant to security analysis. Therefore, the PSA supports the creation of an event abstraction and mapping of events to the corresponding process instance (cf. Figure 3).

*b) Process specification, adaptation, and close-future behavior analysis:* The PSA uses Asynchronous Product Automata (APA) for formal process representation [12]. Process specifications given in EPC can be modeled by APA. EPCs are used in business process engineering, deployment and runtime control and supported by such leading products as SAP R/3 and ARIS [13]. The EPC language is intended to be easy to understand and use by people who are not familiar with formal specification methods. Therefore, the PSA assists the user not only in generation of a EPC, but also in transformation of this informal process model into an operational formal model. Existing EPC models can be adapted step-by-step, using archived event logs for replay and also interactively



XSD* http://namespaces.atos.com/AtosEvent
E* Event MMTSEvent
ID ✓* Attribute: id xsd:string > Normalisation as an ID
A* Attribute: SENDER_PHONE xsd:integer
A* Attribute: RECEIVER_PHONE xsd:integer
A* Attribute: SENDER_ACC xsd:integer
A* Attribute: RECEIVER_ACC xsd:integer
A ✓* Attribute: TRANSACTION_AMOUNT xsd:double > MMTS amount
ID ✓* Attribute: SERVICE_TYPE MMTSTransactionCodeComplete > Normalisation as an ID

Figure 3. Feature selection. Few fields of the events are used, and the concatenation of two of them provides the process id

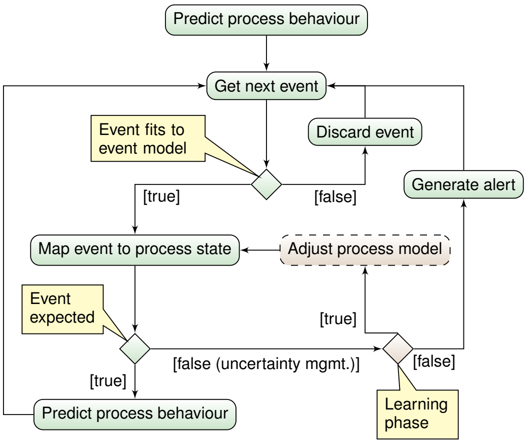

Predict process behaviour
Get next event
Event fits to event model
Discard event
Generate alert
[true]
[false]
Map event to process state
Adjust process model
Event expected
[true]
[false (uncertainty mgmt.)]
[false]
[true]
Predict process behaviour
Learning phase

Figure 4. Runtime behavior of the PSA

during runtime. The *uncertainty management* supports semi-automatic adaptation of process models according to the context conditions. Uncertainty situations can occur during synchronization of the state of a running process instance and the state of the model if the process model is not accurate enough or outdated, or when unknown events are received or expected events are missing. Beside APA models, Petri Net Markup Language (PNML) specifications [14] generated by process discovery tools (e.g., ProM [15]) can be imported. For all specification methods the computation of the close-future behavior is supported.

*c) Security requirements specification and evaluation:* The PSA allows for specification of the required security properties that the monitored process should fulfill (a security model in form of monitor automata), on-the-fly check of security requirements with respect to current process behavior (detection of critical process states), as well as techniques for on-the-fly check of security requirements with respect to close-future process behavior (prediction of critical process states).

*d) Situational awareness and alarm generation:* The PSA provides visualization of current process states with respect to security requirements by means of security monitors, and generation of alarms on detection of critical states.

In the work presented in this paper the PSA has been used in two phases, namely, a *learning phase* and an *anomaly detection phase*.

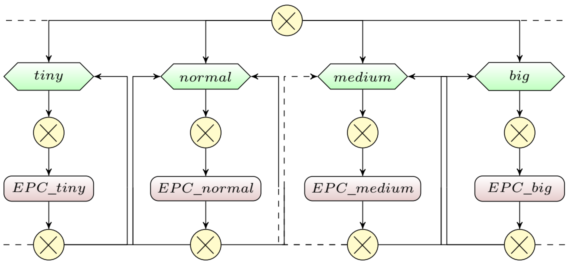


Figure 5. Subgraph of EPC for MMTS

In the initial learning phase, the normal behavior pattern with regard to the transaction characteristics is learned by processing an event log without malicious content. A mapping to classify the transactions with regard to the amount of money transferred (cf. Table I) and the order of such abstract events in the event log is used for this purpose. For example, an event from the MMT system where the amount of transferred money is greater than 500 but less or equal 1000 is mapped to the abstract event $medium$. This mapping is created empirically using real operational logs of the MMT system and can change if different training sets are used. Figure 4 shows an overview of the PSA event processing steps in the learning phase. The dashed action “Adjust process model” is done semi-automatically and requires the user’s involvement during runtime. All other actions are performed automatically.

Figure 5 shows a subgraph of an EPC which was learned for the MMT model. The graph shows the control flow structure of a process as a chain of events and functions. Rectangles with rounded corners denote EPC functions and hexagons denote EPC events. Functions represent active components, i.e., activities, tasks or process steps, which are triggered by events. Events are passive, they represent the occurrence of a state which describes the situation before, or after, a function is executed. Logical $and$, $or$, and $xor$ (exclusive or) operators are used to connect the basic constructs, in this way the control flow is specified. For example, after an event $medium$ the function $EPC\_medium$ is triggered. The expected “normal” events after execution of $EPC\_medium$ are $\{normal, medium, big\}$.

In the anomaly detection phase the PSA identifies deviations from the normal characteristics based on the values from a transaction monitor and generates alerts. In our experimental setup, the transaction monitor has been replaced by synthetic process behavior composed of simulated logs based on properties captured from real logs. In the anomaly detection phase the dashed action “Adjust process model” in Figure 4 is not used. Instead, an alert is generated automatically by the uncertainty management components of the PSA.

In order to reduce the number of alarms, it is necessary to configure the normal behavior model with the help of a real world dataset with annotated transactions (suspicious/not suspicious) or to use an additional component which filters the alerts generated by the PSA. Fraud analysis with additional components is beyond the scope of this paper.

## IV. Experimental Protocol

### A. PSA Configuration

The PSA is used in a non-interactive way: when it detects an unexpected event, an alert is automatically generated.

*1) Definition of a mapping:* The amount of a transaction is a continuous variable in $\mathbb{R}$, therefore discretization is necessary to get computable abstractions of the behavior. For this reason, we have empirically created various classes of transfer amounts (see Table I).

*2) Definition of an EPC:* In order to use the PSA it is necessary to define an EPC which models the MMT process. However, the events generation does not come from the control flow of the mobile money system, but from the behavior of its users. For this reason, the EPC must model the workflow of the user and not the workflow of the system. It is possible to define several processes in one PSA model. The PSA also provides the capability to investigate parallel running process instances with the same behavior. This capability was used to specify a general behavior which is followed by every user.

We found out that it is challenging to define a workflow of transactions because every user is free to use the system as he wants (*i.e.*, he can choose own amounts, frequencies, communities of interests, etc.). There is one process instance for each pair of active users and type of transaction (*i.e.*, $(user_1, CASHIN)$), because we make the assumption that the amount of transactions of the same type (*i.e.*, only $CASHIN$, only $TRANSFER$, etc.) are similar, while amounts of transactions of different kinds are not [16].

For this reason we have to create a more general process. A process behavior representation which is generated by the PSA from the respective EPC (cf. Figure 5) is shown in Figure 6. Each node is a state, each edge is a transition labeled with the event source. For each state, only the following transactions are authorized: do a transfer in the same amount family, do a transfer with the previous amount family, or do a transfer with the next amount family. The very first transition allows to go to any state. All the other possible transactions which are not present in the graph raise an alarm and are considered as being potentially malicious.

### B. Logs at our disposal

*1) Operational logs:* Although different kinds of logs (access, transfer, etc.) can exist in the mobile money transfer system, we only have at our disposal the transactions log. The transactions log contains the sender and receiver of the transaction, its amount, its success, the type of transaction and the type of the sender and of the receiver as well as other fields specifics to the system. These logs are driven by the behavior of the users: indeed the events are propagated only when the user do some transactions. We have more than 4.5 millions of correct events (accepted transactions) acquired on a period of 9 months.

However, as we have no ground truth (*i.e.*, fraudulent or not fraudulent) on these events, we cannot use them directly to detect fraud. They are nevertheless very useful to analyze the ability of the PSA to manage real life events in real time. For detection evaluation (in terms of error rates), we use simulated logs.

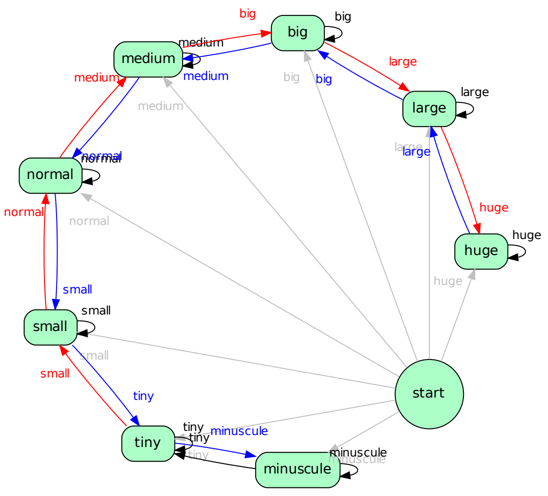


Figure 6. Process behavior representation of the EPC defined for MMTS

*2) Simulated Events:* As real world transactions have no ground truth we have implemented the misuse case in a simulator [17]. This simulated world is based on properties captured from real world transaction events. We are interested in a money laundering scenario (see Figure 2).

We have configured 3 scenarios with different number of users of the following categories:

*a) Regular users:* They make regular transfers (mean amount of 4000, standard deviation of 500), withdrawal and deposits.

*b) Malicious users:* (within the group of regular users). They want to exchange money without leaving direct traces in the system.

*c) Mules:* (within the group of regular users). They receive an amount (min amount of 20, max amount of 100) from a malicious user and transfer it later to another one after keeping 10% of interest.

*d) Merchants:* They correspond to shops where regular users can buy goods with mMoney.

*e) Retailers:* They allow end user to exchange mMoney with real money (and vice versa).

The parameters of the three scenarios are:

**S1** No money laundering: 50 regular users, 8 merchants, 4 retailers. This set serves to verify if normal transfers are detected as being fraudulent (False Positive).

**S2** Money laundering: 50 regular users, 5 mules, 8 merchants, 4 retailers. This set serves to verify if the PSA is able to detect frauds.

**S3** Money laundering with more individuals: 500 regular users, 10 mules, 16 merchants, 4 retailers This set serves to verify if the PSA can detect frauds when there are more non-fraudulent events.

Table I. MAPPING FOR DISCRETIZATION OF THE TRANSACTIONS' AMOUNTS

| **Class** | minuscule | tiny | small | normal | medium | big | large | huge |
|---|---|---|---|---|---|---|---|---|
| **Amount condition** | $\leq 5$ | $\leq 50$ | $\leq 200$ | $\leq 500$ | $\leq 1000$ | $\leq 2000$ | $\leq 5000$ | other |

Note that, we have used a slightly different mapping for the simulated logs in comparison to the real logs.

