UNIVERSITÉ DU LUXEMBOURG

PhD-FSTC-2015-38

The Faculty of Science, Technology and Communication

Dissertation

Defence held on 10/09/2015 in Luxembourg

to obtain the degree of

Docteur de l'Université du Luxembourg

en Informatique

by

Phu Hong Nguyen

Born on 23/06/1983 in Hanoi (Vietnam)

Model-Driven Security With Modularity and Reusability For Engineering Secure Software Systems

Dissertation Defence Committee

Dr-Ing. Yves Le Traon, Dissertation Supervisor
Professor, University of Luxembourg, Luxembourg

Dr. Pierre Kelsen, Chairman
Professor, University of Luxembourg, Luxembourg

Dr. Jacques Klein, Vice Chairman
Senior Research Scientist, University of Luxembourg, Luxembourg

Dr. Jörg Kienzle
Associate Professor, McGill University, Montreal, Canada

Dr. Riccardo Scandariato
Associate Professor, Chalmers University of Technology and University of Gothenburg, Sweden

Copyright

Published by the University of Luxembourg

Typeset in LaTeX

“Anything will give up its secrets if you love it enough...”

George Washington Carver

“If you have to kiss a lot of frogs to find a prince, find more frogs and kiss them faster and faster.”

Ron Kohavi

“In theory, theory and practice are the same. In practice, they are not.”

Albert Einstein

“Life is like riding a bicycle. To keep your balance you must keep moving.”

Albert Einstein

“I think of life as a good book. The further you get into it, the more it begins to make sense.”

Harold S. Kushner

“If you want to go fast, go alone. If you want to go far, go together”

African Proverb

Abstract

Context: The more human beings depend on software systems, the more important role that software security engineering must play to build secure software systems. Model-Driven Security (MDS) emerged more than a decade ago as a specialised Model-Driven Engineering (MDE) research area for engineering secure software systems. MDS is promising but not mature yet. Our recent systematic literature review (SLR) has revealed several current limitations and open issues in the state of the art of MDS research.

Objectives: This PhD work aims at addressing three of the main open issues in the current state of the art of MDS research that are pointed out by the SLR. First, our SLR shows that multiple security concerns need to be handled together more systematically. Second, true Aspect-Oriented Modelling techniques for better ensuring the separation-of-concern in MDS approaches could have been leveraged more extensively. Third, complete tool chains based on integrated MDE techniques covering all the main stages of the development cycle are emerging, but still very rare.

Methods: On one hand, we develop a full MDS framework with modularity based on domain-specific modelling, model transformations, and model-based security testing. This MDS framework can help us to deal with complex delegation mechanisms in access control administration, from modelling till testing. On the other hand, we propose a highly modular, reusable MDS solution based on a *System of Security design Patterns* (SoSPa) and reusable aspect models to tackle multiple security concerns systematically.

Results: First, an extensive SLR has been conducted for revealing and analysing the current state of the art of MDS research. Second, a full MDS framework focusing on modularity has been proposed that integrates domain-specific modelling, model transformations, and model-based security testing to support all the main stages of an MDS development cycle. Third, we have developed a highly reusable, modular MDS approach based on a *System of Security design Patterns* for handling multiple security concerns together systematically. Finally, we have showed how our MDS approaches can be integrated in a full MDS framework, called MDS-MoRe, which could be the basis of a complete tool chain for MDS development of secure systems.

Conclusion: In this thesis, integrated MDS methodologies with modularity and reusability have been proposed for engineering secure software systems. This work has tackled three main current open issues in MDS research revealed from an extensive SLR.

Keywords: Model-Driven Security, MDS, MDE, Aspect-Oriented Modelling, RAM, DSL, Model Transformations, Model Composition, Systematic Review, Security Design Patterns, Pattern Refinement, Security By Design, Security Testing.

Acknowledgements

In my opinion, one way to look at doing a PhD could be described by the African proverb "If you want to go fast, go alone. If you want to go far, go together". My personal commitment to the PhD work is important for its progress but without the support and collaboration from many people, I could not make it that "far" to finish the PhD work and complete this PhD thesis. I would like to take the opportunity to thank all of them, knowing that it is not feasible to name everyone.

My PhD work is supported by the Fonds National de la Recherche (FNR) Luxembourg, under the project C10/IS783852/ MITER. I do remember the day when I had the Skype interview for a PhD position with Prof. Yves Le Traon and Dr. Jacques Klein. The interview was quite short and mainly involved pleasant discussion. I was a bit surprised that I got accepted to the PhD position right after the interview via email. There could be many reasons but one thing among others that I do appreciate is the trust that I have got from Yves and Jacques from day one and through out my PhD work. I believe that many of Yves and Jacques' PhD students always feel the encouragement, trust, and empowerment in the team. Dear Yves, thank you very much for your great kindness, supervision, trust, and leadership for the team and myself. You do care not only about PhD student's work but also personal life. I am always happy about the gift from the team and your personal gift for my son Nem when he was born more than two years ago, as the first child who was born in Serval team. At the same time, we received another good news when our paper (the first one of my PhD work) got accepted at the AOSD conference thanks to one of the ideas you and Jacques suggested to me. I am also very thankful to have Jacques as my scientific advisor and mentor. My PhD work is part of an interesting yet challenging research project that Jacques is the principal investigator. I did benefit a lot from his expertise in the field and his excellent advices for my work. Jacques was always there whenever I needed to discuss any idea. One of the descriptions for the PhD position is the good facilities for doing a PhD in Luxembourg and also a great chance to take part in shaping a very young university as well as a research centre. It is true. I do enjoy being part of such a fast growing university and great research centre. It is my honour to have joined with Jacques in the University Council, as a representative of students. Jacques is my mentor not only scientifically but generally.

I am grateful to have collaborated with quite a number of people during my PhD work, not only in Serval research group but also outside of Luxembourg. In Luxembourg, besides Yves and Jacques, I have co-authored with Tejeddine Mouelhi, Gregory Nain, Mike Papadakis, Moussa Amrani, Iram Rubab, Qin Zhang in some publications. Outside of Luxembourg, it is my great pleasure to have joined work with Max Kramer (Karlsruhe Institute of Technology-Germany), Levi Lucio (McGill University-Canada), Hans

Vangheluwe (University of Antwerp-Belgium), Riccardo Scandariato (Chalmers Technical University and Göteborg University-Sweden, KU Leuven-Belgium), Koen Yskout (KU Leuven-Belgium), and Thomas Heyman (KU Leuven-Belgium). I would like to thank Prof. Wouter Joosen, Dr. Riccardo Scandariato for hosting me at KU Leuven as a visiting scholar. During my stay at KU Leuven, I did benefit from gaining more experience in research collaboration and had very promising shared work with Koen Yskout, Thomas Heyman, and Riccardo Scandariato.

In my PhD defence committee, it is my great honour to have Prof. Pierre Kelsen as the chairman, and Prof. Jörg Kienzle, Prof. Riccardo Scandariato as the external reviewers. Thank you all for your time and effort to review this thesis and examine my PhD work. I also thank Pierre, Yves, and Jacques for the very useful annual CET meetings for monitoring the progress of my PhD work. Many thanks to Claudia Thür for the great administrative support and arrangements for my PhD defence.

I would like to thank all the colleagues and friends at SnT, at the University of Luxembourg, and other friends in Luxembourg. Many thanks to the SerValers for our interesting discussions, group meetings, and frequent coffee breaks. I find it an excellent tradition of our SerVal team to celebrate any good news (e.g. newly accepted publications) by buying croissants for our coffee breaks. I am thankful to have wonderful Vietnamese friends in Luxembourg who could be as close as my family members. Besides, I appreciate very much the pleasant collaboration with Martin Kracheel, and the great support from the SnT officials, such as Prof. Björn Ottersten (Director of SnT), Mrs. Marie-France Gallo for the activities of SnT PhD student Association. Moreover, it has been my great honour and pleasure to be part of the committee of LuxDoc (the organisation of young researchers in Luxembourg), and the Toastmasters in Luxembourg.

Last but most important is the great support from my family, especially during my PhD work. It is the constant encouragement from my father for me to keep studying at all the levels of education. I am thankful to have my sister who has taken care of my father while I am pursuing higher education abroad. She is an image of my late Mother who had always dedicated everything to her family. I also appreciate the support from my parents-in-law who went all the way twice from Vietnam to Luxembourg to help when my children were born. Finally, I cannot express how grateful I am to have the countless support from my wife Minh-Ngoc Phung. It is her love, understanding, and devotion that allowed me to concentrate on my PhD work. Giving births and taking care of our two sons during my PhD study can say all for her hard work and countless support for me. Thank you, my dear wife and my lovely little sons Gia-Minh (Nem), Quoc-Gia (Cha) for being with me during a difficult but still full of happiness time.

Contents

List of Figures

List of Tables

Abbreviations

3GLs	**3**rd-generation **P**rogramming **L**anguages
AC	**A**ccess **C**ontrol
AOM	**A**spect-**O**riented **M**odelling
AOSD	**A**spect-**O**riented **S**oftware **D**evelopment
ASMS	**A**uction **S**ale **M**anagement **S**ystem
ATL	**A**tlas **T**ransformation **L**anguage
CCCMS	**C**ar **C**rash **C**risis **M**anagement **S**ystem
CIA	**C**onfidentiality, **I**ntegrity, and **A**vailability
CMS	**C**risis **M**anagement **S**ystem
CPS	**C**yber-**P**hysical **S**ystem
CVL	**C**ommon **V**ariability **L**anguage
CWE	**C**ommon **W**eakness **E**numeration
DAC	**D**iscretionary **A**ccess **C**ontrol
DEBAC	**D**ynamic **E**vent-**B**ased **A**ccess **C**ontrol
DSL	**D**omain **S**pecific **L**anguage
DSM	**D**omain **S**pecific **M**odelling
DSML	**D**omain **S**pecific **M**odelling **L**anguage
EMF	**E**clipse **M**odeling **F**ramework
GMF	**G**raphical **M**odeling **F**ramework
GPLs	**G**eneral-purpose **P**rogramming **L**anguages
LMS	**L**ibrary **M**anagement **S**ystem
MAC	**M**andatory **A**ccess **C**ontrol
MAPE	**M**onitoring, **A**nalysis, **P**lanning, and **E**xecution
MBE	**M**odel-**B**ased **E**ngineering
MBT	**M**odel-**B**ased **T**esting

MD	**M**odel-**D**riven
MDA	**M**odel-**D**riven **A**rchitecture
MDD	**M**odel-**D**riven **D**evelopment
MDE	**M**odel-**D**riven **E**ngineering
MDS	**M**odel-**D**riven **S**ecurity
MDS-MoRe	**M**odel-**D**riven **S**ecurity with **Mo**dularity and **Re**usability
MLD	**M**aster-**L**evel **D**elegation policy
MMT	**M**odel-to-**M**odel **T**ransformation
MOF	**M**eta-**O**bject **F**acility
MPM	**M**ulti-**P**aradigm **M**odelling
MTT	**M**odel-to-**T**ext **T**ransformation
OCL	**O**bject **C**onstraint **L**anguage
OMG	**O**bject **M**anagement **G**roup
OrBAC	**Or**ganization-**B**ased **A**ccess **C**ontrol
OTP	**O**ne **T**ime **P**assword
PIM	**P**latform-**I**ndependent **M**odel
PSM	**P**latform-**S**pecific **M**odel
QVT	**Q**uery/**V**iew/**T**ransformation
RAM	**R**eusable **A**spect **M**odel
RBAC	**R**ole-**B**ased **A**ccess **C**ontrol
SLR	**S**ystematic **L**iterature **R**eview
SOA	**S**ervice-**O**riented **A**rchitecture
SOC	**S**eparation **of** **C**oncerns
SoSPa	**S**ystem **of** **S**ecurity (design) **Pa**tterns
TCP	**T**ransmission **C**ontrol **P**rotocol
ULD	**U**ser-**L**evel **D**elegation policy
UML	**U**nified **M**odelling **L**anguage
UWE	**U**ML-based **W**eb **E**ngineering
VMS	**V**irtual **M**eeting **S**ystem
XACML	e**X**tensible **A**ccess **C**ontrol **M**arkup **L**anguage
XML	**E**xtensible **M**arkup **L**anguage

Dedicated to my father Nguyen Duc Mai and my late mother Duong Thi Dinh, who made many sacrifices for their children, especially for my academic pursuit. Unfortunately, because of cancer my mother could not see that her children have grown up. But surely, she will always be remembered...

Chapter 1

Introduction

Contents

Nowadays, the security of software systems is getting more important than ever. With the progress of the digital age, software systems play an essential role in our daily lives. Indeed, software has become ubiquitous. We are relying on (mostly network-connected) software in so many places from military systems, governmental systems, banking systems, airlines systems, airplanes, trains, cars, to mobile phones on our hands. Moreover, with the cyber-physical systems (CPSs) are becoming more popular, the security of CPSs also plays a big role in the physical safety of human-beings around these systems. However, the security of computer systems and networks cannot be ensured by only enhancing network security and other perimeter solutions. It is also essential for ensuring security by building better, secure software [149].

Building complex, secure software is very hard. Software systems are getting much more complex, and seriously facing many more security challenges. As a consequence, software systems are not always secure and reliable. Few days pass without new stories in the newspapers about software vulnerabilities exploited by attackers. This would be only the visible part of an iceberg. So far, more or less a thousand of software weaknesses have been listed by the Common Weakness Enumeration (CWE)[1] as the causes of software vulnerabilities exploited by attackers. Thus, more innovative, sound methodologies for developing modern secure software systems are vital.

[1] http://cwe.mitre.org/

The main technical motivation for this PhD work is showed in Section 1.1. Derived from the motivation, in Section 1.2 our research objectives and questions are given. Section 1.3 presents our research proposals to answer the research questions. The main contributions of this thesis are listed in Section 1.4. Finally, Section 1.5 outlines the main content of this thesis.

1.1 Motivation

With the progress of the digital age, software security engineering is becoming more important than ever for building secure software systems. Technically, there could be three main challenges for modern software security engineering. First, the complexity of software systems and evolving security threats are irrefutably going up together. In fact, taking into account security concerns while developing (already complex) systems makes the development process more stressful, error-prone, and difficult. Second, although the complexity of systems being developed and maintained is continuously increasing, economic pressure often reduces the development time and increases the frequency of demanded modifications. The development process has to be more productive and at the same time more systematic, reliable, especially for developing secure software systems. The importance of security by design is gaining much more awareness by both industry and academia, especially regarding the following third challenge. Third, in many cases, security apparently was not systematically engineered into the software systems but came as an after-thought. Security requirements are often scattered among functional requirements. While security threats are becoming more dangerous, varied, and evolving, security requirements must evolve accordingly, and often getting more complex. Therefore, they can hardly be integrated properly into the traditional software development process. As a consequence, many security weaknesses, which have been exploited in practice, already made the headlines of the newspapers. All these issues urge for more timely, innovative, and sound methodologies for better supporting the development of reliably secure software systems.

Model-Driven Security (MDS) emerged more than a decade ago as a specialised *Model-Driven Engineering* (MDE) approach for supporting the development of secure software systems. MDE has been considered by some researchers as a solution to the handling of complex and evolving software systems [38]. MDE leverages models and transformations as main artefacts at every development stage. MDS is the common term of scientific approaches for the model-driven development of secure systems. MDS specialises MDE by taking into account security requirements with functional requirements, from very beginning, and in every stage of the development process. MDS does not only take into consideration the notion of security by design but also many benefits of MDE such as high-level abstraction and productivity. By modelling the desired system and manipulating models, the level of abstraction is higher than code-level that brings several significant benefits, especially regarding security engineering. First, security concerns are considered (modelled, manipulated) together with the business logic (and other quality attributes like performance) from the very beginning, and throughout the MDS development life cycle. In this way, security requirements are considered early, and implemented more properly in the resulting secure system. Second, reasoning about the desired systems at the model level would encourage the adoption of model-based verification and validation methods. Model-based verification and validation methods are getting more mature to ensure the correctness of the functional properties as well as the quality properties such as the security properties of a system. Formal methods such as model checking and model-based analysis could be employed for verifying security properties. Model-based security testing methods could be employed for validating the resulting secure systems (especially in where formal methods would not be applicable). Moreover, using models at a higher-level than the final target platform and independently from business functionality enables platform independence as well as cross-platform interoperability. Third, MDS is productive, and supposedly less error-prone than traditional development methods by leveraging on MDE automation provided by automated model-to-model transformations (MTTs) and model-to-text transformations (MTTs, code generation). All these points show that MDS could be the solution to all the challenges for developing modern secure systems mentioned before.

From the beginning of the 21st century until now, the very first MDS publications followed by a considerable number of MDS research papers published have shown the great attention of research community to this area, knowing that security getting more crucial. MDS leverages the key advantages of MDE, in combination with security engineering techniques, for tackling the challenges and advancing methodologies for developing secure systems. MDS is promising but not yet mature. Our recent systematic literature review (SLR) of MDS reveals the main limitations and open issues in the current state of the art of MDS research [182], [183]. This PhD work aims at addressing three of the current main open issues in MDS research. First, the SLR shows that multiple security concerns have not been tackled systematically by the existing MDS studies. Developing modern secure systems must always address multiple security concerns together to minimise different security leaks, and to make these systems resilient to different security attacks. Second, leveraging Aspect-Oriented Modelling (AOM) techniques for better ensuring the separation-of-concern in MDS approaches is not popular. Enhancing separation-of-concern in the development process, between engineering security and engineering main system functionalities, is important especially for developing complex software systems. Third, complete tools chains based on integrated MD methodologies covering all the main stages of an MDS development cycle are emerging but still very rare. As a model-driven software development methodology, MDS can only achieve its full potential strengths and be more adoptable by industry if tool chains supporting the MDS development processes are available. Addressing those three open issues could advance the MDS research area.

1.2 Research Questions

This PhD work aims at advancing the current MDS research area by tackling (some of) its main open issues in the state of the art. We formulate the central research questions to be discoursed in this thesis as follows.

RQ: *How to advance the current* MDS *research?*

We decompose this central research question in more detailed research questions. To advance the current MDS research, we first need to know its existing limitations or open issues.

RQ0: *What are the current limitations and open issues in the state of the art of* MDS *research?*

After identifying the current limitations and open issues, new MDS methodologies are to be proposed to tackle them.

RQ1: *How can the current limitations and open issues in the state of the art of* MDS *research be addressed? In other words, what new* MDS *methodologies can be introduced to address (some of) them?*

More specifically, this PhD work particularly focuses on addressing three of the main open issues mentioned in the previous section. Thus, RQ1 is detailed into three following subquestions.

RQ1.1: *How to address multiple security concerns more systematically?*

RQ1.2: *How to leverage* AOM *techniques to better enhance separation-of-concern in* MDS *development processes?*

RQ1.3: *How to build a tool chain which is based on integrated* MDE *techniques covering all the main stages of an* MDS *development cycle?*

Moreover, we are interested in some specific key artefacts of MDS, namely security modelling, composing, and testing in an MDS development cycle. We want to find out how MDE techniques (modelling, model composition, model-based testing) could be leveraged in new MDS methodologies.

RQ2: *How can* MDE *techniques such as modelling, model composition, model-based testing be leveraged in new* MDS *methodologies for addressing the current limitations and open issues of* MDS *research?*

RQ2.1: *How can modelling techniques be used for specifying multiple security concerns systematically?*

More specifically, how can multiple security concerns be modelled systematically, in particular without any consideration of a target model i.e. the model in which the security models will be composed?

RQ2.2: *How can model composition techniques be employed for composing the security models with the target system model?*

How can (a subset of selected) security models be automatically, systematically composed with the target model to obtain a new model of the system augmented of security properties?

RQ2.3: *How can model-based security testing techniques be applied to facilitate the validation of the resulting secure systems?*

How can the security enforced model be tested against security requirements, by construction?

Thus, while finding the answers for RQ1 and its subquestions, we also construct these answers to answer RQ2 and its subquestions. In other words, we design our research to find the answers for both RQ1 and RQ2 at the same time.

1.3 Proposed Research Roadmap

To answer RQ0, at least a survey on the state of the art of MDS research must be conducted. Normally, a survey can be conducted by reviewing the related work that is well known in the field of MDS. More than a normal survey, we propose to conduct our review in a systematic way which results in a systematic literature review (SLR) of MDS. A SLR is more than a normal survey because it has a predefined review protocol with clear selection criteria, evaluation criteria, and a systematic process of extracting, synthesising, analysing data for answering the review questions. We follow the common guideline for conducting SLR in Software Engineering by Kitchenham [129] et al. Moreover, our SLR [183] is the first in the field of Software Engineering that combines a snowballing strategy [235] with database searching which results in an extensive SLR of MDS.

To answer RQ1 and RQ2, we conduct our research in two main parts. On one hand, we develop a full MDS framework with modularity based on domain-specific modelling (DSM), model transformations (model-to-model transformation MMT, and model-to-text transformation MTT), and model-based security testing. This MDS framework allows us to deal with complex delegation mechanisms in access control administration, from modelling till testing. By focusing on modularity, our MDS framework enables the (component-based) secure systems to evolve dynamically.

On the other hand, we propose an MDS solution to tackle multiple security concerns together systematically. Our MDS approach is based on a *System of Security design Patterns* (SoSPa). In SoSPa, security design patterns are collected, specified as reusable aspect models (RAM) [122] to form a coherent system of them that guides developers in systematically addressing multiple security concerns at the same time. Specifically, SoSPa consists of not only a catalog of security design patterns dealing with multiple security concerns, but also interrelations among security patterns. With SoSPa, we do promote not only modularity but also reusability in the MDS research.

Finally, we show how our MDS approaches can be integrated in a full MDS framework with modularity and reusability, called MDS-MoRe, which can support all the main stages of an MDS development cycle.

1.4 Contributions and Publications

The main contributions of this PhD work are as follows. They are grouped according to the main points in the proposed research roadmap. The corresponding publications are also indicated. The greater part of the work in each publication is directly attributed to the PhD candidate (except [144] that is partly presented in Sections 2.2, 2.3 and briefly in Appendix B).

1.4.1 An Extensive Systematic Review of MDS

First, an extensive SLR has been conducted for revealing and analysing the current state of the art of MDS research (see Chapter 3). The main contributions of this work are: (1) the detailed and condensed results on key MDS artefacts of all identified primary MDS publications; (2) a diagnosis of the limitations and open issues of current MDS research with suggestions for potential MDS research directions; (3) a classification of principal and emerging/less common MDS approaches; and 4) some trend analyses of MDS research.