### C. Experiment

Our analysis tackles the computational and recognition performance of the PSA. We seek to ascertain how long it takes to process an event or spread an alert and whether it is possible to treat in real time the stream of events of an operational MMT system. The computations were done with a personal computer (2 cores CPU at 2.70GHz, 4Gb RAM). Finally, we will measure the error rates of the PSA. Any potential errors in detection will be taken advantage of in order to learn and improve the detection scheme.

### D. Evaluation metrics

The computational performance of the PSA is evaluated by counting the number of events per second successfully processed by the PSA. For the recognition performance we use several metrics: (a) *False Positive*, not fraudulent event is detected as being fraudulent; (b) *False Negative*, fraudulent event is detected as being not fraudulent; (c) *True Positive*, fraudulent event is detected as being fraudulent; (d) *True Negative*, not fraudulent event is detected as being not fraudulent.

## V. EXPERIMENTAL RESULTS

### A. Real events analysis

For the real events, we are interested in the computational performances. Figure 8a represents the number of transactions between the different states (only the transactions present in the events log are displayed). Most transactions are not considered as suspicious and most suspicious transactions are between the state "tiny" and "normal" (so a more accurate process model would allow such kind of transitions).

We found out that there was one process per pair of users and transaction type. With the real log, the PSA was able to manage 640,000 instances without any problem. 40 minutes were enough to process 4.5 millions of events, with the process behavior presented in Figure 8a, and produce 0.5 millions of alerts. 33 minutes were enough for a complete run which does not generate alerts. Set $X$ as the time to process an event and $Y$ the additional time to process an alert. They can be found by solving these two equations: $X * 4.5M = 33 * 60$ and $X * 4.5M + Y * 0.5M = 40 * 60$, which gives $X = 0.00044$ and $Y = 0.00480$. Thus, in the best theoretical case (no alerts are generated), the PSA is able to process more than 2200 events/second ($1/X$), while in the worst theoretical case (all events raise an alert), the number is reduced to 191 events/second ($1/(X+Y)$). We can summarize these results by saying that the PSA is able to manage an average of 100 millions of events per day on a standard computer ($(2200 + 191)/2 * 60 * 60 * 24$).

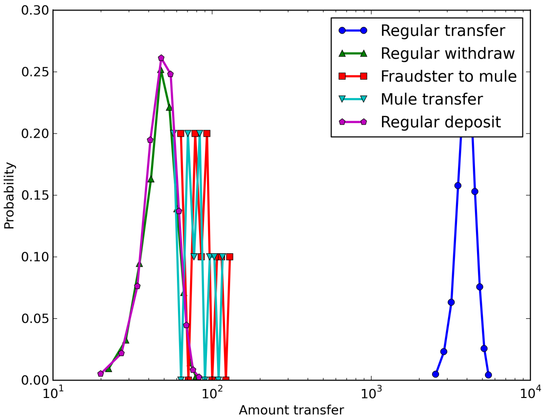


Figure 7. Histogram of the transactions amount of scenario S3

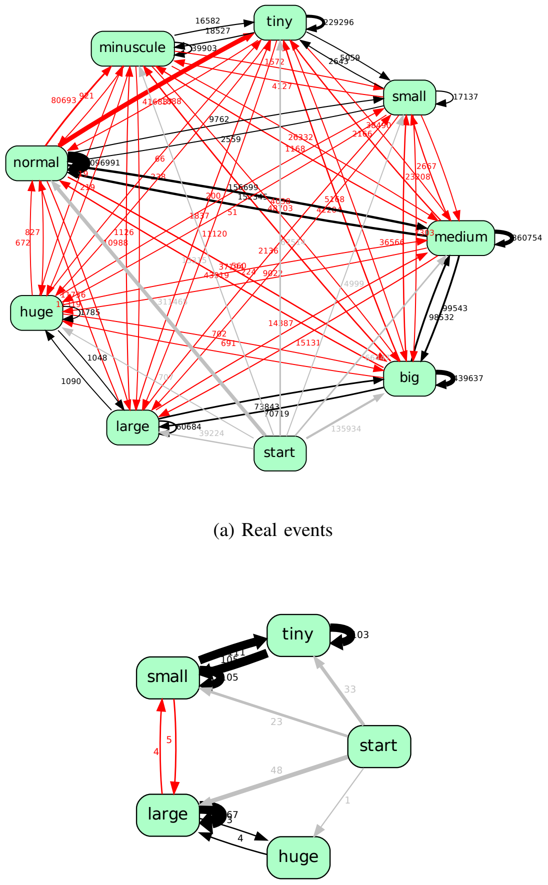
### B. Simulated events analysis

In case of the simulated events, we are interested in the recognition performances. As the simulation is stochastic, the evaluation has been repeated several times. However, the results are very similar for each run.

Figure 7 presents the histogram of the transactions amounts depending on their type in scenario S3. We can see that the distribution of the mule transfer is a bit translated to lower amounts in comparison to the distribution of initial fraudster's transfers (because of the interest kept by the mule) and overlaps with the distribution of withdrawal and deposits.

Figure 8b represents the number of transactions between the different states in scenario S2. For the simulated logs, we can see that 9 (4+5) transactions have been detected as suspicious. However, most of the transactions are in the range of authorized transactions according to the EPC.

Table II gives the recognition performance of the PSA on the selected example of each scenario, and Figure 9 presents the transactions of the participants involved in the fraud of S2 (the other users which made no transactions with the fraudsters are not displayed for clarity reasons). Each node represents a user, each edge represents a transaction, the label of an edge represents the index of the transaction in the sequence. Small gray edges represent True Negative, orange (gray for printed version) edges represent False Positive, green (dark gray) edges represent True Positive, red (black) edges represent False Negative. For scenario S1, the ratio of False Positive is null. For scenario S2, the ratio of False Positive is equal to $5/655 \simeq 1\%$, while the ratio of False Negative is of $6/10 \simeq 60\%$ if we consider the whole set of irregular transactions (first fraudster to mules and mules to last fraudster), or $1/5 \simeq 20\%$ if we consider the subset of irregular transactions detectable by the PSA (mules to last fraudster). For scenario S3, the ratio of False Positive is of $3/5297 \simeq 0.05\%$, while the ratio of False



(a) Real events

(b) Simulated events on scenario S2

Figure 8. Transactions obtained on a simulated run with a limited amount of users. Width of an edge is proportional to the number of transactions and red color corresponds to generated alerts

Negative is of $11/20 \simeq 55\%$ if we consider the whole set of irregular transactions, or $1/20 \simeq 5\%$ if we consider the subset of irregular transactions detected by the PSA.

The fraudulent transactions made from the initial fraudster to the mules are not detected as being suspicious, which is correct as the EPC has not been constructed to detect these transactions. The transactions of all mules, except *EU0*, to the fraudster are correctly detected as being suspicious. All the non-suspicious transactions of mule *EU0* are detected as being suspicious.

### C. Discussion

As we have no ground truth on the real world events, we cannot verify the recognition performance of the PSA. However we can notice that the number of alarms is quite important, and obviously superior to the real quantity of suspicious transactions. In order to reduce the number of alarms, it would be necessary to fine-tune the EPC with the help of a real world dataset with annotated transactions (suspicious/not suspicious)

Table II. SENSITIVITY AND SPECIFICITY OF THE PSA ON DETECTION OF ANOMALIES IN TRANSACTIONS

| | **Fraudulent** | **Normal** | *Total* |
|---|---|---|---|
| **Alarm raised** | 0 | 0 | 0 |
| **Alarm not raised** | 0 | 1077 | 1077 |
| *Total* | 0 | 1077 | 1077 |

(a) Scenario S1

| | **Fraudulent** | **Normal** | *Total* |
|---|---|---|---|
| **Alarm raised** | 5-1 | 5 | 9 |
| **Alarm not raised** | 1+5 | 650 | 656 |
| *Total* | 10 | 655 | 665 |

(b) Scenario S2

| | **Fraudulent** | **Normal** | *Total* |
|---|---|---|---|
| **Alarm raised** | 10-1 | 3 | 12 |
| **Alarm not raised** | 1+10 | 5294 | 5305 |
| *Total* | 20 | 5297 | 5317 |

(c) Scenario S3

or to use an additional component which filters the alerts generated by the PSA.

The PSA shows the correct behavior in all scenarios. The detection errors with user *EU0* come from the fact that the very first transaction of the user is the fraudulent one. The respective process state component is then set to this amount. Thus all his next transactions are detected as being suspicious as there is no transition in the process behavior representation to the state related to this amount. As a matter of fact, the other transactions of this user will be indefinitely detected as being suspicious in the future.

Usually, the evaluation of anomaly detection tools is done using a ROC curve [18] (sensitivity and specificity obtained for various configuration thresholds $\tau \in \mathbb{R}$). The PSA cannot be configured with such a simple threshold. Instead, there is a complex configuration ($\rho = (\rho_{EPC}, \rho_{mapping})$) composed of an EPC configuration ($\rho_{EPC} \in \mathbb{E}$, $\mathbb{E}$ is the set of possible EPCs) associated with a discretization scheme for transfer amounts ($\rho_{mapping} \in \mathbb{M}$, $\mathbb{M}$ is the set of possible mapping functions). It is thus difficult to automatically run through the possible choices of EPCs and discretization schemes in order to obtain several configurations giving the ROC curve. For this reason, we provide only one performance point.

## VI. Related Work

With respect to the exhaustive survey of approaches in the field of business process management given in [19], the functionality the PSA prototype [4], [5] used in this work could be classified as "check conformance using event data" approach. In this approach, information is used both from the process model and the event data in order to identify deviations of runtime behavior from expected behavior. The trend for this specific aspect of business process management, as presented in [19], shows a growing interest in the last three years. A similar approach is described in [20] but the focus is on quantification of inconsistencies by the formation of metrics. We consider the framework presented in [15] on runtime compliance verification for business processes as complementary to our work.