This work has been reported in a conference paper [182] and an extended journal paper [183].

- **Phu Hong Nguyen**, Jacques Klein, Yves Le Traon, and Max E. Kramer. “A Systematic Review of Model-Driven Security.” In the 20th Asia-Pacific Software Engineering Conference (APSEC, 2013), vol. 1, pp. 432-441. IEEE, 2013. [182]
- **Phu Hong Nguyen**, Max E. Kramer, Jacques Klein, and Yves Le Traon. “An Extensive Systematic Review on the Model-Driven Development of Secure Systems.” In Information and Software Technology, 2015. [183]

Additionally, a detailed analysis on some of the most popular MDS approaches has been performed [144]. This work complements for the SLR because it provides a thorough analysis on some specific well-known MDS approaches revealed in the SLR (see Appendix B). The main contributions of this work are: (1) a comprehensive taxonomy for MDS; and (2) a thorough evaluation and discussion of some of today’s most relevant MDS approaches. We have published this work in a book chapter [144].

- Levi Lucio, Qin Zhang, **Phu Hong Nguyen**, Moussa Amrani, Jacques Klein, Hans Vangheluwe, and Yves Le Traon. “Advances in Model-Driven Security.” Advances in Computers 93 (2014): 103-152. [144]

1.4.2 MDS with Modularity

A full MDS framework with modularity has been proposed that integrates domain-specific modelling (DSM), model transformations (model-to-model transformation MMT, and model-to-text transformation MTT), and model-based security testing to support all the main stages of an MDS development cycle (see Chapter 5). The main contributions of this work are: (1) DSLs for separately modelling access control, delegation and business logic as separate concerns; (2) security enforcement by leveraging automated model transformation/composition (from security model to architecture model); (3) a mutation analysis approach to test the security policy enforcement.

The results of this work have been published in a conference paper [190], an extended journal paper [184], and a workshop paper [191].

- **Phu Hong Nguyen**, Gregory Nain, Jacques Klein, Tejeddine Mouelhi, and Yves Le Traon. "Model-driven adaptive delegation." In Proceedings of the 12th annual international conference on Aspect-oriented software development, pp. 61-72. ACM, 2013. [190]

- **Phu Hong Nguyen**, Gregory Nain, Jacques Klein, Tejeddine Mouelhi, and Yves Le Traon. "Modularity and Dynamic Adaptation of Flexibly Secure Systems: Model-Driven Adaptive Delegation in Access Control Management." In Transactions on Aspect-Oriented Software Development XI, pp. 109-144. Springer Berlin Heidelberg, 2014. [184]

- **Phu Hong Nguyen**, Mike Papadakis, and Iram Rubab. "Testing Delegation Policy Enforcement via Mutation Analysis." In Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on, pp. 34-42. IEEE, 2013. [191]

1.4.3 MDS with Reusability

We propose an MDS approach based on a *System of Security design Patterns* (SoSPa) for addressing multiple security concerns together systematically (see Chapter 6). The main contributions of this work are: (1) hierarchical reusable aspect models (RAM) with a refinement process for specifying security design patterns from abstract level till detailed design level; (2) explicitly specified interrelations among security design patterns for systematically dealing with multiple security concerns; (3) an MDS framework supporting secure systems development based on SoSPa.

The results of this work were published in [181] and [185]. In this work, our approach has provided not only the modularity but also a highly reusable solution in MDS.

- **Phu Hong Nguyen**, Jacques Klein, and Yves Le Traon. "Model-Driven Security with A System of Aspect-Oriented Security Design Patterns." In 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling. 2014. [181]

- **Phu Hong Nguyen**, Koen Yskout, Thomas Heyman, Jacques Klein, Riccardo Scandariato, and Yves Le Traon. "SoSPa: A System of Security Design Patterns for Systematically Engineering Secure Systems." In ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems. 2015. [185]

Finally, we showed how our MDS approaches can be integrated in a full MDS framework, called MDS-MoRe, which could be the basis for building complete tool chain(s) supporting the MDS development of secure systems (see Chapter 4).

1.5 Thesis Outline

The remainder of this thesis is structured as follows.

Chapter 2 provides background concepts and terminologies which are used in this thesis. Specifically, brief introductions to software security engineering, model-driven software engineering, model-driven security, and model-based security testing are given. An informal introduction to MDS can be found in Appendix A.

Chapters 3 to 6 contain the main contributions of this thesis. In Chapter 3, the current state of the art of MDS is analysed via an extensive systematic literature review (SLR) of MDS and a thorough survey on advances in MDS. Our SLR is the first in the field of Software Engineering that combines a snowballing strategy with database searching. The results show the identified primary MDS studies, the overall status of the key artefacts of MDS, and some trends analysis of MDS research. Based on the insights from our review results, we suggest some potential MDS research directions that have been partly taken in this thesis. Additionally, our survey on advances in MDS gives more details on some of the most well-known MDS studies (see Appendix B).

Chapter 4 introduces our unified approach, MDS-MoRe, for model-driven development of secure systems focusing on modularity and reusability. The architecture of MDS-MoRe framework and the processes in the framework are presented in this chapter.

Chapter 5 presents a full MDS approach with modularity, from modelling to testing, for developing secure systems enabling dynamic adaptation. Our proposed modular model-driven framework can be used for 1) specifying access control, delegation and business logic as separate concerns; 2) dynamically enforcing/weaving access control policies with various delegation features into security-critical systems; and 3) providing a flexibly dynamic adaptation strategy. With modularity and dynamic adaptation, this approach allows modern secure systems to adapt and evolve at runtime. More details on platform-dependent implementations can be found in Appendix C. Besides, we also propose to adopt mutation analysis in testing the security of the constructed systems. A set of mutation operators is used to introduce mutants into security policies and thus, enable mutation testing.

Chapter 6 describes our MDS approach for enhancing not only modularity but also reusability based on a *System of Security design Patterns* (SoSPa). In SoSPa, security design patterns are collected, specified as reusable aspect models to form a coherent system of them that guides developers in systematically addressing multiple security concerns. SoSPa consists of not only interrelated security design patterns but also a refinement process towards their application. After reading Chapter 6, readers interested in more details of our unified *System of Security design Patterns* can find more in Appendix D.

Finally, Chapter 7 summarises and discusses the approaches proposed in the previous chapters for the model-driven development of secure systems. Based on the discussion, possible directions for future work are given.

The recommended flow for reading this thesis is given in Fig. 1.1.

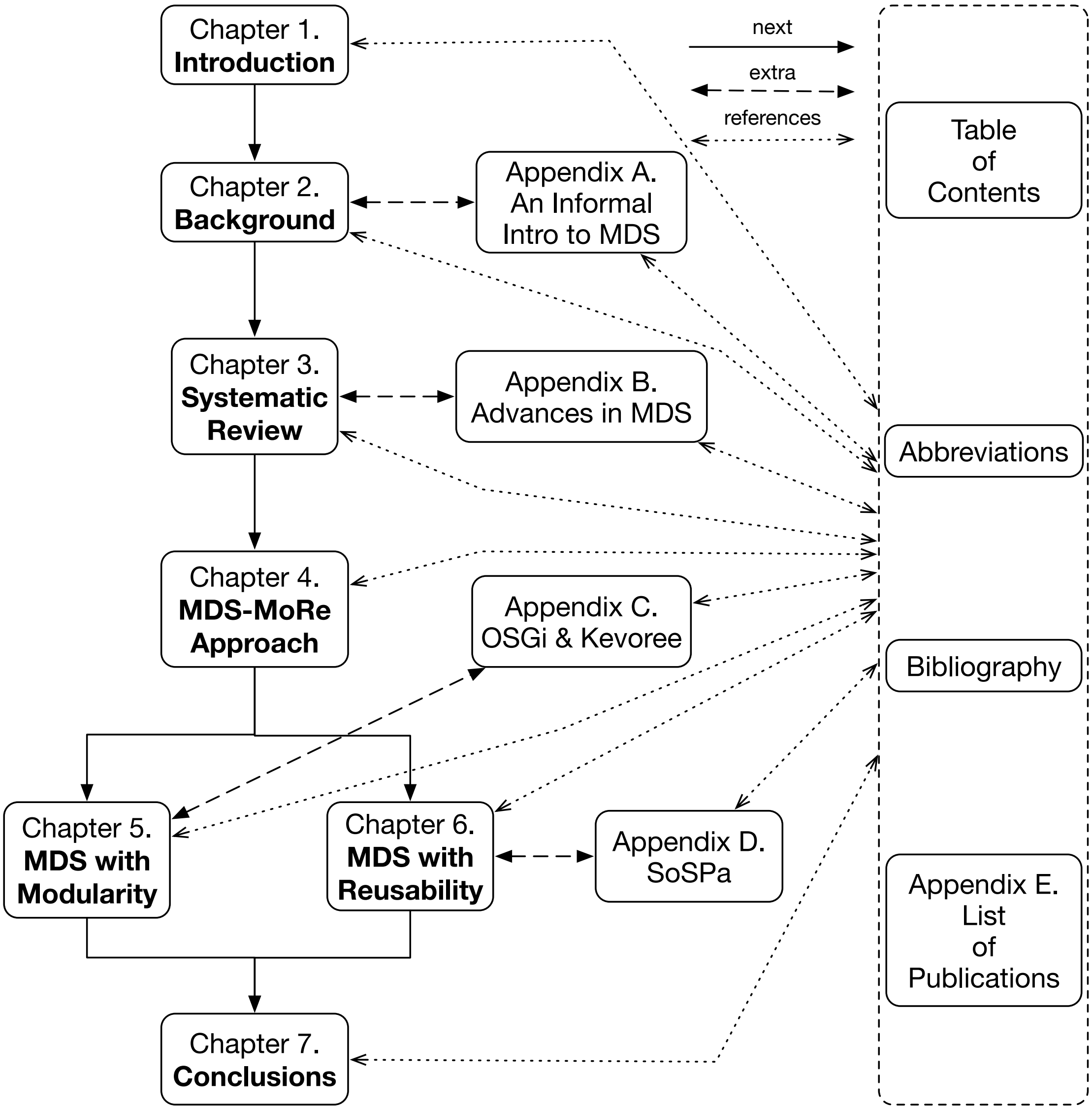

Chapter 1. **Introduction**
next
extra
references
Table of Contents
Chapter 2. **Background**
Appendix A. An Informal Intro to MDS
Chapter 3. **Systematic Review**
Appendix B. Advances in MDS
Abbreviations
Chapter 4. **MDS-MoRe Approach**
Appendix C. OSGi & Kevoree
Bibliography
Chapter 5. **MDS with Modularity**
Chapter 6. **MDS with Reusability**
Appendix D. SoSPa
Appendix E. List of Publications
Chapter 7. **Conclusions**

Figure 1.1: Thesis Structure

Chapter 2

Background

Contents

This chapter provides the background concepts used in this thesis. The research domain of this PhD work is MDS that lies in the intersection of software security engineering and model-driven software engineering. The key concepts of software security engineering are given in Section 2.1. Then, we present in Section 2.2 the main artefacts of model-driven software engineering. Section 2.3 shows a brief history of MDS. Some model-based security testing concepts are given in Section 2.4.

2.1 Software Security Engineering

In this thesis, our MDS approaches should be understood as software security engineering processes. These processes are for specifying, designing, implementing, and testing software for security to identify and solve security problems in the software itself. The most common concepts of software security engineering are given in this section. A short introduction of security patterns is also presented.

2.1.1 Software Security Terms

Several key terms in the domain of software security from [149] are recalled as follows. ***Software security*** *is the ability of software to resist, tolerate, and recover from events* *that intentionally threaten its dependability with the preservation of availability, integrity, and confidentiality of information.*

Secure software *is software that is resistant to intentional attack as well as unintentional failures, defects, and accidents.*

Security engineering *is about building systems to remain dependable in the face of malice, error, or mischance. As a discipline, it focuses on the tools, processes, and methods needed to design, implement, and test complete systems, and to adapt existing systems as their environments evolve [18].*

Asset *is "anything that has value to the organisation" and which therefore requires protection.*

Stakeholder *is an organisation or person who places a particular value on assets.*

Security objective *is a statement of intent to counter threats and satisfy identified security needs.*

Threat *a potential violation of security [40]*

Attack *is an action that could cause a violation of security to occur.*

Attacker *is an entity that carries out attacks.*

Vulnerability *is a weakness of an asset or control, which may be exploited by a threat.*

Countermeasure *is an action taken to protect an asset against threats and attacks.*

Risk *is the combination of the probability of an event and its consequence.*

2.1.2 Security Concerns

Software security is often associated with the three globally accepted security concerns or properties, namely Confidentiality, Integrity, and Availability (CIA). CIA concerns are recalled from [40] as follows.

Confidentiality *is the concealment of information or resources.* Unauthorised parties are prevented from knowing the information or resources, even from being aware of their existence.

Integrity *refers to the trustworthiness of data or resources, and it is usually phrased in terms of preventing improper or unauthorised change.*

Availability *refers to the ability to use the information or resource desired.*

Besides CIA, accountability is another security concern that is discussed in this thesis.

Accountability refers to the ability to keep tracks of who did what and when.

2.1.3 The STRIDE Threat Model

For designing secure software, the STRIDE threat model that consists of six threat categories has been used [140]. These six threat categories could be considered as the refinement of the security concerns mentioned before.

Spoofing identity: Attacker successfully pretends to be an authorised user that he is not.

Tampering with data: Attacker tampers with data by modifying, deleting, adding, or reordering data.

Repudiation: No proof exists to prove that a certain action has been performed.

Information disclosure: An unauthorised user got access to information that should not be exposed to unauthorised user.

Denial of service: Attacker destroys the usefulness of the system for valid users.

Elevation of privilege: Attacker or unprivileged user gets more access to assets in the system than they are authorised for.

2.1.4 Security Solutions

The security solutions for addressing the STRIDE or the security concerns are briefly categorised as follows.

Authentication as recalled from [40] *is the binding of an identity to a principal. The external entity must provide information to enable the system to confirm its identity. This information comes from one (or more) of the following* (called credentials).

1. *What the entity knows (such as passwords or secret information)*
2. *What the entity has (such as a badge or card)*
3. *What the entity is (such as fingerprints or retinal characteristics)*
4. *Where the entity is (such as in front of a particular terminal)*

The authentication process consists of obtaining the authentication information from an entity, analysing the data, and determining if it is associated with that entity. This means that the computer must store some information about the entity. It also suggests *that mechanisms for managing the data are required.* In other words, building an authentication solution must consider where the credentials are stored and updated, how they are protected, and how accounts are created/renewed/cancelled/locked.

Authorisation or Access Control is for controlling access to specific resources. Normally, an entity must be authenticated first to access to a system. Then the authenticated entity's access to specific resources in the system is controlled by using some model of access control based on authorisation rules/permissions. Building an authorisation solution must consider where the permissions are stored and protected, how management and lifecycle of the permissions are implemented.

Cryptography is for confidentiality by encrypting *readable data (plain text) into an unreadable format (cipher text) that can be understood only by the intended recipient after decryption, the inverse function that makes the encrypted information readable again.* [72]. Cryptography is also for integrity and authenticity by preventing alterations (using keyed cryptography hash, e.g. MD5, SHA2), by preventing removal/insertion (using sequence number in message authentication code), and by verifying the origin of a digital document (using digital signature). *There are two types of encryption: symmetric and asymmetric. In symmetric encryption a common key is used for both encryption and decryption. In asymmetric encryption a public/private key pair is used for encryption/decryption: the sender encrypts the information using the receiver's public key, while the receiver uses their private key to decrypt the ciphered text* [72]. Building a cryptography solution must consider the management of certificates for public keys, where the keys are stored, how they are protected, and how they are distributed.

Auditing *is an a posteriori technique for determining security violations: logging (recording of system events and actions) and auditing (analysis of these records). Auditing plays a major role in detection of security violations and in postmortem analysis to determine precisely what happened and how* [40]. Building an auditing solution must make sure all interesting actions are logged with enough detail, and note that audit trails (logs) can be target of attacks. Hence, they need to be maintained in a trusted manner.

Monitoring and intrusion detection is for misuse detection by comparing to a model of attack, for anomaly detection by comparing to a model of normal behaviour. Monitoring can happen at several levels, e.g. network activity (TCP flows), user activity (login/logout, servicer invocations), and service activity (execution flows). Building a monitoring solution must consider where the sensors are located, where the events are stored, how the events are aggregated, and how status and incidents are reported.

2.1.5 Security Patterns

Designing secure software requires experience and expertise. One way to get both could be leveraging security patterns. Software design patterns [84] have been very popular and successful in many areas of software development. Security patterns are patterns that provide domain-independent, time-tested security knowledge and expertise. They are supposed to be reusable bricks upon which sound and secure software can be built. From security engineering's point of view, one of the best practices is the use of security patterns to guide security at each stage of the development process [211]. According to Schumacher et al. [211], a security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution. Patterns are applied in the different architectural levels of the system to realise security mechanisms. Security patterns typically do not exist in isolation because applying one solely can not make a system secure to different threats. So far, catalogs of security patterns are the most accessible, well organised, documented resources of different security solutions for different security concerns, e.g. [72, 220, 211].

A higher organisation of security patterns than catalogs is called a system of security patterns. Related patterns should be highly integrated for working together to address more complex security problems in the real world. A system of security design patterns is a collection of patterns for designing secure systems, together with guidelines for their implementation, combination, and practical use in secure systems development. We develop such a system of security design patterns in Chapter 6.

2.2 Model Driven Engineering

This section first recalls the key ideas behind Model-Driven Software Engineering or *Model-Driven Engineering* (MDE) [144]. MDE has been introduced as a better methodological approach to software engineering than using general purpose programming languages (GPLs). Two of the main advantages of MDE are improving the productivity in engineering software and the quality of software. This is because the engineering methodology of MDE is at a higher level of abstraction than the methodology of using GPLs. MDE follows the irrefutable trend of software engineering in raising the level of abstraction. This can be seen in the introduction of third-generation programming languages (3GLs or GPLs) that are much more machine independent. Before 3GLs, second generation programming languages, a.k.a. assembly languages, are machine-dependent. MDE supports building and using the abstractions of the processes and the concepts in engineering software that could be easier to understand, verify, and simulate than using

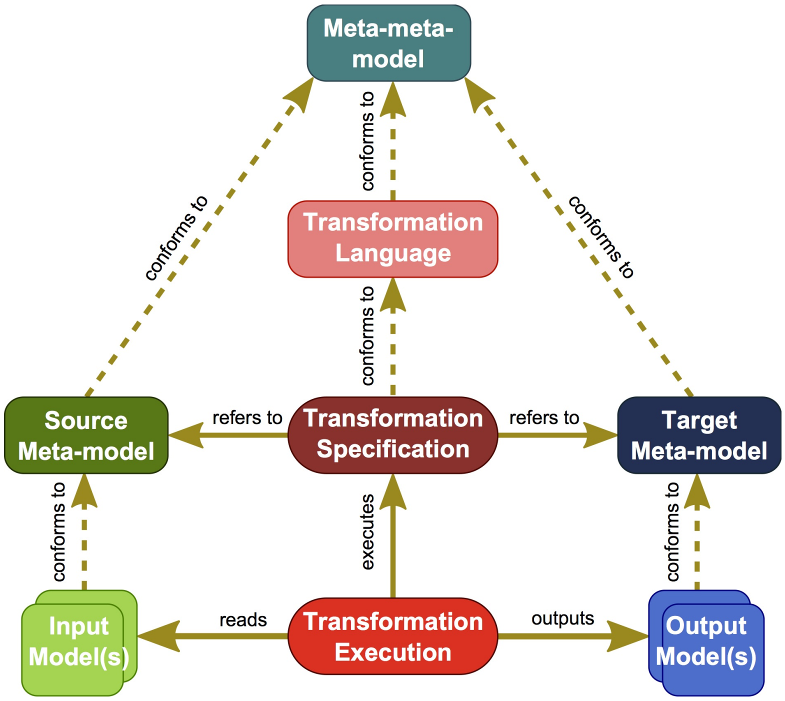


FIGURE 2.1: The Model Transformation Process adapted from [221]

GPLs. The reasoning behind this claim is that those abstractions are close to the domain being addressed by the engineers, whereas GPLs are built essentially to manipulate computer architecture concepts. In other words, MDE allows engineering software in a more intuitive way, close to the problem domain that supposedly could provide better (domain-specific) designs and enable automation to derive computer programs (or the implementation by GPLs) from domain-specific designs.

The remainder of this section presents the key concepts and artefacts of MDE as well as the main approaches following the MDE paradigm.

2.2.1 Models, Metamodels and Model Transformations

Models are the central artefact in MDE. A model in the software engineering world is an abstraction of anything one wishes to capture for engineering purposes. Abstraction means that a model contains only the details that are essential at a given stage of the engineering cycle. Other details are left out at this given stage. The reason is to better focus on engineering the essential concepts at hand at a given stage. In the MDE world, a *model* is defined by using a given modelling language. Such a language (a.k.a. *formalism*) is called *metamodel* that is used to specify all different instantiations (models) of computational artefacts that share the same abstraction concerns.

Besides models, model transformations can be considered as the heart and soul of MDS [212]. Model transformations are developed for manipulating models according to the different purposes in different stages of the engineering cycle. In other words, model transformations drive the engineering process of MDE. For example, model transformation(s) can be used to transform some (model of) user requirements into models for analysis purposes. Model transformation(s) can be used to transform a UML sequence diagram into a formalism for an automated verification by a formal verification tool. Model transformation(s) can also be used to transform models into code. Model transformation languages/engines such as ATL [20], Kermeta [176] or QVT [67] have been developed to provide stable platforms for writing and executing model transformations.

2.2.2 Model-Driven Engineering Approaches

In this section, we briefly summarise the main approaches following the MDE paradigm. First, *Model-Driven Architecture*, which is an early OMG standard generative approach, is presented. Next, we show why the *Domain-Specific Modelling* approach proposes defining a language for each different domain that contributes to an application. Then, *Multi-Paradigm Modelling*, a generalisation of Domain-Specific Modelling Languages, is discussed to show how the different models of computation interact. Last but not least, we present the *Aspect-Oriented Modelling* approach that studies more precisely how different models can be combined or composed.