Many data-mining algorithms have been adapted for fraud detection in the banking field. Filters, decision trees and logistic

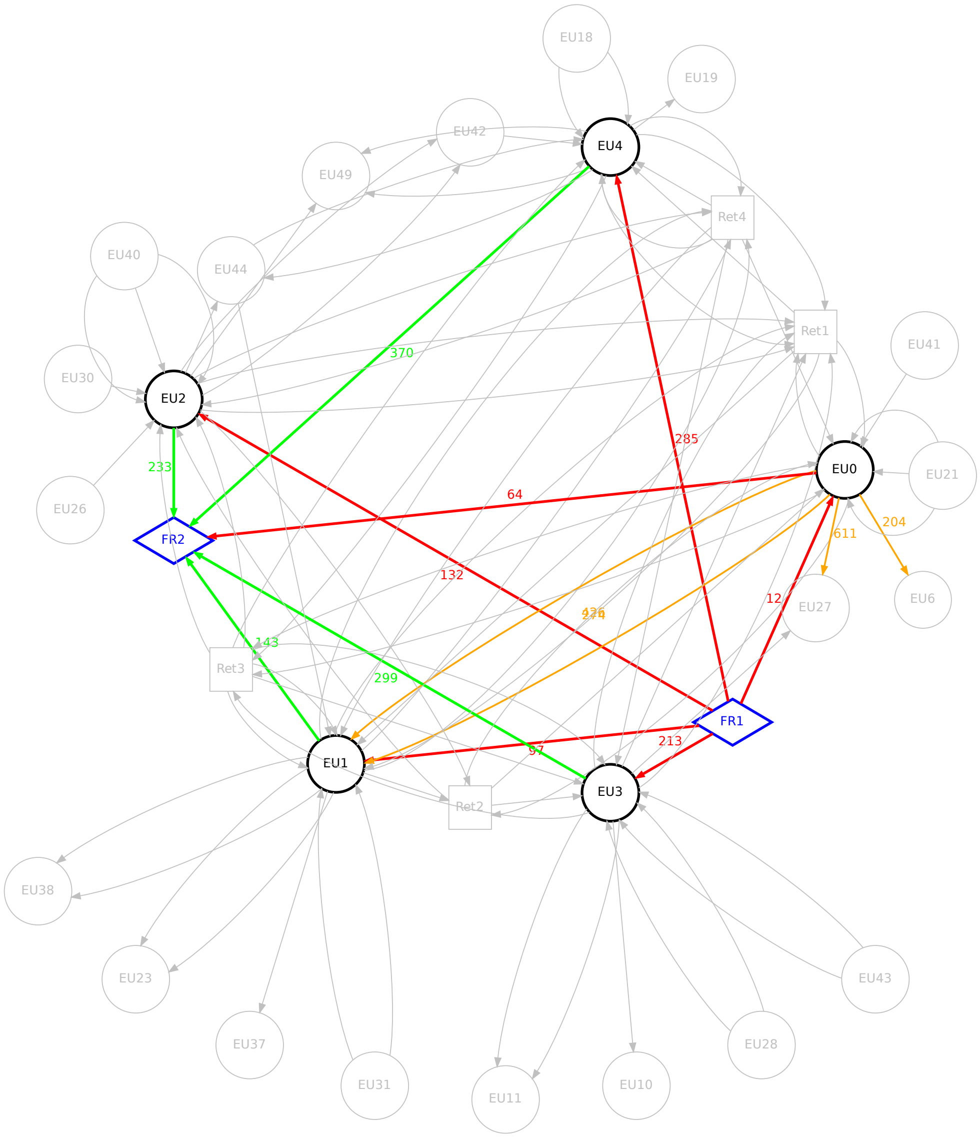


Figure 9. Representation of the users involved in the fraud and their transactions in scenario S2. The transfers of the mules, except those of $EU0$, have been detected. Various transfers from $EU0$ raised false alarms

regression are the most used because they can be easily interpreted. As a result, it is easier for an operator to explain to a client why a specific transaction is considered as being fraudulent. Other methods involving automated model learning are more rarely used because of the difficulty to interpret results and of the need for training data. However, there are some industrial solutions based on such methods. VISA, for example, implements neural networks in their fraud detection tool, RST (Real-Time Scoring) [21]. This tool associates a score to a transaction and raises an alert if the score exceeds a threshold chosen by the bank. However, it is the bank's responsibility to find out the reasons why a transaction which raised an alert should be blocked.

Bhattacharya et.al. [22] and Delamaire et.al. [23] published a state-of-the-art of the data-mining algorithms used for detecting frauds among credit card transactions. They show that several attempts were undertaken to adapt neural networks, SVMs, Bayesian networks, decision trees, expert systems and Hidden Markov Models to the field of credit card transactions. In [24], a separate change detection model for each cell in a multi-dimensional data cube is used for a change detection system for VISA. To our knowledge, not all mobile payment services include automated fraud detection solutions. The surveillance can be manual or based on business rules. However, the M-PESA service, which is one of the most well known MMT services, has deployed MinotaurTM Fraud Management Solution in 2012 [25]. This fraud management system is based on the use of business rules and neural networks [26]. To our knowledge, there are no public works concerning the study and the adaptation of fraud detection methods to mobile payment systems. Therefore, we cannot easily compare our work to existing systems.

## VII. Conclusion

The work presented in this paper utilizes alerts generated by the uncertainty reasoning component of the PSA to detect money laundering patterns in synthetic process behavior composed of simulated logs based on properties captured from real world transaction events.

We have shown that the PSA is able to raise alerts in a simulated scenario of fraud with mules. For this simulated scenario, the detection is efficient, but show that such system could be sensitive to noise in a real world system. It would be necessary to improve the resistance to noise through a correlation of the generated alerts or by an application of specific evaluation of the process states when an alert is generated (for example, move to the critical state if the same alert has been raised several times).

Results of the PSA should be associated with decision and reaction systems in order to modify the security rules of the MMT system to automatically block the fraud [3]. In order to ease the evaluation of the system, it could be interesting to develop methods able to automatically produce a huge quantity of EPCs to provide several evaluation points.

## Acknowledgment

The presented work was developed in context of the project MASSIF (ID 257475) being co-funded by the European Commission within the Seventh Framework Programme.

## References

[1] CCK, “Quarterly sector statistics report,” Communications Commission of Kenya, Tech. Rep., 2012.

[2] Orange, “Orange money,” http://www.orange.com/en/press/press-releases/press-releases-2012/Orange-Money-reaches-4-million-customers-and-launches-in-Jordan-and-Mauritius, June 2012, last visit on 12/04/2013.

[3] R. Rieke, L. Coppolino, A. Hutchison, E. Prieto, and C. Gaber, “Security and reliability requirements for advanced security event management,” in *Computer Network Security*, ser. LNCS, I. Kotenko and V. Skormin, Eds., 2012, vol. 7531, pp. 171–180.

[4] R. Rieke and Z. Stoynova, “Predictive security analysis for event-driven processes,” in *Computer Network Security*, ser. LNCS. Springer, 2010, vol. 6258, pp. 321–328.

[5] J. Eichler and R. Rieke, “Model-based Situational Security Analysis,” in *Workshop on Models@run.time*. CEUR, 2011, vol. 794, pp. 25–36.

[6] G. Keller, M. Nüttgens, and A.-W. Scheer, “Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)",” *Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes*, vol. 89, 1992.

[7] R. Rieke, E. Prieto, R. Diaz, H. Debar, and A. Hutchison, “Challenges for advanced security monitoring – the MASSIF project,” in *Trust, Privacy and Security in Digital Business*, ser. LNCS, S. Fischer-Hübner, S. Katsikas, and G. Quirchmayr, Eds. Springer, 2012, vol. 7449, pp. 222–223. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32287-7_23

[8] M. Achemlal, S. Gharout, C. Gaber, M. Llanes, E. Prieto, R. Diaz, L. Coppolino, A. Sergio, R. Cristaldi, A. Hutchison, and K. Dennie, “Scenario requirements,” MASSIF FP7-257475, Tech. Rep., 2011.

[9] W. Jack, S. Tavneet, and R. Townsend, “Monetary theory and electronic money: Reflections on the kenyan experience,” *Economic Quarterly*, no. 96, First Quarter 2010 2010.

[10] FINTRAC Typologies and Trends Reports, “Money laundering and terrorist financing trends in fintrac cases disclosed between 2007 and 2011,” http://www.fintrac-canafe.gc.ca/publications/typologies/2012-04-eng.asp#s1-1, April 2012, last visit on 21/05/2013.

[11] Internal Revenue Service (IRS), “Examples of money laundering investigations - fiscal year 2012,” http://www.irs.gov/uac/Examples-of-Money-Laundering-Investigations-Fiscal-Year-2012, October 2012, last visit on 21/05/2013.

[12] P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche, “The sh-verification tool – abstraction-based verification of co-operating systems,” *Formal Aspects of Computing, The International Journal of Formal Method*, vol. 10, pp. 381–404, 1998. [Online]. Available: http://sit.sit.fraunhofer.de/smv/publications/download/FormAsp.ps

[13] W. M. P. van der Aalst, “Formalization and verification of event-driven process chains,” *Information & Software Technology*, vol. 41, no. 10, pp. 639–650, 1999.

[14] M. Weber and E. Kindler, “The petri net markup language,” in *Petri Net Technology for Communication-Based Systems*, ser. LNCS. Springer, 2003, vol. 2472, pp. 124–144.

[15] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, “Monitoring business constraints with linear temporal logic: An approach based on colored automata,” in *Business Process Management (BPM 2011)*, ser. LNCS, vol. 6896. Springer, 2011, pp. 132–147.

[16] R. Rieke, R. Giot, and C. Gaber, “Predictive security analysis - concepts, implementation, first results in industrial scenario,” 2013, talk at CYBER SECURITY & PRIVACY EU FORUM 2013. [Online]. Available: http://www.cspforum.eu/uploads/Presentation-Roland_Rieke.pdf

[17] C. Gaber, B. Hemery, M. Achemlal, M. Pasquet, and P. Urien, “Synthetic logs generator for fraud detection in mobile transfer services,” in *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems (CTS2013)*, 2013.

[18] T. Fawcett, “Roc graphs: Notes and practical considerations for researchers,” *Pattern Recognition Letters*, vol. 27, no. 8, pp. 882–891, 2004.

[19] W. M. P. van der Aalst, “Business process management: A comprehensive survey,” *ISRN Software Engineering*, p. 37, 2013.

[20] A. Rozinat and W. van der Aalst, “Conformance checking of processes based on monitoring real behavior,” *Information Systems*, vol. 33, no. 1, pp. 64 – 95, 2008.

[21] VISA, “Security and trust at every level,” http://www.visaeurope.com/en/about_us/security.aspx, last visit on 22/03/2013.

[22] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, “Data mining for credit card fraud: A comparative study,” *Decision Support Systems*, vol. 50, 2011.

[23] L. Delamaire, H. Abdou, and J. Pointon, “Credti card fraud and detection techniques : a review,” *Banks and Bank systems*, vol. 4, 2009.

[24] C. Curry, R. L. Grossman, D. Locke, S. Vejcik, and J. Bugajski, “Detecting changes in large data sets of payment card data: a case study,” in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD ’07, 2007, pp. 1018–1022.

[25] E. Okutyi, “Safaricom tightens security on m-pesa with fraud management system,” http://www.humanipo.com/news/1341/Safaricom-tightens-security-on-M-Pesa-with-Fraud-Management-system, August 2012, last visited on 22/03/2013.

[26] Neural technologies, “Minotaurtm fraud detection software - finance sector,” http://www.neuralt.com/fraud_detection_software.html, last visited on 23/03/2013.

# P19

## MONITORING SECURITY COMPLIANCE OF CRITICAL PROCESSES

| | |
|---|---|
| **Title** | Monitoring Security Compliance of Critical Processes |
| **Authors** | Roland Rieke, Jürgen Repp, Maria Zhdanova, and Jörn Eichler |
| **Publication** | *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, 0:525–560. |
| **ISBN/ISSN** | ISBN-13: 978-1-4799-2728-9 |
| **DOI** | `http://dx.doi.org/10.1109/PDP.2014.106` |
| **Status** | Published |
| **Publisher** | IEEE |
| **Publication Type** | Conference Proceedings |
| **Copyright** | 2014, IEEE |
| **Contribution of Roland Rieke** | Main Author, editor, and presenter at the special session on "Security in Networked and Distributed Systems" at the 22th Euromicro Conference on Parallel, Distributed and Network-Based Processing. |

Table 24: Fact Sheet Publication *P19*

Publication P19 [Rieke, Repp, Zhdanova & Eichler, 2014] addresses the following research questions:

RQ10 *How can security analysis at runtime exploit process models to identify current and close-future violations of security requirements?*

With respect to RQ10, this paper presents an approach to support evaluation of the security status of processes at runtime. The approach is based on operational formal models derived from process specifications and security policies comprising technical, organisational, regulatory and cross-layer aspects. A process behaviour model is synchronised by events from the running process and utilises prediction of expected close-future states to find possible security violations and allow early decisions on countermeasures.