2.2.2.1 Model-Driven Architecture

In 2001, OMG launched a proposal on Model-Driven Architecture (MDA) to help standardise model definitions, and favour model exchanges and compatibility. We recall in [144] the following points of MDA according to [132]:

- It builds on the UML, an already standardised and well-accepted notation, already widely used in Object-Oriented systems. In an effort to harmonise notations and clean the UML's internal structure, Meta-Object Facility (MOF) was proposed for coping with the plethora of model definitions and languages;
- It proposes a pyramidal construction of models as can be observed in Fig.2.2: artefacts populating the level $M0$ represents the actual system; those in the $M1$ level model the $M0$ ones; artefacts belonging to the $M2$ level are metamodels, allowing the definition of $M1$ models; and finally, the unique artefact at the $M3$ level is MOF itself, considered as meta-circularly defined as a model itself;

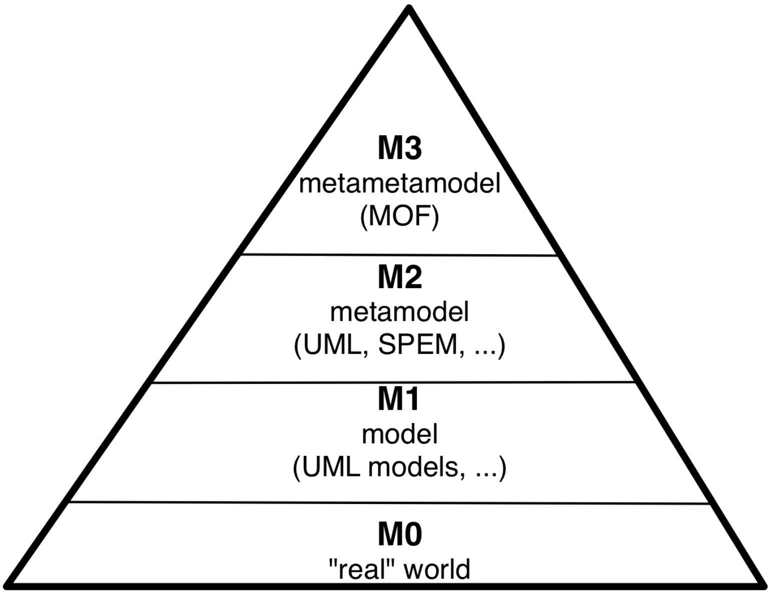


Figure 2.2: The MDA Pyramid [adapted from 223]

- Along with this pyramid, MDA enforces a particular vision of software systems development seen as a process with the following steps (Fig. 2.3): requirements are collected in a Computation Independent Model (Cim), independently of how the system will be ultimately implemented; then a Platform Independent Model (PIM) describes the design and analysis of all system parts, independently of any technical considerations about the final execution platforms and their embedded technologies; a PIM is then refined into a Platform Specific Model (PSM) and combined with a Platform Description Model (PDM) to finally reach code that will run on a specific platform.

MDA promotes a *vertical* separation of concerns [144]. This means that the system is designed at a high level, without any considerations about the target platform specificities. These specificities are then integrated by automated generators to produce code compliant with each platform. This methodology directly inspired several MDS approaches for enforcing security concerns within software applications.

2.2.2.2 Domain Specific Modelling

The "divide-and-conquer" approach is popular to tackle the increasing complexity of current software systems. In this way, the design activity is divided into several areas of concern and focusing on specific aspects of the system. One of the main goals is to make system specifications closer to the domain's experts, which facilitates controlling the quality of the produced artefacts. It is even possible to allow domain experts to create

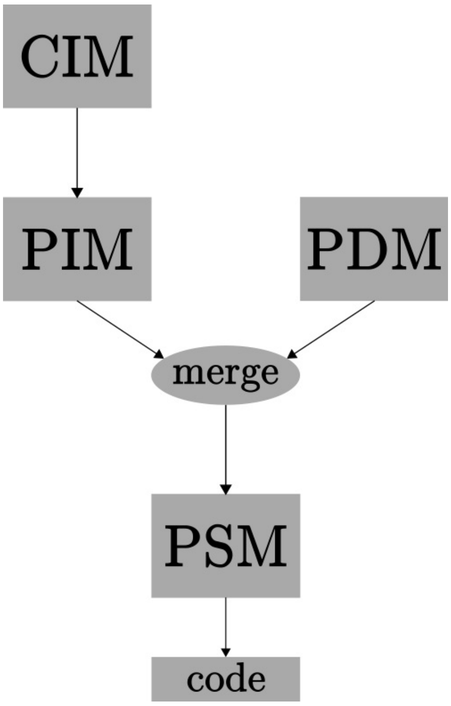


FIGURE 2.3: Model-Driven Architecture Overview from [144]

the computable system specifications in their own domain "language". Therefore, the abstraction level of the produced specifications has been raised with the immediate benefit of increasing the level of confidence attached to them. Within MDE, Domain-Specific Modelling (DSM) is a key methodology for the effective and successful specification of such systems. This methodology makes systematic use of Domain-Specific Modelling Languages (DSMLs, or DSLs for short) to represent the various artefacts of a system as models. DSLs allow designers' efforts to be focused on the variable parts of the design while the underlying machinery takes care of the repetitive, error-prone, and well-known processes of manipulating design models such as code generation. In MDS, DSLs are commonly used for better capturing the specific semantics of security mechanisms.

As part of MDE, DSM also aims at improving the productivity of the development process and the quality of the produced artefacts. A well-known white paper on the subject from [155] presents anecdotal evidence that DSLs can boost productivity up to 10 times, based on experiences with developing operating systems for cell phones for Nokia^{TM} and Lucent^{TM}. These encouraging results pushed the scientific community to invest further in this topic and build environments to facilitate the construction, management, and maintenance of DSLs. This effort has been materialised with concrete frameworks: EMF and GMF [162], AToM^3 [137] or Microsoft's^{TM} DSL Tools [60], among others.

2.2.2.3 Multi-Paradigm Modeling

Multi-Paradigm Modelling (MPM), as introduced by [168], is a perspective on software development in which models should be built at the right level of abstraction and with the right type of models regarding their purpose. Automatic *model transformations* can be used to manipulate models from one representation to another during the model-driven development of a system. In this case it is thus desirable to consider modelling as an activity that spans different models, expressed in different paradigms. The main claimed advantage of such an approach is that the software engineer can benefit from the already existing multitude of languages and associated tools for describing and automating software development activities. The task of transforming data in between formalisms is delegated to specialised machinery such as model transformation engines.

For example in a MPM approach, a UML state chart model representing the abstract behaviour of a software system is used to generate code for execution on a given platform. The same state chart can also be transformed into a formalism that is amenable for verification. Another possible advantage of this perspective on software development is the independence on implementation platform. Models in a MPM approach can be used for different purposes and are platform-independent. Therefore, toolsets for implementing a particular model-driven software development methodology become flexible. Formalisms and transformations may be potentially plugged in and out of a development toolset given their explicit representation.

2.2.2.4 Aspect-Oriented Modelling

Aspect-Oriented Modelling (AOM) is the key in the mix of MDD and Aspect-Oriented Software Development (AOSD). AOSD truly follows the well-known principle of “divide-and-conquer”. More specifically, AOSD aims at addressing crosscutting concerns (such as security, synchronisation, concurrency, persistence, performance, among others) of a system. By separately, specifically dealing with each crosscutting concern, the complexity in developing system could be reduced much more than when addressing all crosscutting concerns together with the system concurrently. Therefore, AOSD provides means for the systematic identification, separation, representation and composition of crosscutting concerns. The modularisation of crosscutting concerns in AOSD had been popularised at code level by the AspectJ programming language [121]. However, together with the emergence of MDE, handling crosscutting concerns earlier in the software life-cycle, for instance at design time [57], or during requirements analysis [100] has become more popular. Crosscutting concerns are encapsulated in separate modules, called *aspects*. Once the different aspects are specified, they can be assembled to build

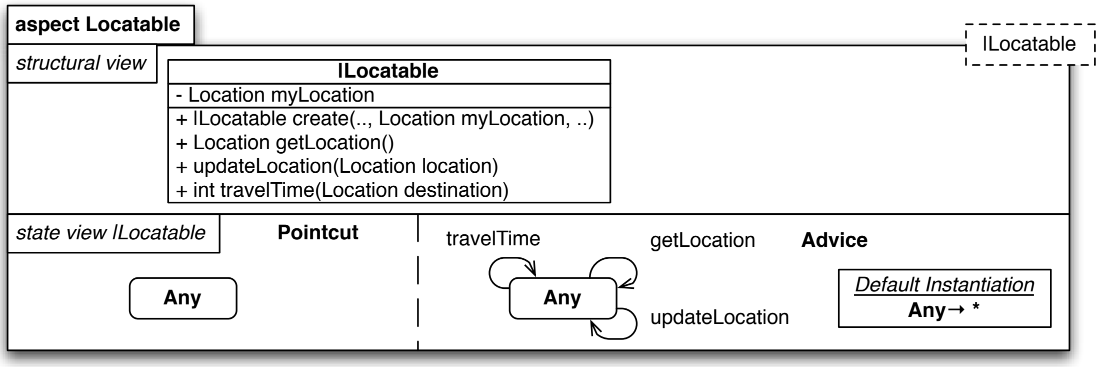


FIGURE 2.4: Aspect *Locatable* [124]

the whole application. This mechanism of integration is called *weaving*. There exist different techniques to represent, compose or weave aspects at model level, e.g. [58, 82, 79, 131, 234, 165, 134].

One of the most complete, reusable AOM methodologies with tools support for multi-view modelling is Reusable Aspect Model (RAM) [123, 122, 7, 124]. RAM [122] is an aspect-oriented multi-view modelling approach with tool support for aspect-oriented design of complex systems. In RAM, any concern or functionality that is *reusable* can be modelled using class, sequence, and state diagrams in an aspect (RAM) model. A RAM model can be (re)used within other models via its clearly defined *usage and customisation interfaces* [13]. The usage interface of a RAM model consists of all the *public* attributes and methods of the class diagrams in the model. The customisation interface of a RAM model consists of all the *parameterised* model elements (marked with a vertical bar |) of the partially defined classes and methods in the model. For example, Fig. 2.4 shows *Locatable* aspect having the parameterised class namely |Locatable. A RAM model can be (re)used by composing the *parameterised* model elements with the model elements of other models. A RAM model can also reuse other RAM models in a hierarchical way. RAM weaver [7] is used to flatten aspect hierarchies to create the composed design model.

In MDS, Aspect-Oriented Modelling (AOM) techniques are naturally suitable to be leveraged. By using AOM, security solution models can be developed relatively separated from the business model (or target model) of a system. Then, model composition techniques can be used to compose the security solution models into the target model of a system.

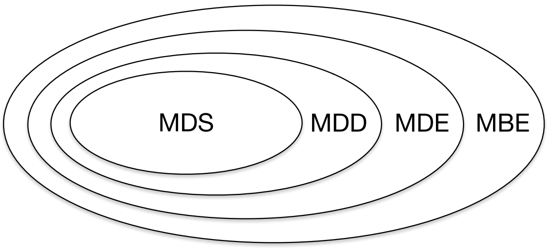


Figure 2.5: Relations among MBE, MDE, MDD and MDS.

2.3 Model-Driven Security

Model-Driven Security (MDS) can be seen as a specialisation of MDE for supporting the development of security-critical applications. MDS makes use of the conceptual approach of MDE as well as its associated techniques and tools to propose methods for the engineering of secure applications. More specifically, (security-oriented) *models* are the central artefacts in every MDS approach. Besides being used to describe the system's business logic, they are used extensively to capture security concerns. Models allow the introduction and enforcement of security in the application being built. In this section, we start by mentioning several early MDS approaches in which (security-oriented) *models* had become a central artefact like in an MDE context. We then present a view on the position of MDS research regarding MDE and other related Model-Driven methodologies.

2.3.1 A Brief History of MDS

In [144], we briefly review several contributions that provided early first ideas and definitions for MDS in the context of MDE. It is important to note that, from the early beginning of MDS, the approaches used models to focus on capturing the domain and the security properties, abstracting away from implementation details. The use of models in advanced (formal) verification and validation techniques was already promoted. Two well known UML-based approaches that have been discussed are UMLsec [111, 109, 110], SecureUML [143, 30, 26]. Two other approaches that promoted the use of DSLs are also discussed: [145] and [136]. More details on this discussion can be found in [144].

2.3.2 MDS regarding MBE, MDE, MDD

Numerous security engineering techniques exist which support the development of secure systems. There are also many MDE techniques for the development and maintenance of software systems in general. Our research focus, however, is only on MDE approaches that are specifically customised for supporting the development of secure systems. As we already mentioned, MDS can be considered a subset of MDE. We will now clarify the relations between MDE, Model-Based Engineering (MBE), Model-Driven Development (MDD), security engineering, and MDS. Regarding MBE, MDE, and MDD, we agree with the point of view presented by Brambilla et al. [48, p. 9]. Specifically, MBE can be used for development processes in which models may not necessarily be the central artefacts for development. E.g., if models are only used for documentation purposes and not in automated transformations. MDE can be seen as a subset of MBE in which models have to be the key artefacts throughout the development, i.e. models "drive" the process in every step: All development, evolution, and migration tasks have to be influenced by explicit models. MDD can be considered a subset of MDE that only denotes development activities with models as the primary artefact. Normally, model-to-model transformations (MMTs) or model-to-text transformations (MTTs) are used in MDD to obtain other models or to generate code. Thus, MDS refers to all research approaches that focus on a MDD process for building secure systems. Figure 2.5 depicts these subset relations.

2.4 Model-Based (Security) Testing & Mutation Analysis

Because we leverage model-based testing and mutation analysis in our MDS approach, these two methodologies are shortly introduced in this section.

2.4.1 Model-Based (Security) Testing

Model-based testing (MBT) is a variant of testing that relies on the behavior models of a system under test (SUT) and/or its environment to (automatically) derive test cases for the system [226]. Therefore, MBT allows the tests derivation process to be structured, reproducible, programmable, and documented.

A MBT approach in which the security requirements of a SUT are the main focus for testing is called model-based security testing (MBST) [71]. MBST is an important part of MDS because the verification and validation of any (newly built) secure systems are essential. In MDS, MBST can be integrated logically because of its "model-based" nature. Only that we need to introduce specific security testing requirements into MBT to identify vulnerabilities and ensure security functionality.

2.4.2 Mutation Analysis

Mutation analysis [64, 93] is a well established technique supporting various software development activities like testing [104] and debugging [196]. It operates by defining mutation operators to introduce artificial defects called mutants into the artefacts of the program under investigation [193]. Its requirement is to design test cases capable of revealing these defects. The utilised test cases are examined to reveal the introduced mutants. A mutant is "killed" if it is detected by a test case. The quality of testing process can be measured by the ratio of the killed mutants per the totally introduced mutants (called mutation score). Mutation score can help to work on improving the test cases. Researchers have shown that mutants despite being artificially introduced behave quite similarly to real faults [19]. Thus, the benefits of their use are evident. Section 5.7 presents how we leveraged mutation analysis for security testing.

Chapter 3

An Extensive Systematic Review of Model-Driven Security

Contents

Over a decade of research on MDS has resulted in a large number of MDS studies and publications. To provide a detailed analysis of the state of the art in MDS, a systematic literature review (SLR) is essential. This chapter presents how we conducted an extensive SLR on MDS, and the results.

3.1 Introduction

For more than a decade since MDS first appeared, a considerable number of MDS publications has shown a great attention of the research community to this area. The MDS approaches vary greatly in many artefacts such as the security concerns addressed, the modelling techniques used, the model transformations techniques used, the targeted application domains, or the evaluation methods used. To provide a detailed state of the art in MDS, a full systematic literature review (SLR) is needed.

So far, a full SLR on MDS does not exist. Surveys on MDS approaches ([118, 28, 144, 227]) could provide in-depth analyses of some well-known MDS approaches, but do not summarise the complete research area systematically. [103] could be closer to our work, but has several limitations in terms of scope and methodology. E.g., it missed many important primary MDS approaches such as UMLsec [107], and aspect-oriented approaches. In contrast, our SLR is performed in both width and depth of MDS research that reveals an extensive set of primary MDS studies. Furthermore, our review provides a detailed overview on the key artefacts of every MDS approach such as used modelling techniques, considered security concerns, employment of model transformations, verification or validation methods, and targeted application domains. Finally, we present trend analyses for MDS publications, and for the addressed security concerns and other key artefacts.

The main contributions of this chapter are: 1) detailed and condensed results on key MDS artefacts of all identified primary MDS publications; 2) a diagnosis of limitations and open issues of current MDS approaches with suggestions for potential MDS research directions; 3) a classification of principal and emerging/less common MDS approaches; and 4) trend analyses.

The remainder of this chapter is structured as follows. Section 3.2 provides main background concepts and definitions. The objective of this SLR, its research questions, search strategy, and selection process are described in Section 3.3. In Section 3.4, we present our evaluation criteria and data extraction strategy. Section 3.5 shows the main results of our review. Threats to validity are discussed in Section 3.6. In Section 3.7, related work is presented. Section 3.8 concludes the chapter by summarising the results, highlighting open issues, and giving some thoughts on future work.

3.2 Systematic Literature Review and Snowballing

SLR is a means for thoroughly answering a particular research question, or examining a particular research topic area, or phenomenon of interest, by systematically identifying, evaluating, and interpreting all available relevant research [129]. Well-known guidelines for conducting SLRs in software engineering were provided by Kitchenham [129] and Biolchini et al. [39]. All individual studies that are identified as relevant research contributing to a SLR are called *primary* studies [129]. In this chapter, based on the numbers of publications and citations of *primary* MDS studies, we further classify them into *principal* MDS studies, and *less common* or *emerging* MDS studies.

In a SLR, it is crucial to transparently and correctly identify as many relevant research papers in the focus of the review as possible. The search strategy is key to the identification of primary studies and ultimately to the actual outcome of the review [235]. The guidelines by Kitchenham [129] for SLRs in software engineering suggest to start with a database search that is based on a search string and also called *automatic search* in this chapter. They also recommend complementary searches, e.g. a *manual search* on journals and conferences proceedings, references lists, and publications lists of researchers in the field.

Both automatic search and manual search have limitations [235]: The former depends on the selection of databases, on database interfaces and their limitations, on the construction of search strings, and on the identification of synonyms. The latter depends on the selection of research outlets, e.g. journals or conferences, and cannot be exhaustive. Therefore Wohlin et al. [235] proposed the snowballing search strategy as a first step to systematic literature studies. The key actions of the snowballing search strategy are: 1) identify a starting set of primary papers; 2) identify further primary papers using the reference lists of each primary paper (backward snowballing); 3) identify further primary papers that cite the primary papers (forward snowballing); 4) repeat steps 2 and 3 until no new primary papers are found. We are convinced, that the snowballing search strategy complements the automatic and manual search strategies of Kitchenham [129]. In our SLR we defined and performed a snowballing search strategy that builds on the set of primary papers found in automatic and manual searches. Details of our search strategy are presented in Section 3.3.

3.3 Our systematic review method

Our SLR method is based on the guidelines of Kitchenham [129], and the snowballing strategy of Wohlin et al. [235]. We presented the motivation for our review in Section 3.1 and state our research questions in the next section. Based on these research questions, we developed a review protocol, which was evaluated before conducting the review. Figure 3.1 shows an overview of our SLR process. We combined an automated database search (Section 3.3.2.2), a manual search in relevant journals and conference proceedings (Section 3.3.2.3), and a snowballing strategy (Section 3.3.2.4) to identify as many primary MDS papers as possible. For our predefined protocol we clarify the selection criteria (Section 3.3.3) to reduce a possible bias in the selection process (Section 3.3.4). The quality assessment, data extraction and synthesis of the primary MDS studies are based on a fixed set of evaluation criteria (Section 3.4). The results obtained from classifying, synthesising, analysing, and comparing the data extracted from the primary MDS studies are presented in Section 3.5.

3.3.1 Research Questions

This SLR aims to answer the following research questions:

RQ1: How do existing* MDS *approaches support the development of secure systems?

This question is further divided into the following subquestions:
RQ1.1: What kinds of *security concerns* are addressed and what *security mechanisms* are used by these MDS approaches?
RQ1.2 : How do the MDS approaches *specify* or *model* security requirements together with functional requirements? Is there any tool that supports the *modelling* process?
RQ1.3 : How are *model-to-model transformations* (MMTs) used and which MMT engines are used? Is there any tool support for the transformation process?
RQ1.4 : How are *model-to-text transformations* (MTTs) used to generate code, including security infrastructure and configuration? Which tools are used for the generation process?
RQ1.5 : Which *methods* were used to evaluate the approaches? What results have been obtained?
RQ1.6 : Which *application domains* are addressed by the MDS approaches?

RQ2 : What are current limitations of existing* MDS *research?

RQ3 : What are open issues to be further investigated?

3.3.2 Search Strategy

We developed a hybrid strategy to exhaustively search for MDS papers. The goal was not to miss any relevant MDS paper and therefore to find as many primary MDS papers as possible. Our hybrid strategy consists of three parts: automatic search (Section 3.3.2.2), manual search (Section 3.3.2.3), and snowballing (Section 3.3.2.4). In each step, we applied inclusion and exclusion criteria (Section 3.3.3) to select primary MDS studies.

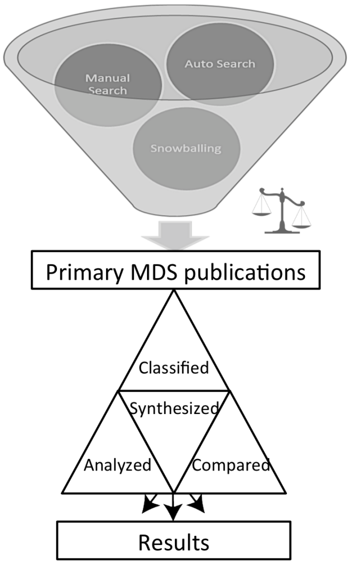


FIGURE 3.1: An overview of our SLR process.

3.3.2.1 Identification of a Search String

Based on the research questions (Sect. 3.3.1), we created search terms to form search strings, e.g. *model-driven*, *model-based*, *security*. We divided our search terms into three categories: MDE (model-driven, model-based, model*, MDA, UML), modelling (specify*, design*), transformations (transform*, code generation) and security.

To form the search string, we used a conjunction that combines disjunctions of the keywords of each term group. We had to refine our search string several times to make sure that as many potential relevant papers as possible are reached and had to adapt it according to the required format of the search engines.