In particular, the algorithm for the evaluation of security requirements at runtime is described.

RQ11 *How can security analysis at runtime be integrated in a security management strategy?*

In this paper, the implementation of the PSA@R approach by the prototype, the PSA, is described and results of evaluation of specific aspects, such as effects of the number of security requirements, different abstraction levels and the variation of prediction depths are provided.

RQ12a *Can the developed methods and tools be successfully adapted to large scale industrial scenarios?*

RQ12b *What are the performance effects of the number of events, processes, security requirements, predicted steps, and of event abstraction?*

In this paper, the applicability of the PSA@R approach is exemplified by a misuse case scenario from a hydroelectric power plant that was analysed in the European research project MASSIF. Security requirements are taken from a combined technical and organisational process from a hydroelectric power plant in a dam [Romano et al., 2012]. Since dams are complex infrastructures, a huge number of parameters must be monitored to guarantee safety and security. Which parameters are actually monitored, depends on the dam's structure, design, purpose and function [Coppolino et al., 2012].

In particular, the algorithm for the evaluation of security requirements at runtime is described and an extensive example concerning safety critical actions in the control room is given.

In the project MASSIF [Rieke et al., 2012] PSA@R is currently applied to check security requirements in four industrial domains: (i) the management of the Olympic Games IT infrastructure [Vianello et al., 2013]; (ii) a mobile phone based Mobile Money Transfer Service (MMTS) [Gaber et al., 2013], facing high-level threats such as money laundering; (iii) managed IT outsource services for large distributed enterprises and (iv) an IT system supporting a critical infrastructure (dam) [Romano et al., 2012]. The hydroelectric power plant scenario (iv) has been used to demonstrate the capability of the PSA prototype to process and correlate events from heterogeneous sources. To evaluate the PSA prototype with respect to performance issues, however, event logs from scenario (ii) have been used as a resource intensive application which requires high throughput.

The measurements presented evaluate the execution time and the number of events received by the PSA. Four aspects important from the application perspective have been examined: (i) effects of the number of security requirements to the execution time; (ii) effects of the abstraction level to analysis; (iii) effects of cycle reduction in a Reachability Graph (RG); (iv) effects of changing prediction depths.

During the experiment the security requirements were successfully checked in all combinations.

# Monitoring Security Compliance of Critical Processes

Roland Rieke[*][†], Jürgen Repp[†], Maria Zhdanova[†], and Jörn Eichler[‡]
[*]Philipps-Universität Marburg, Germany
[†]Fraunhofer SIT, Darmstadt, Germany
Email:{roland.rieke,juergen.repp,maria.zhdanova}@sit.fraunhofer.de
[‡]Fraunhofer AISEC, Munich, Germany
Email: joern.eichler@aisec.fraunhofer.de

***Abstract*—Enforcing security in process-aware information systems at runtime requires the monitoring of systems' operation using process information. Analysis of this information with respect to security and compliance aspects is growing in complexity with the increase in functionality, connectivity, and dynamics of process evolution. To tackle this complexity, the application of models is becoming standard practice. Considering today's frequent changes to processes, model-based support for security and compliance analysis is not only needed in pre-operational phases but also at runtime.**

**This paper presents an approach to support evaluation of the security status of processes at runtime. The approach is based on operational formal models derived from process specifications and security policies comprising technical, organizational, regulatory and cross-layer aspects. A process behavior model is synchronized by events from the running process and utilizes prediction of expected close-future states to find possible security violations and allow early decisions on countermeasures. The applicability of the approach is exemplified by a misuse case scenario from a hydroelectric power plant.**

***Keywords-predictive security analysis; process behavior analysis; security modeling and simulation; security monitoring; critical infrastructures; security information and event management.***

## I. Introduction

Electronic business processes contribute significantly to the performance of today's enterprises and their correct execution is vital for many companies. Automated enactment of business processes applying Information Technology (IT) does not only bring competitive advantages but induces higher security risks. A new Internet security threat report [1] states a more than 81% surge in malicious attacks including sophisticated targeted attacks. Yet, existing Business Process Management (BPM) methodologies often neglect security and dependability objectives [2]. At the same time, business processes become more complex encompassing a wide range of heterogeneous systems and applications and undergo continuous changes to sustain business competitiveness [3]. Another dimension is added by the inter-connection of business processes with modern automated management systems that support remote control of multiple infrastructures. Thus, cross-layer connections between high-level business processes, organizational processes, and low-level technical processes controlling sensors and actuators in cyber-physical systems emerge. Increasing complexity and changeability complicates analysis of distinctive process properties demanding frequent adjustments of process models to address changing business needs [4]. This involves not only functional correctness of a process model, but also related compliance and security features. Hard-coded controls can restrain flexibility required to ensure adequate formal representation of evolving processes [5], [6].

We present an approach for predictive security analysis at runtime, which allows to add security requirements regarding process behavior during execution without the need to modify the corresponding process model. In doing so, we do not intend to diminish the significance of security-by-design. Our work is aimed as a critical add-on in order to address the dynamics of electronic business processes. Based on close-future behavior models computed on-the-fly from process specifications, we demonstrate early detection of deviations of process execution from expected behavior which can be caused by attacker intervention. We propose a new method for security analysis at runtime exploiting process behavior models, which enables on-the-fly security compliance checks and prediction of close-future violations of security requirements. The proposed integration of simulation and runtime monitoring allows for early security warnings and predictive alarms on possible security critical states in close future. In order to demonstrate how our model-based runtime analysis is applied, we have chosen processes from a hydroelectric power plant in a dam that was analyzed in a European research project [7]. We describe an implementation of our approach and provide results of evaluation of specific aspects, such as effects of the number of security requirements, different abstraction levels and the variation of prediction depths.

Section II of this paper gives an overview of our approach for predictive security analysis at runtime. Section III introduces the operational process model, the close-future behavior model, and the synchronization with the running process. Section IV presents the security model applied at runtime to identify security relevant states. Section V provides an example for the runtime analysis of security requirements from a hydroelectric power plant. Section VI describes the prototype implementation and evaluation results. Section VII reviews related work and Section VIII presents conclusions and further research.

## II. Predictive Security Analysis at Runtime

In this section we introduce a new model-based approach for Predictive Security Analysis at Runtime (PSA@R). Our approach integrates formal process modeling with simulation of (close-future) process behavior triggered by real-time data. Process behavior models are used to identify and predict violations of security requirements during process execution.

In PSA@R the operation of a system or a system of systems is observed analyzing events received from this system. PSA@R is not executed by this *observed system* but rather by an *observing system* such as a Security Information and Event Management (SIEM) system. It is presupposed here that the observing system itself is trustworthy. A SIEM system can be easier protected against attacks than the system under observation. Regarding the observed system it is assumed that its purpose is given by technical, organizational, and business processes and that the intended behavior can be specified by process models. The behavior of the observed system is then a composition of the behaviors of the running processes.

PSA@R operates with formalized views on the control flow and security properties of a business process that can exist in any common or application-specific technical workflow notation [8], [9], [10]. A *process model*, which provides a formal representation of the controlled process, and an *event model*, an abstraction defining the internal mapping for input event streams, need to be created at the preliminary stage of PSA@R. Security requirements to be satisfied during process execution are formalized by a *security model*, which must be derived systematically [11], [12] at the initialization time.

At the analysis stage of PSA@R, the formal models are applied to monitor and predict process behavior and identify security relevant states on-the-fly. Figure 1 illustrates steps of predictive security analysis at runtime. Given the process model and the current state of the running process, a *process behavior model* representing the adjacent expected future states of the process can be computed. The process behavior model is synchronized with the running process through events received from the execution environment. Incoming events are interpreted using the event model and mapped to the process behavior model. Compliance of the predicted states with the established security policy is evaluated against the security model. To identify security relevant states on-the-fly PSA@R uses a new method described in Section IV, which enables detection of (close-future) requirements violation.

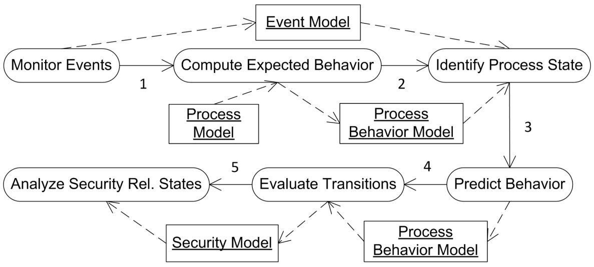


Fig. 1: Analysis stage of PSA@R

## III. Process Model

This section introduces the formal process model, which is utilized to reflect the current state of the system and provides the basis for the prediction of close-future actions. PSA@R uses a process model given by an Asynchronous Product Automata (APA) representation that provides a flexible operational specification concept for cooperating systems [13]. An APA consists of a family of *elementary automata* communicating by common components of their state (shared memory).

*Definition 1:* An *Asynchronous Product Automaton* consists of

- a family of *state sets* $Z_s, s \in \mathbb{S}$,
- a family of *elementary automata* $(\Phi_e, \Delta_e), e \in \mathbb{E}$ and
- a *neighbourhood relation* $N : \mathbb{E} \to \mathcal{P}(\mathbb{S})$.

$\mathbb{S}$ and $\mathbb{E}$ are index sets with the names of state components and of elementary automata and $\mathcal{P}(\mathbb{S})$ is the power set of $\mathbb{S}$. For each elementary automaton $(\Phi_e, \Delta_e)$ with *Alphabet* $\Phi_e$, its *state transition relation* is

$$\Delta_e \subseteq \bigtimes_{s \in N(e)}(Z_s) \times \Phi_e \times \bigtimes_{s \in N(e)}(Z_s).$$

For each element of $\Phi_e$ the state transition relation $\Delta_e$ defines state transitions that change only the state components in $N(e)$. An APA's (global) *states* are elements of $\bigtimes_{s \in \mathbb{S}}(Z_s)$.
To avoid pathological cases it is generally assumed that $\mathbb{S} = \bigcup_{e \in \mathbb{E}}(N(e))$ and $N(e) \neq \emptyset$ for all $e \in \mathbb{E}$.
Each APA has one *initial state* $q_0 = (q_{0s})_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$. In total, an APA $\mathbb{A}$ is defined by

$$\mathbb{A} = ((Z_s)_{s \in \mathbb{S}}, (\Phi_e, \Delta_e)_{e \in \mathbb{E}}, N, q_0).$$

*Definition 2:* An elementary automaton $(\Phi_e, \Delta_e)$ is *activated* in a state $q = (q_s)_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$ as to an *interpretation* $i \in \Phi_e$, if there are $(p_s)_{s \in N(e)} \in \bigtimes_{s \in N(e)}(Z_s)$ with $((q_s)_{s \in N(e)}, i, (p_s)_{s \in N(e)}) \in \Delta_e$. An activated elementary automaton $(\Phi_e, \Delta_e)$ can execute a state transition and produce a *successor state* $p = (p_s)_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$, if $q_r = p_r$ for $r \in \mathbb{S} \setminus N(e)$ and $((q_s)_{s \in N(e)}, i, (p_s)_{s \in N(e)}) \in \Delta_e$. The corresponding state transition is $(q, (e, i), p)$.