3.3.2.2 Step 1: Automatic Search in Databases for Scientific Literature

Using the search string described earlier, we performed automatic search within five electronic databases for publications between 2000 and 2014: IEEE Xplore[1], ACM Digital Library[1], Web of Knowledge (ISI)[1], ScienceDirect (Elsevier)[2], and SpringerLink (MetaPress)[2].

3.3.2.3 Step 2: Manual Search in Conferences Proceedings and Journals

To ensure the correctness and completeness of our review, we also conducted two manual searches: a manual search in potentially relevant peer-reviewed journals, and another

[1] ieeexplore.ieee.org, dl.acm.org, apps.webofknowledge.com

[2] sciencedirect.com, link.springer.com

one in potentially related conference proceedings. We selected journals and conferences that are highly ranked either in the domain of software engineering (SE) or security and privacy (S&P). We manually searched for all published papers from 2001 to 2014 in 10 journals and 10 conference proceedings as shown in Table 3.1 and 3.2.

Table 3.1: Journals used in our manual search.

Acronym	**Full Name**	**Field**	**Rating**
TSE	IEEE Transactions on Software Engineering	SE	56
JSS	Journal of Systems and Software	SE	34
IEEE S&P	IEEE Security & Privacy	S&P	31
TISSEC	ACM Transactions on Information and System Security	S&P	29
TDSC	IEEE Transactions on Dependable and Secure Computing	S&P	28
COMPSEC	Computers & Security	S&P	27
INFSOF	Information & Software Technology	SE	27
SOSYM	Software and System Modeling	SE	27
TOSEM	ACM Transactions on Software Engineering and Methodology	SE	25
ESE	Empirical Software Engineering	SE	20

Table 3.2: Conference proceedings used in our manual search.

Acronym	**Full Name**	**Field**	**Rating**
ICSE	International Conference on Software Engineering	SE	60
CCS	ACM Conference on Computer and Communications Security	S&P	54
S&P	IEEE Symposium on Security and Privacy	S&P	49
USENIX	USENIX Security Symposium	S&P	39
AOSD	Modularity/Aspect-Oriented Software Development	SE	37
NDSS	Network and Distributed System Security Symposium	S&P	35
ACSAC	Annual Computer Security Applications Conference	S&P	29
SACMAT	Symposium on Access Control Models and Technologies	S&P	28
ESORICS	European Symposium on Research in Computer Security	S&P	24
MODELS	Model Driven Engineering Languages and Systems	SE	21

The 10 journals are chosen based on the relevance, the high impact index (Journal Citation Reports 2011), and the field ranking in the last 10 years according to the Microsoft

Research website. 6 journals from SE and 4 journals from S & P were selected. We added the Empirical Software Engineering journal in order to find empirical validations of MDS approaches. The 10 conferences are also chosen on the relevance, and the conferences field ranking in the last 10 years according to the Microsoft Research website.

3.3.2.4 Step 3: Snowballing for a complete set of primary MDS papers

The automatic search and manual search processes yielded a set 95 primary MDS papers. To make sure that our final set of MDS papers is complete we adopted the snowballing strategy presented by Wohlin et al. [235]. We use the big set of primary MDS papers provided by automatic and manual searches as input for our snowballing strategy as follows.

Figure 3.2 shows how we formed the input set of MDS papers for snowballing. After conducting the automated search and applying the primary study selection procedures, we obtained a first set of 80 MDS papers (Step 1). Similarly, after conducting the manual search and applying the primary study selection procedures, we obtained a second set of 29 MDS papers (Step 2). We merged these two sets in order to form a set of selected MDS papers that was used for partially conducting our snowballing strategy. Jalali et al. [102] provided a comparison between the SLR method and the snowballing method. They state that the snowballing method can be used to complement the automated search and manual search in terms of closing the final set of primary MDS papers. Because we already performed the automatic and manual searches for obtaining a set of 95 primary MDS papers, we only adopted the following 3 out of 5 steps of the snowballing strategy:

1. *Backward snowballing*: identify further potential primary MDS papers in the reference lists of the current primary MDS papers. Initially this is the set of papers found by the automated search and manual search.

2. *Forward snowballing*: identify further potential primary MDS papers by searching for papers that cite a current primary MDS papers. We used Google Scholaras recommended [235], because it captures more than individual databases.

3. If no new papers are found by repeating steps 1 & 2, then identify further primary MDS papers by searching publications lists on personal homepages or author pages of database and institutions for the primary authors of the identified primary MDS approaches. This step was performed to ensure that the most recent publications on the same or similar topics are included. If additional papers are identified then go back to Step 1.

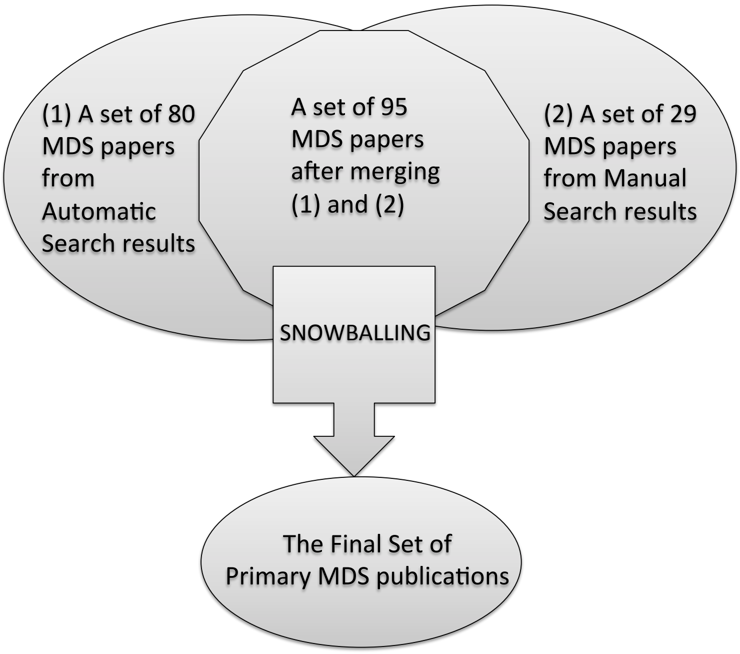

(1) A set of 80 MDS papers from Automatic Search results
A set of 95 MDS papers after merging (1) and (2)
(2) A set of 29 MDS papers from Manual Search results
SNOWBALLING
The Final Set of Primary MDS publications

Figure 3.2: Snowballing after Automatic Search & Manual Search.

Once no additional papers were found in step 3, we closed the cycle of identified primary MDS papers for data extraction, synthesis, and evaluation.

3.3.3 Inclusion and Exclusion Criteria

We already discuss our definition of MDS to give a better idea how we consider a paper as an MDS paper in Section 3.2. Here, we show in detail the inclusion and exclusion criteria that have been used in our primary MDS studies selection process.

MDS approaches for developing secure system vary a great deal as different security concerns can be addressed and different model-driven techniques can be used. Therefore, it was absolutely necessary to define thorough inclusion and exclusion criteria to select the primary studies for answering our research questions:
1. Papers not written in English were excluded and already filtered out in our search process.
2. Papers with less than 5 pages in IEEE *double-column format or less than 7 pages in* LNCS *single-column format were excluded.*
3. Papers not concerned with MDE *were excluded. For example, papers addressing security problems without using* MDE *techniques were excluded.*
4. Papers proposing model-driven approaches without a focus on security concerns were excluded. E.g., model-driven approaches for performance analysis were excluded.
5. When a single approach is presented in more than one paper describing different

TABLE 3.3: Summary of the selection process based on Automatic Search

Source	IEEE	ACM	ISI	SD	SL	**Total**
Search results	2997	1506	3299	828	2003	**10633**
After reviewing titles/keywords	109	90	91	24	81	**395**
After reading abstracts	78	44	35	19	61	**237**
After skimming/scanning	31	21	17	15	20	**104**
After final discussion						**93**
Finally selected						**80**

parts of the approach, we included all these papers, but still considered them as a single approach.

6. When more than one paper described the same or similar approaches, we only included the one with the most complete description of the approach. E.g., an extended paper [184] published in a journal will be selected instead of its shorter version [190] published in a conference proceeding.

7. Papers with insufficient technical information regarding their approaches were excluded. E.g., papers that neither provide a detailed description of secure models, nor a precise security notion, nor transformation techniques, were considered incomplete and were excluded.

8. Only papers with a MDD *perspective, i.e.* MDE *papers in which models are central artefacts throughout the development phase, were selected. Papers using model-based techniques only for verifying or analysing security mechanisms without a link to the implementation code were excluded.*

9. Papers with less than 2 citations per year minus 2 as reported by Google Scholar were excluded.

With these 9 clearly defined inclusion and exclusion criteria, we were able to perform the selection process in a more transparent and less biased way.

3.3.4 Primary Studies Selection and Its Results

Here we present the selection process conducted while performing each search step in the three-pronged search process and its results. Figure 3.3 shows details of our whole selection process with all the numbers of MDS papers selected in each step.

3.3.4.1 Selection Process in the Automatic Search Step

Table 3.3 shows the results of our automatic search that is explained as follows. The papers found from the repositories described in Section 3.3.2.2 were divided among

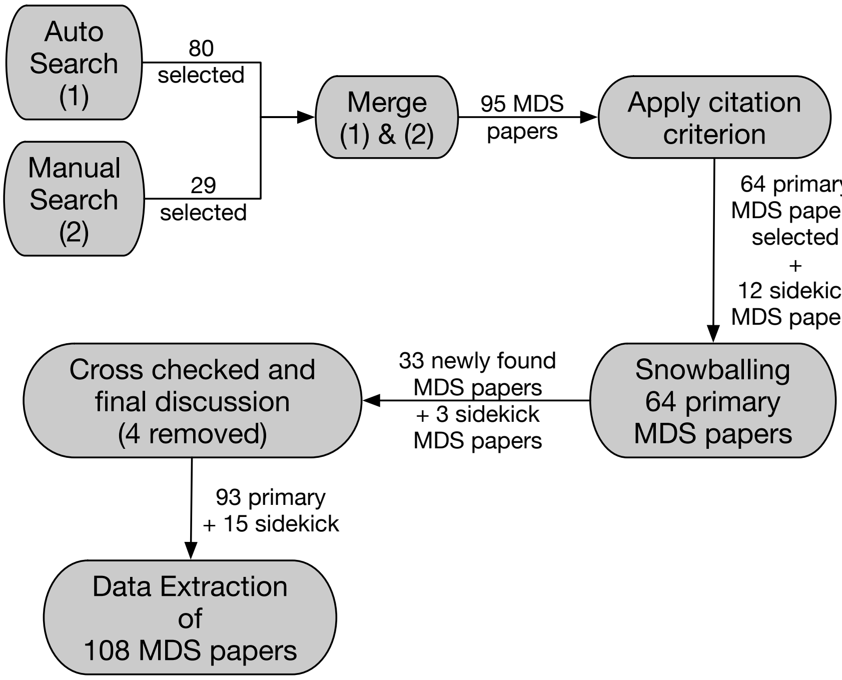

Auto Search (1)
80 selected
Manual Search (2)
29 selected
Merge (1) & (2)
95 MDS papers
Apply citation criterion
64 primary MDS papers selected + 12 sidekick MDS papers
Snowballing 64 primary MDS papers
33 newly found MDS papers + 3 sidekick MDS papers
Cross checked and final discussion (4 removed)
93 primary + 15 sidekick
Data Extraction of 108 MDS papers

FIGURE 3.3: The Selection Process with all the steps

reviewers. For each paper, we first read the paper's title, keywords, and the venue where the paper was published to see whether it is relevant to our research topic. If the title and keywords of a paper were insufficient for deciding whether to include or exclude it, we further checked the paper's abstract. If the abstract of the paper were insufficient for deciding whether to include or exclude it, we further skimmed (and scanned if necessary) the paper's full text. Once each reviewer had done selecting candidate papers from his repositories, all the candidate papers from different repositories were merged to remove duplicates. We kept track of this merging process to see which duplicates were found. Duplicated papers were directly included in the final set of selected papers. All other candidate papers, were discussed by at least two reviewers. Some border-line papers were checked by all reviewers. We maintained a list of rejected candidate papers, with reasons for the rejection, after discussion among reviewers. In the end, 80 MDS papers were selected.