However, PSA@R does not depend on the formal method chosen for model representation. The only requirement is, that it must be possible to compute the process behavior from the process model (cf. Section III-C). For example, Petri nets [9] also meet this requirement and models produced in Petri Net Markup Language (PNML) [14] by process mining and discovery tools [15] can be used instead of APA specifications.

### A. Process Behavior Model

Formally, the behavior of an operational APA model of a business process is described by a Reachability Graph (RG), also referred to as Labeled Transition System (LTS) [16].

*Definition 3:* The behavior of an APA is represented by all possible coherent sequences of state transitions starting with initial state $q_0$. The sequence

$$(q_0, (e_1, i_1), q_1)(q_1, (e_2, i_2), q_2) \ldots (q_{n-1}, (e_n, i_n), q_n)$$

with $i_k \in \Phi_{e_k}$, where $\Phi_{e_k}$ is the alphabet of the elementary automaton $e_k$, represents one possible sequence of actions of an APA.

State transitions $(p,(e,i),q)$ may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA: $(p,(e,i),q)$ is the edge leading from $p$ to $q$ and labelled by $(e,i)$. The subgraph reachable from the node $q_0$ is called *Reachability Graph* of an APA.

*Example 1:* A process specification provides the control flow structure of a process as a sequence of *events* and *functions*. In an APA model that is derived from a process specification, the set of possible output events of a process function can be used as the alphabet of the elementary automaton representing the function [17]. So the interpretation $i$ is the output event. An example for a state transition is: $(p,(transfer, event = 'critical'),q)$. The parameters of this state transition are the state $p$, the tuple composed of the elementary automaton *transfer* and its interpretation $event = 'critical'$, and the follow-up state $q$.

### B. Event Model

A stream of events characterizes one specific execution trace of the observed system. This trace is a shuffle of the traces of the executed process instances. The event model determines the internal mapping for the runtime events defined by an event schema. To reduce the complexity only data required for the analysis or in generated alarms should be used in the model.

Formally, it is assumed that an event represents a letter of the alphabet that denotes the possible actions in the system. Different formal models of the same system are partially ordered with respect to different levels of abstraction.

*Definition 4: Abstractions* are described by so-called alphabetic language homomorphisms. These are mappings $h^* : \Sigma^* \longrightarrow \Sigma'^*$ with $h^*(xy) = h^*(x)h^*(y)$ , $h^*(\varepsilon) = \varepsilon$ and $h^*(\Sigma) \subset \Sigma' \cup \{\varepsilon\}$. So they are uniquely defined by corresponding mappings $h : \Sigma \longrightarrow \Sigma' \cup \{\varepsilon\}$. In the following both the mapping $h$ and the homomorphism $h^*$ is denoted by $h$. In general, let $\check{L} \subset \check{\Sigma}^*$ and $L \subset \Sigma^*$ be prefix closed languages. $\check{L}$ is called *finer than* $L$ and $L$ is called *coarser than* $\check{L}$ iff an alphabetic homomorphism $\nu : \check{\Sigma}^* \to \Sigma^*$ exists with $\nu(\check{L}) = L$.

Let now $P$ denote a finite set of process instances $i$ of some process with $i \in P$ and let $\Sigma_i$ denote pairwise disjoint copies of $\Sigma$. The elements of $\Sigma_i$ are denoted by $e_i$ and $\Sigma_P := \bigcup\limits_{i \in P} \Sigma_i$. The index $i$ describes the bijection $e \leftrightarrow e_i$ for $e \in \Sigma$ and $e_i \in \Sigma_i$. Now the projection $\pi$ identifies events from a specific process instance $i$.

*Definition 5:* For $i \in P$, let $\pi_i^P : \Sigma_P^* \to \Sigma^*$ with

$$\pi_i^P(e_r) = \begin{cases} e \mid & e_r \in \Sigma_i \\ \varepsilon \mid & e_r \in \Sigma_P \setminus \Sigma_i \end{cases} .$$

This is similar to the notion of a *correlation condition* [18] that defines which sets of events in the service log belong to the same instance of a process.

*Remark 1:* For effective use of PSA@R it is assumed that a process instance projection is possible for each event. In many applications, a process instance identification is directly available as an attribute of the event. Sometimes a set of attributes identifies the process instance. In some cases the assumption about pairwise disjoint alphabets is not true.

If the event data contain redundant or irrelevant attributes, a proper subset of attributes for use in model construction has to be selected. In order to avoid state space explosion problems, the *coarsest* abstraction that still contains all security relevant information should be used.

*Example 2:* Let us assume that $\Sigma$ is the alphabet of events from the measured system and for a given event $e$ the term $\#(e)$ denotes the value of an attribute involved in a transaction. Let $h_2, h_3 : \Sigma^* \to \{'high', 'medium', 'low'\}^*$ the homomorphisms given by

$$h_2(e) = \begin{cases} 'high' \mid & 10^5 < \#(e) \\ 'low' \mid & \#(e) \leq 10^5 \end{cases}$$

$$h_3(e) = \begin{cases} 'high' \mid & 10^5 < \#(e) \\ 'medium' \mid & 10^3 < \#(e) \leq 10^5 \\ 'low' \mid & \#(e) \leq 10^3 \end{cases} .$$

Then $h_3$ and $h_2$ can be used to differentiate process control flow with respect to events with different attribute values. $h_3$ is finer than $h_2$ because $\nu : \{'high', 'medium', 'low'\}^* \to \{'high', 'low'\}^*$ exists.

### C. Prediction of Close-future Process Actions

At runtime, the current state of the process behavior model of the process instance $i$ is synchronized with the running process using the projection of the measured events to the respective state transitions $(p,(e,i),q)$ of the RG. PSA@R uses the RG to predict the close-future behavior of the process instance. As the process description is formalized in the process model, a subgraph of the RG can always be computed on-the-fly starting with the current state of the process instance. The *prediction depth* is the depth of this subgraph starting from the current state.

*Example 3:* The approach taken for the prediction of close-future behavior within a process is illustrated by Fig. 2. The ellipses in the event stream pane denote the observed events, whereby the filled ellipses $e_0$, $e_1$, $e_2$, and $e_3$ denote the events that belong to the specific process instance $i$, i.e.,

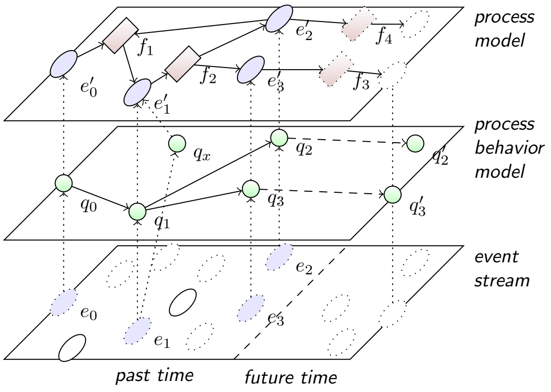


Fig. 2: Predict close-future process behavior

$e_0, e_1, e_2, e_3 \in \Sigma_i$. The ellipses in the process model pane denote abstract events with respect to the event model, e.g., $e'_1 = h(\pi_i^P(e_1))$. The dotted arrows denote this mapping. The rectangles in the process specification pane denote the process functions and the solid lines denote the transitions. If in Fig. 2 the function $f_2$ is modeled by the elementary automaton *transfer* and $e'_3 = h(\pi_i^P(e_3)) = 'high'$ and the depicted process instance $i$ is in the state $q_1$ and the event $e_3$ is received, then the transition $(q_1, (transfer, event = 'high'), q_3)$ will match the current situation. The process specification contains possible close-future functions $f_3$ and $f_4$ and associated events to be predicted. The dashed arrows in Fig. 2 denote the predicted close-future process behavior.

## IV. On-the-fly Identification of Critical States

In addition to the predicted process behavior, the security model is needed to identify security relevant states of the current state of the business process. As a notation for the security model we use monitor automata.

*Definition 6:* A monitor automaton $\mathbb{M}$ consists of a set $\mathscr{M}$ of labeled states, an alphabet $\Lambda$ of predicates on RG state transitions, a transition relation $\mathscr{T}_{\mathscr{M}} \subseteq \mathscr{M} \times \Lambda \times \mathscr{M}$, a set of initial states $\emptyset \neq \mathscr{M}_0 \subset \mathscr{M}$, and a set of accepting states $\mathscr{M}_f \subset \mathscr{M}$.

Predicates of $\mathbb{M}$ are applied to state transitions $(p_i, (e_j, i_k), q_l)$ of the RG.

*Example 4:* The predicate $(, (, event = 'high'), )$ is *true* if $'high'$ is bound to the interpretation variable *event* of the interpretation $i_k$. No condition for the predecessor state $p_i$, successor states $q_l$ and the elementary automaton $e_j$ is given.

With a monitor automaton it is possible to express security requirements with respect to current and close-future behavior of a process represented by a RG. In this case accepting states refer to *security critical states*. Each state of the RG has an associated state set of $\mathbb{M}$, which is computed during simulation. Security critical states are reached whenever an accepting state of $\mathbb{M}$ becomes a member of such state set.

State sets of $\mathbb{M}$ are successively assigned to RG states during simulation as follows: $\mathscr{M}_0$ is assigned to the initial state $q_0$ of the given RG. Each predicate $\lambda \in \Lambda$ of $\mathbb{M}$ is of the form $\lambda(x)$, where $x$ is a state transition $(p, (e, i), q)$ of the RG. Each $\lambda \in \Lambda$ is associated with one of the transitions $\mathscr{T}_{\mathscr{M}}$ of $\mathbb{M}$. During the run of the simulation, for each transition $(q_i, (e_j, i_k), q_x)$ of the RG, the monitor automaton state set for the RG state $q_x$ is computed as follows:
Let $\mathscr{A}_i \subseteq \mathscr{M}$ be the state set of $\mathbb{M}$ assigned to the current RG state $q_i$. The set $\mathscr{T}_{A_i}$ of transitions to be checked is now given by:

$$\mathscr{T}_{A_i} = \{(m_m, \lambda_o, m_n) \in \mathscr{T}_M \mid m_m \in \mathscr{A}_i\}.$$

All predicates $\lambda_o$ have to be checked for the current transitions $(q_i, (e_j, i_k), q_x)$ of the RG. Based on the monitored transitions $\mathscr{M}\mathscr{T}_x$ new states $\mathscr{B}_x$ of $\mathbb{M}$ and the changed states $\mathscr{C}_x$ of $\mathbb{M}$ are computed as follows:

$$\mathscr{M}\mathscr{T}_x := \{(m_m, m_n) \in \mathscr{M} \times \mathscr{M} \quad | \quad (m_m, \lambda_o, m_n) \in \mathscr{T}_{A_i} \wedge \lambda_o((p_i, (e_j, i_k), q_x))\}$$

$$\mathscr{B}_x := \{m_n \in \mathscr{M}\mathscr{T}_x | (m_m, m_n) \in \mathscr{M}\mathscr{T}_x\}$$
$$\mathscr{C}_x := \{m_m \in \mathscr{M}\mathscr{T}_x | (m_m, m_n) \in \mathscr{M}\mathscr{T}_x\}$$

The computation of $\mathscr{B}_x$ and $\mathscr{C}_x$ will be implicitly assumed when used in Algorithm 1 and 2. The set $\mathscr{R}\mathscr{S}_n$ includes possible states in the RG which represents the current state of the real system. After the occurrence of a certain event and extension of the RG if necessary, the new set $\mathscr{R}\mathscr{S}_{n+1}$ and the corresponding monitor automaton state set $\mathscr{A}_i^r$ has to be computed for every state $q_i \in \mathscr{R}\mathscr{S}_{n+1}$ by Algorithm 1.

*Algorithm 1 (Security compliance check):*

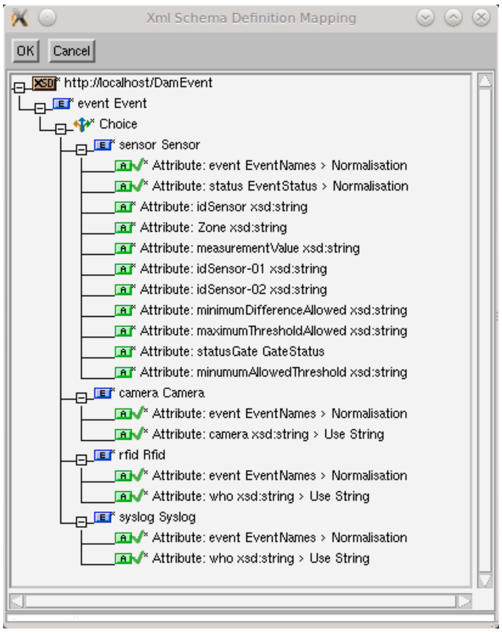
```
$SP := \emptyset$
for $(p_i, (e_j, i_k), q_l) \in \{p_i | p_i \in \mathscr{R}\mathscr{S}_n\} \wedge \lambda_e((p_i, (e_j, i_k), q_l))$ do
    $SP := SP \cup \{p_i\}$
    if $\mathscr{B}_l = \emptyset$ then
        if $q_l \in SP$ then
            $\mathscr{A}_l^r := \mathscr{A}_l^r \cup \mathscr{A}_i^r$
        else
            $\mathscr{A}_l := \mathscr{A}_i$
    else
        if $q_l \in SP$ then
            $\mathscr{A}_l^r := \mathscr{A}_i^r \cup \mathscr{B}_l$
        else
            $\mathscr{A}_l^r := \mathscr{B}_l \cup (\mathscr{A}_i^r \setminus \mathscr{C}_l)$
$\mathscr{R}\mathscr{S}_{n+1} := SP$
```

For the RG state set $\mathscr{R}\mathscr{S}_{n+1}$ new state sets $\mathscr{A}_i^p$ of $\mathbb{M}$ have to be computed for every predicted RG state $q_i$. The function *visit* sets a mark to a certain state which can be checked by the predicate *visited*. The predicate *closer* indicates that the current path to elements of the state set $\mathscr{R}\mathscr{S}_{n+1}$ to the node given as a parameter is shorter than the paths to this node processed before. The monitor automaton state sets of the predicted states which can be reached from states of the set $\mathscr{R}\mathscr{S}$ are computed by Algorithm 2.

*Algorithm 2 (Predict security violations):*

```
$S := \emptyset$
for $q_l \in \mathscr{R}\mathscr{S}_{n+1}$ do
    $S := S \cup \{q_l)\}$
    while $S \neq \emptyset$ do
        $S := S \setminus \{q_{i_2}\}$
        for $(q_{i_2}, (e_{j2}, i_{k_2}), q_{l_2}) \in$ RG do
            $A := \begin{cases} A_{i2}^r \mid & \neg visited(q_{i2}) \wedge \mathscr{A}_{i_2}^p = \emptyset \\ \mathscr{A}_{i_2}^p \mid & \textbf{\textit{else}} \end{cases}$
            $visit(q_{i_2})$
            if $\mathscr{B}_{l_2} = \emptyset$ then
                if visited$(q_{l_2})$ then
                    $\mathscr{A}_{l_2}^p := \mathscr{A}_{l_2}^p \cup \mathscr{A}_{i_2}^p$
                    if $closer(q_{l_2}, \mathscr{R}\mathscr{S}_{n+1})$ then
                        $S := S \cup \{q_{l_2})\}$
                else
                    $\mathscr{A}_{l_2}^p := \mathscr{A}_{i_2}^p$
                    $S := S \cup \{q_{l_2}\}$
            else
                if visited$(q_{l_2})$ then
                    $\mathscr{A}_{l_2}^p := \mathscr{A}_{i_2}^p \cup \mathscr{B}_{l_2}$
                else
                    $\mathscr{A}_{l_2}^p := \mathscr{B}_{l_2} \cup (\mathscr{A}_i^p \setminus \mathscr{C}_{l_2})$
                $S := S \cup \{q_{l_2})\}$
```

In this algorithm we do not analyze the consequences of reaching security critical states. Trigger actions such as generating alerts, which are executed when accepting monitor



```
XSD http://localhost/DamEvent
  E event Event
    Choice
      E sensor Sensor
        A Attribute: event EventNames > Normalisation
        A Attribute: status EventStatus > Normalisation
        A Attribute: idSensor xsd:string
        A Attribute: Zone xsd:string
        A Attribute: measurementValue xsd:string
        A Attribute: idSensor-01 xsd:string
        A Attribute: idSensor-02 xsd:string
        A Attribute: minimumDifferenceAllowed xsd:string
        A Attribute: maximumThresholdAllowed xsd:string
        A Attribute: statusGate GateStatus
        A Attribute: minumumAllowedThreshold xsd:string
      E camera Camera
        A Attribute: event EventNames > Normalisation
        A Attribute: camera xsd:string > Use String
      E rfid Rfid
        A Attribute: event EventNames > Normalisation
        A Attribute: who xsd:string > Use String
      E syslog Syslog
        A Attribute: event EventNames > Normalisation
        A Attribute: who xsd:string > Use String
```

Fig. 3: Attribute selection and mapping

automaton states become members of a state set, can be defined. Different security properties might be monitored simultaneously by allowing more than one transition of the monitor automaton to be triggered at the same time.

## V. Hydroelectric Power Plant Security

In order to demonstrate what kind of security requirements we consider and how our model-based runtime analysis is applied, we use a combined technical and organizational process from a hydroelectric power plant in a dam [7] and explain the evaluation of security requirements for this process at runtime. Since dams are complex infrastructures, a huge number of parameters must be monitored to guarantee safety and security. Which parameters are actually monitored, depends on the dam's structure, design, purpose and function [19].

Figure 3 shows a mapping (an event model) with regard to the events from dam sensors, cameras, RFID scanners, and syslog.

Here, we examine a misuse case related to the insider threat that is still prevalent and posing a serious risk to critical infrastructures [20]. We assume that the respective security goal is given as: *All safety critical actions in the control room are carried out by a dam operator with administrative rights.* Other types of security requirements, which could be supervised by PSA@R, are typical authenticity and integrity requirements like the following example: *Whenever a certain control decision is made, the input information that presumably led to it must be authentic [19].* Specifically, *authenticity* can be seen as the assurance that a particular action has occurred in the past.

### A. Misuse Case Scenario

The storage dam of the hydroelectric power plant is remotely controlled by a Supervisory Control And Data Acquisition (SCADA) system from the control station located in the control room. Physical (Radio Frequency Identification (RFID) based) and logical access controls are deployed. A disgruntled employee of the dam with a non-administrative role (i.e., cleaning staff) but who is enabled to access the control room uses stolen administrator credentials to open dam gates.

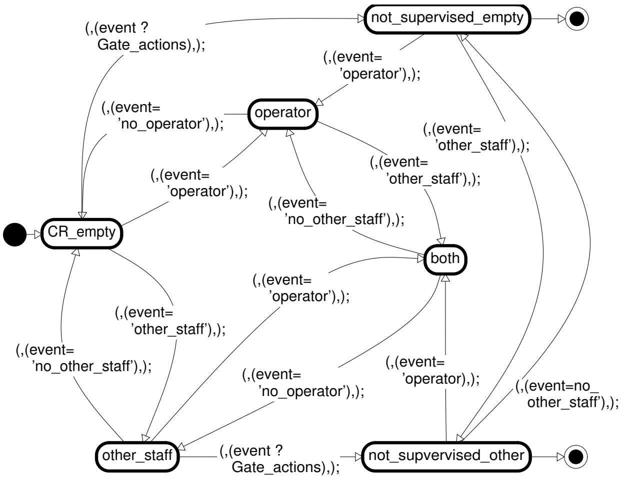


Fig. 4: Monitor automaton for hydroelectric power plant

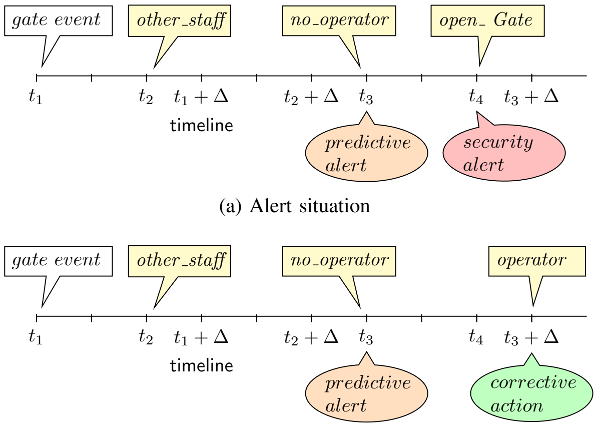
There are several attack steps. First, the disgruntled employee uses his RFID badge to enter the control room while an administrator is inside. The disgruntled employee waits until the administrator leaves the control room and uses the stolen administrator credentials to log in into the control system. Then he issues the open gate command from the control station. The water gates open discharging the dam's reservoir. The decrease of the water level endangers the people using the dam's reservoir for recreational activities.

Note, that this attack can be discovered if the system is able to detect that the administrator command was issued while no employee with the administrator role had accessed the control room with her badge.

### B. Specification of a Monitor Automaton

The security requirements that certain actions of the dam workflows have to be supervised will be controlled by a monitor automaton $\mathbb{M}$ as introduced in the previous section. Figure 4 shows a specification of $\mathbb{M}$.

The initial state of $\mathbb{M}$ *CR_empty* (control room is empty) is marked by the filled circle. The critical states *not_supervised_empty* and *not_supervised_other* are marked by the circle with the small filled circle inside. These states reflect the situation that an action from the set *Gate_actions* has been executed while the control room is either empty or only manned with non-administrative staff. If one of these states is reached during prediction an alarm will be generated. The predicates attached to the arcs of $\mathbb{M}$ define predicates for transitions of the RG. This automaton is scheduled according to the algorithm presented in the previous section during the computation of the RG in the prediction process. The



(a) Alert situation

(b) Predicted situation not occurred

Fig. 5: Security reasoning

$\lambda$ predicates in Section IV correspond to the predicates of the arcs in this automaton.

Predicates of the monitor automaton are applied to state transitions of the RG $(p_i,(e_j,i_k),q_l)$, for example, the predicate $(,(,event='no\_other\_staff'),)$ is true if $'no\_other\_staff'$ is bound to the interpretation variable *event* of the interpretation $i_k$. No condition for the predecessor and successor state $p_i$, $q_l$ and the elementary automaton $e_j$ is given in this example. The event $'no\_other\_staff'$ (all non-administrative staff left the control room) referenced in the above predicate is a higher level event generated by preprocessing the low-level events from the RFID scanners and events from cameras which capture the motion of staff at the entrance of the control room.

### C. Evaluation of State Transitions

In order to exemplify the security analysis at runtime, let us assume that the system is in a state where an operator is present in the control room, there is only one monitor automaton as shown in Fig. 4 defined, and the current state of the monitor automaton is *operator*. We now describe the reasoning process at runtime. Figure 5a shows a possible timeline of events. A *security warning* indicates a situation where a security requirement is broken but has no negative impact at creation time. A *predictive alert* is raised when a broken security requirement might lead to a security critical situation in the close future. A *security alert* is raised if a security critical situation has been detected. These warnings and alerts are mapped to corresponding events and fed into the runtime environment.

If at time $t_1$ an event from a gate function is received, then the state component of the process model representing the status of the gate will be changed but the state of the monitor automaton will not change. The reachability analysis does not "see" an upcoming security violation within the scope $\Delta$, so no alarm has to be triggered.

If at time $t_2 > t_1$ the event $'other\_staff'$ produced by the RFID scanners of the control room is received, then the state component of the process model representing the manning of the control room will be changed and the monitor automaton changes the state to *both*. No security violations is "seen" within the scope $\Delta$.

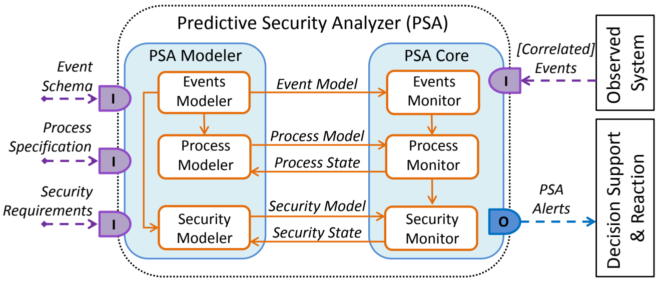


Fig. 6: Architecture of the PSA

If at time $t_3 > t_2$ the event $'no\_operator'$ is received which indicates that the last operator has left the control room, then the state component of the process model representing the manning of the control room will be changed and the monitor automaton also changes the state to *other_staff*. Now in one possible process execution sequence, an event from a gate function such as *open_Gate* is reachable within $\Delta$. In this situation the reachability analysis shows that this function would violate an associated security requirement. Therefore, a predictive alert is raised because a broken security requirement might lead to a security critical situation in the close future.

If at time $t_4 > t_3$ an event from a gate function such as *open_Gate* is received, then the monitor automaton changes to the critical state *not_supervised_other*. As a security critical situation has now been detected, a security alert is raised.

Now let us assume that at time $t_3+\Delta$ an event is received which indicates that an operator is back in the control room and the critical state was not reached as predicted. In this case, we know that the issued predictive alert did not lead to a security alert (cf. Fig. 5b). Therefore, a corrective action such as the reduction of the security warning level or lifting of restrictions on the business process may be necessary.

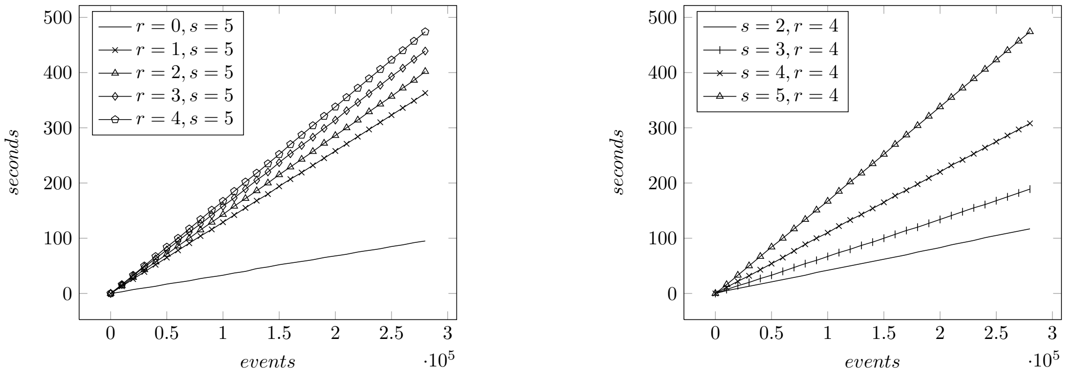
## VI. Evaluation of Security Analysis at Runtime

To evaluate the performance of different modeling strategies in the scope of PSA@R, we have implemented a prototype, the PSA, that supports the complete life-cycle of security analysis at runtime from formal process specification to exhaustive validation, including visualization and inspection of computed RGs and monitor automata. Our implementation is based on Common LISP [21] and technical specifications from [13].

### A. Prototype Architecture

Figure 6 shows the architecture of the PSA consisting of two main parts: the *PSA Modeler* that provides functionality for process formalization and the *PSA Core* that performs process security analysis. "I" and "O" are input and output interfaces.

During initialization an operator uses the PSA Modeler components to formalize input required for process simulation. The *Event Modeler* supports the derivation of an event model from given event schemata (cf. Fig. 3) and stores the respective



(a) Execution time depending on number $r$ of security requirements

(b) Execution time depending on number $s$ of successor states

Fig. 7: Execution time measurements

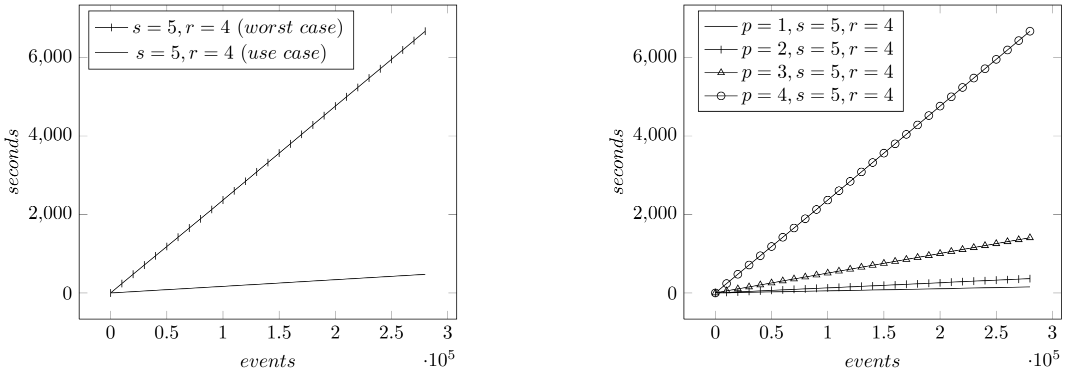
mapping for interpretation of runtime events. The *Process Modeler* allows to formalize process specifications using methods introduced in Section III. The *Security Modeler* provides means for graphical specification of monitor automata representing security models (cf. Section IV).

To launch the security analysis the PSA models with initial configurations such as the initial state of a process model and an active security model need to be loaded into the PSA Core in the form of compiled code. During the monitoring and analysis stage the PSA Core components receive runtime events from the observed system, for example, events from dam's SCADA system and RFID scanners. The *Event Monitor* interprets these events in accordance with the defined event model. The normalized events are fed to the *Process Monitor* that performs behavior prediction based on the process model. If a legitimate event does not comply with the model, the PSA supports an adjustment of the model on-the-fly within the process modeler utilizing backward references from the compiled process model. To detect security violations the information about predicted state transitions is forwarded to the *Security Monitor*. By executing the security model the Security Monitor identifies process states critical from the security perspective and issues alerts that are forwarded to decision support and reaction for further processing. Backward references within the compiled security model allow to show the current state within the security modeler.

### B. Evaluation

In the project MASSIF [22] PSA@R is currently applied to check security requirements in four industrial domains: (i) the management of the Olympic Games IT infrastructure [23]; (ii) a mobile phone based Mobile Money Transfer System (MMTS) [24], facing high-level threats such as money laundering; (iii) managed IT outsource services for large distributed enterprises and (iv) an IT system supporting a critical infrastructure (dam) [7]. We used the hydroelectric power plant scenario (iv) to demonstrate the capability of the PSA prototype to process and correlate events from heterogeneous sources (cf. Section V). To evaluate the PSA prototype with respect to performance issues, however, we used event logs from scenario (ii) as a resource intensive application which requires high throughput. In this case, events referred to transactions conducted in an MMTS and processes represented user behaviors observed from transaction logs [25]. To achieve high load a recorded event stream was sent directly to the PSA socket interface. Measurements were produced on a personal computer with Intel Core 6700 CPU and 4GB memory.

The measurements presented evaluate the execution time and the number of events received by the PSA. We have examined four aspects important from the application perspective: (i) effects of the number of security requirements to the execution time; (ii) effects of the abstraction level to analysis; (iii) effects of cycle reduction in a RG; (iv) effects of changing prediction depths. The prediction depth $p = 4$ was used in (i)–(iii), but did not effect the performance of the simulation because the complete RG could be computed in advance. Figure 7a shows that the execution time depends linearly on the number of received events and the gradient of this linear slope is determined by the number of security requirements. In order to investigate effects of the abstraction level we have evaluated finer and coarser process models. A coarser model results in less successor states and thus reduces the effort for the monitoring algorithm. The abstraction level can be adjusted, for instance, as shown in Example 2. In [8] Mendling presented an extensive metrics analysis on four collections of 2003 Event-driven Process Chain (EPC) process specifications. In this study, the number of nodes a connector is in average connected to resulted in 3.56 for the mean value $\mu$ and 2.40 for the standard deviation $\sigma$. Therefore, for our performance measurements we used a number $s \in \{2, 3, 4, 5\}$ of successor states. Figure 7b shows that the execution time depends linearly on the number of received events. The moderate increase of gradients of the corresponding linear slopes was caused by the optimization of the monitoring algorithm related to cycles in the RG. These experiments show that one month of data logged in an MMTS (285.619 events from 50.265 processes) is analyzed by the PSA within two to eight minutes depending on the model abstraction. In order to simulate the possible worst case for five successor



(a) Comparison of MMTS use case with worst case

(b) Execution time depending on prediction depth $p$

Fig. 8: Worst case behavior and influence of prediction depth

states (four requirements) we produced a synthetic model. Figure 8a displays the comparison between the MMTS model and the worst case model. The worst case performance can be improved by reasonably limiting the number of predicted steps. Figure 8b illustrates the effects of reduced prediction depth in the worst case model. During the experiment the security requirements were successfully checked in all combinations.

## VII. Related Work

The work presented here combines specific aspects of security analysis with generic aspects of process monitoring, simulation, and analysis. The background of these aspects is given by the utilization of models at runtime [26]. The proposed approach is similar to the approaches described in [17], [27] in terms of event-driven process analysis. However, in our work we focus on an integrated algorithm for computation of reachability graphs with evaluation of security properties given by monitor automata. Recently, runtime monitoring of concurrent distributed systems based on Linear Temporal Logic (LTL), state-charts, and related formalisms has also received attention [28]. However, these works are mainly focused on error detection, e.g., concurrency related bugs.

Approaches focusing on security models at runtime are given in [29], [30]. The first work proposes a novel methodology to synchronize an architectural model reflecting access control policies with the running system. Therefore, the methodology emphasizes policy enforcement rather than security analysis. The integration of runtime and development-time information on the basis of an ontology to engineer industrial automation systems is discussed in [30]. Schneider [31] analyzed a class of safety properties and related enforcement mechanisms that work by monitoring execution steps of some target system, and terminating the target's execution, whenever it is about to execute an operation, which would violate the security policy. Extensions of this approach are discussed in [32]. However, security automata as defined in [31] are related to a specific trace of execution, whereas in the monitor automata concept proposed here, the whole RG is used as a reference to the possible system's behavior. Patterns and methods to allow for monitoring security properties are developed in [33], [34], [35].

Diverse categories of tools applicable for modeling and simulation of business processes are based on different semi-formal or formal methods such as EPCs [8] or Petri nets [9]. Likewise, some general-purpose simulation tools such as CPNTools [36] were proven to be suitable for simulating business processes. The process mining framework ProM [15] supports plug-ins for different types of models and process mining techniques. However, independently from the tools and methods used, such simulation tools concentrate on statistical aspects, redesign, and commercial optimization of the business process. On the contrary, we propose an approach for *on-the-fly* dynamic simulation and analysis on the basis of operational formal models. This includes consideration of the current process state and the event information combined with the corresponding steps in the process model. We consider the framework presented in [37] on runtime compliance verification for business processes as complementary to our work.

## VIII. Conclusions and Further Work

In this paper, we presented an integrated approach called PSA@R to analyze the security status of a process and to identify possible violations of the security policy in close future. The approach also provides early awareness about deviations of a running process from expected behavior as specified by the model. When such anomalies refer to process misbehavior or disruption, alarms will be raised for decision support and reaction. Moreover, we described how to extend process behavior computation with algorithms for on-the-fly security compliance checks and prediction of close-future security violations. Thus, our integrated security analysis approach identifies current and close-future violations of the security policy. As security relies on the compliance of actual behavior with the given specifications this early detection of changes and reaction elevates security of the process in question. In combination with other novel applications PSA@R enables anticipatory impact analysis, decision support and impact mitigation by adaptive configuration of countermeasures. Moreover, we assume that our results can also be applied to on-the-fly analysis of compliance and dependability requirements. In further work, we consider to integrate methods, such as the one described in [38] using metrics to quantify deviations from process specifications.

## Acknowledgment

The presented work was developed in context of the project MASSIF (ID 257475) being co-funded by the European Commission within the Seventh Framework Programme and the project ACCEPT (ID 01BY1206D) being funded by the German Federal Ministry of Education and Research.

## References

[1] P. Wood, "Internet Security Threat Report, 2011 Trends, Vol. 17," Symantec Corporation, Technical Report, April 2012.

[2] M. Klemen, S. Biffl, and T. Neubauer, "Secure business process management: A roadmap," in *Proceedings of the First International Conference on Availability, Reliability and Security (ARES)*. IEEE, 1 2006, pp. 457–464.

[3] P. H. C. Wolf, "The state of business process management," http://www.bptrends.com/, BPTrends Report, 2012.

[4] P. Tallon, "Inside the adaptive enterprise: an information technology capabilities perspective on business process agility," *Information Technology and Management*, vol. 9, no. 1, pp. 21–36, 2008.

[5] S. Rinderle-Ma, M. Reichert, and B. Weber, "Relaxed compliance notions in adaptive process management systems," in *Proceedings 27th Int'l Conference on Conceptual Modeling (ER'08)*, ser. LNCS, no. 5231. Springer, October 2008, pp. 232–247.

[6] D. Schumm, F. Leymann, Z. Ma, T. Scheibler, and S. Strauch, "Integrating compliance into business processes: Process fragments as reusable compliance controls," in *Proceedings of the Multikonferenz Wirtschaftsinformatik, (MKWI 2010)*. Universitätsverlag Göttingen, 2010.

[7] L. Romano, S. D. Antonio, V. Formicola, and L. Coppolino, "Enhancing SIEM technology to protect critical infrastructures," in *CRITIS 2012, the seventh CRITIS Conference on Critical Information Infrastructures Security*, September 2012.

[8] J. Mendling, *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, ser. LNBIP. Springer, 2008, vol. 6.

[9] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[10] W. Arsac, L. Compagna, G. Pellegrino, and S. Ponta, "Security Validation of Business Processes via Model-Checking," in *Engineering Secure Software and Systems (ESSoS 2011)*, ser. LNCS. Springer, 2011, vol. 6542, pp. 29–42.

[11] B. Fabian, S. Gürses, M. Heisel, T. Santen, and H. Schmidt, "A comparison of security requirements engineering methods," *Requirements engineering*, vol. 15, no. 1, pp. 7–40, 2010.

[12] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Comput. Stand. Interfaces*, vol. 32, no. 4, 2010.

[13] P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche, "The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems," *Formal Aspects of Computing*, vol. 10, no. 4, pp. 381–404, 1998.

[14] M. Weber and E. Kindler, "The petri net markup language," in *Petri Net Technology for Communication-Based Systems*, ser. LNCS. Springer, 2003, vol. 2472, pp. 124–144.

[15] W. M. P. van der Aalst, B. F. van Dongen, C. Günther, A. Rozinat, H. M. W. Verbeek, and A. J. M. M. Weijters, "ProM: The Process Mining Toolkit," in *BPM 2009 Demonstration Track*, vol. 489. CEUR, 2009, pp. 1–4.

[16] D. A. Peled, *Software Reliability Methods*, 1st ed. Springer, 2001.

[17] J. Eichler and R. Rieke, "Model-based Situational Security Analysis," in *Workshop on Models@run.time*. CEUR, 2011, vol. 794, pp. 25–36.

[18] H. R. Motahari-Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah, "Event correlation for process discovery from web service interaction logs," *The VLDB Journal*, vol. 20, no. 3, pp. 417–444, Jun. 2011.

[19] L. Coppolino, M. Jäger, N. Kuntze, and R. Rieke, "A Trusted Information Agent for Security Information and Event Management," in *ICONS 2012, The Seventh International Conference on Systems*. IARIA, 2012, pp. 6–12.

[20] M. E. Luallen, "Managing Insiders in Utility Control Environments," SANS, A SANS Whitepaper in Association with SANS SCADA Summits, Q1, 2011, March 2011.

[21] G. L. Steele, Jr., *Common LISP: the language (2nd ed.)*. Newton, MA, USA: Digital Press, 1990.

[22] R. Rieke, E. Prieto, R. Diaz, H. Debar, and A. Hutchison, "Challenges for advanced security monitoring – the MASSIF project," in *Trust, Privacy and Security in Digital Business*, ser. LNCS, S. Fischer-Hübner, S. Katsikas, and G. Quirchmayr, Eds. Springer, 2012, vol. 7449, pp. 222–223.

[23] E. Prieto, R. Diaz, L. Romano, R. Rieke, and M. Achemlal, "MASSIF: A promising solution to enhance olympic games it security," in *Global Security, Safety and Sustainability & e-Democracy*, ser. LNICST, C. K. Georgiadis *et al.*, Eds. Springer, 2012, vol. 99.

[24] C. Gaber, B. Hemery, M. Achemlal, M. Pasquet, and P. Urien, "Synthetic logs generator for fraud detection in mobile transfer services," in *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems (CTS2013)*, 2013.

[25] R. Rieke, M. Zhdanova, J. Repp, R. Giot, and C. Gaber, "Fraud detection in mobile payment utilizing process behavior analysis," in *Proceedings of 2013 International Conference on Availability, Reliability and Security, ARES 2013*. IEEE Computer Society, 2013, pp. 662–669.

[26] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering*. IEEE, 2007, pp. 37–54.

[27] R. Rieke and Z. Stoynova, "Predictive security analysis for event-driven processes," in *Computer Network Security*, ser. LNCS. Springer, 2010, vol. 6258, pp. 321–328.

[28] T. Massart and C. Meuter, "Efficient online monitoring of LTL properties for asynchronous distributed systems," Université Libre de Bruxelles, Tech. Rep., 2006.

[29] B. Morin, T. Mouelhi, F. Fleurey, Y. Le Traon, O. Barais, and J.-M. Jézéquel, "Security-driven model-based dynamic adaptation," in *Automated Software Engineering (ASE 2010)*. ACM, 2010, pp. 205–214.

[30] M. Melik-Merkumians, T. Moser, A. Schatten, A. Zoitl, and S. Biffl, "Knowledge-based runtime failure detection for industrial automation systems," in *Workshop Models@run.time*. CEUR, 2010, pp. 108–119.

[31] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30–50, 2000.

[32] F. Martinelli, I. Matteucci, and C. Morisset, "From qualitative to quantitative enforcement of security policy," in *MMM-ACNS*, ser. LNCS, I. V. Kotenko and V. A. Skormin, Eds., vol. 7531. Springer, 2012, pp. 22–35.

[33] C. Serban and B. McMillin, "Run-time security evaluation (RTSE) for distributed applications," in *Symposion on Security and Privacy*. IEEE, 1996, pp. 222–232.

[34] T. Tsigritis and G. Spanoudakis, "Diagnosing runtime violations of security & dependability properties," in *Software Engineering and Knowledge Engineering (SEKE 2008)*. KSI, 2008, pp. 661–666.

[35] A. Evesti, E. Ovaska, and R. Savola, "From security modelling to run-time security monitoring," in *European Workshop on Security in Model Driven Architecture (SECMDA 2009)*. CTIT, 2009, pp. 33–41.

[36] A. Rozinat, M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, and C. J. Fidge, "Workflow simulation for operational decision support," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 834–850, 2009.

[37] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *Business Process Management (BPM 2011)*, ser. LNCS, vol. 6896. Springer, 2011, pp. 132–147.

[38] S. Banescu and N. Zannone, "Measuring privacy compliance with process specifications," in *Workshop on Security Measurements and Metrics (MetriSec 2011)*. IEEE, 2011.

Part IV

# APPENDIX

*Count what is countable, measure what is measurable, and what is not measurable, make measurable.*

— Galileo Galilei

# A DECLARATION

Ich versichere, dass ich meine Dissertation “Security Analysis of System Behaviour – From ’Security by Design’ to ’Security at Runtime’ –” selbständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe. Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient.

*Griesheim,*

Roland Rieke