3.3.4.2 Selection Process in the Manual Search Step

29 candidate MDS papers were found in the manual search step. By merging with the set of 80 papers above, we obtained in total 95 MDS papers.

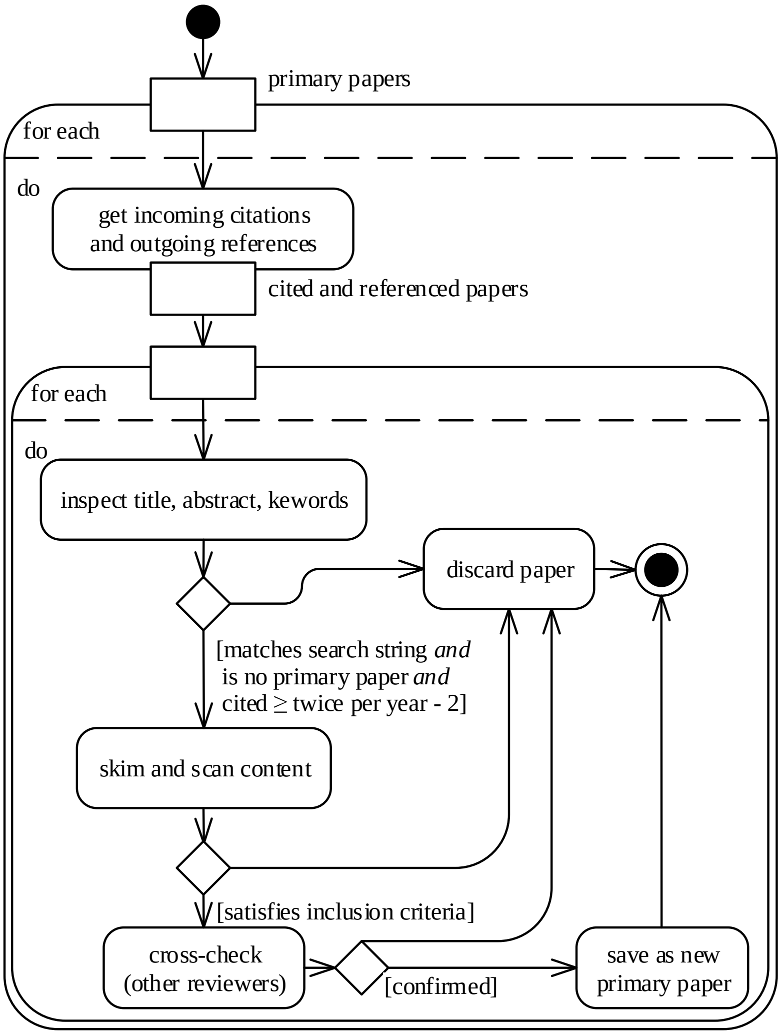

primary papers
for each
do
get incoming citations and outgoing references
cited and referenced papers
for each
do
inspect title, abstract, kewords
discard paper
[matches search string *and* is no primary paper *and* cited ≥ twice per year - 2]
skim and scan content
[satisfies inclusion criteria]
cross-check (other reviewers)
[confirmed]
save as new primary paper

FIGURE 3.4: Our selection process while snowballing

3.3.4.3 Selection Process in the Snowballing Step

After the first two steps, we conducted the snowballing as described in Section 3.3.2.4. However, once obtaining all the numbers of citations of every paper in the set of 95 MDS papers above, we found out that some papers are much less cited than others, or even having no citation at all. We argue that the papers without a minimum number of citations after getting published for a specific period could be considered as not significant in terms of research impact and continuation. On the other hand, we also were not too strict on this aspect. Specifically, we decided that papers with the number of Google Scholar citations[3] less than 2 citations per year minus 2 are excluded. Thus, the selection criterion 9 about number of Google Scholar citations was added. This means we leave out the papers that are not active, and do not have a minimum impact

[3] The citations of these 95 MDS papers were dated on May 19, 2014

after being published for more than 2 years. Of course, this also means the recent MDS papers published in 2013 and 2014 are not excluded by this citation criterion.

In 95 MDS papers, 31 papers were removed according to this citation criterion. Consequently, we used 64 primary MDS papers as the input for our snowballing process. In the snowballing step, we also apply the citation criterion[4] together with other criteria to select primary MDS papers. Details of our selection process while snowballing are shown in Figure 3.4. It is also important to note that every MDS candidate paper is cross-checked by three reviewers before any inclusion or exclusion decision. After all three steps, we have ended up with 93 primary MDS papers. However, we realised that some MDS papers, which were removed because of the citation criterion, should be put back in the final set as "sidekick" MDS papers. The main reason is that those MDS papers contain extra details of the approaches presented in the selected primary MDS papers. A "sidekick" MDS paper is a true MDS paper that was only excluded because of the citation criterion. Every "sidekick" MDS paper is part of a primary MDS approach. If they were removed, some important properties of the relevant primary MDS approaches could be missing in the data analysis. E.g., a paper presents an empirical study of a primary MDS approach. We would miss that empirical study of the primary MDS approach if the "sidekick" paper was removed because of the citation criterion. Thus, 15 "sidekick" MDS papers were put back in the final set. In the end, the final set of 108 MDS papers is used for data extraction and evaluation.

3.4 Evaluation criteria & Data extraction strategy

Classifications and taxonomies are important in any research domain, e.g. [62], [151]. In this section, we describe a set of key artefacts of MDS that forms a so-called evaluation taxonomy of MDS. We derived our evaluation taxonomy from our research questions. Moreover, our evaluation taxonomy are also based on the synthesis of evaluation criteria described in [120] and [118]. Having an evaluation taxonomy makes it more systematic to assess key artefacts of MDS as well as classify and compare different MDS approaches.

Our taxonomy of MDS classifies different dimensions that one has to take into account while leveraging MDE techniques for developing secure systems. The elements of our taxonomy are described as follows. For each element, the data extraction strategy is described to show how we extracted data from the primary studies to answer our research questions.

[4] The citations of MDS papers found in snowballing were dated on-the-fly.

Security concerns: In this dimension, we classify primary studies according to the security concerns/mechanisms that the MDS approaches are dealing with. The range of security concerns is broad, e.g. authorisation, authenticity, availability, confidentiality, integrity, etc. We will count the number of papers addressing each security concern. Thus, security topic areas that addressed by the MDS approaches are measured quantitatively.

Modelling approaches: Security concerns can be modelled separately or not from the business logic, and by using different modelling techniques/languages. Primary studies can be classified by the paradigms of modelling, i.e. *Aspect-Oriented Modelling* (AOM) or non-AOM. In AOM approaches, security concerns are modelled in separate *aspect models* to be eventually woven (integrated) into the *primary model(s)*. Using AOM, security concerns can be modelled separately, modularly in design units (aspects) [214]. Vice versa, in non-AOM approaches, security concerns are not modelled as AOM aspects. That means security concerns can be modelled together with business logic in every place where they are needed. But, we also classify as non-AOM approaches where security concerns modelled separately (*separation of concerns*) from the business logic that can be integrated later into the system. E.g., a non-AOM approach could (separately) specify an access control policy using a *Domain-Specific Language* (DSL)[5], and then transform and/or generate XACML[6] standard file for enforcing the access control policy. In other words, we would like to know the percentage of non-AOM approaches compared to the percentage of "full" AOM/*Aspect-Oriented Software Development* (AOSD) approaches. Separation of concerns can be considered as a key principle to cope with modern complex systems. Furthermore, approaches are also classified by the modelling languages, e.g. UML diagrams, UML profiles, or some kinds of DSLs, used to model security concerns and business logic. The outcome models are classified as of type standard or non-standard, and structural, behavioural, functional or other types. The granularity levels of outcome models are also reviewed.

Model-to-model transformations (MMTs) & tools: MMTs can take part in the key steps of the development process, e.g. for composing security models into business models and/or transforming *platform-independent models* (PIMs) to *platform-specific models* (PSMs). We extract data related to MMTs for answering the following questions: How well-defined are the MMTs rules? How MMTs are implemented? Using which MMT engines (e.g. ATL[7], QVT[8], Kermeta[9], Graph-based MMTs, etc.)? Is there any tool support for the transformation process? What is the automation level

[5] http://martinfowler.com/books/dsl.html

[6] *extensible Access Control Markup Language*, a XML-based declarative access control policy language

[7] http://www.eclipse.org/atl/

[8] http://projects.eclipse.org/projects/modeling.mmt

[9] www.kermeta.org

of MMTs: *automatic* (if entire process of creating the target model can be done automatically), *semi-automatic*, and *manual*. Some information about the classification of MMTs should also be extracted to see if it supports well for the security mechanisms? E.g., *endogenous* MMTs or *exogenous* MMTs used?

Model-to-text transformations (MTTs, code and/or security infrastructure generation) & tools: MDE also supports the development of secure systems by automatically generating code, including (partial) complete, configured security infrastructures. Data should be extracted to see the main purposes of using code generation techniques. Is the whole system including security infrastructure generated? Or just the security infrastructure (configuration) is generated? Can fully code and/or security infrastructure be generated? Or just the (code) skeleton of the system is generated? Which tools are used for the code generation process?

Application domains: MDS approaches are also classified on the target application domains of the secure systems. Some MDS approaches might target only a specific application domain. Some might explicitly be applicable to different application domains in general. Others might implicitly be applicable to different application domains. Some examples of application domains are information systems, web applications, databases, secure smart-card systems, embedded systems, distributed systems, etc. The application domains might be overlapping but could show relatively the intended application domain(s) of a specific MDS approach.

Evaluation methods: To point out the limitations of each approach, we check again how the approach has been evaluated. How many case studies have been performed? What results have been obtained? What other evaluation methods (other than case studies) have been applied to evaluate these approaches? This can be answered by extracting data from the validation section of each paper.

To make the data extraction consistent among the reviewers, we all tried to extract the relevant data from a small set of prospective primary papers. We then discussed to ensure a common understanding of all the extracted data items and refined the data extraction procedure. Excel files were used for storing the extracted data while a tool called Mendeley[10] was used in reviewing and controlling the selected papers. The final set of primary studies (selected papers) was divided among reviewers. Each reviewer examined again the allocated papers and enriched the Excel files to ensure detailed data according to the taxonomy has been extracted from the selected papers. The data extraction forms of each reviewer were read and discussed by two other reviewers. All ambiguities were clarified by discussion among the reviewers.

[10] http://www.mendeley.com/

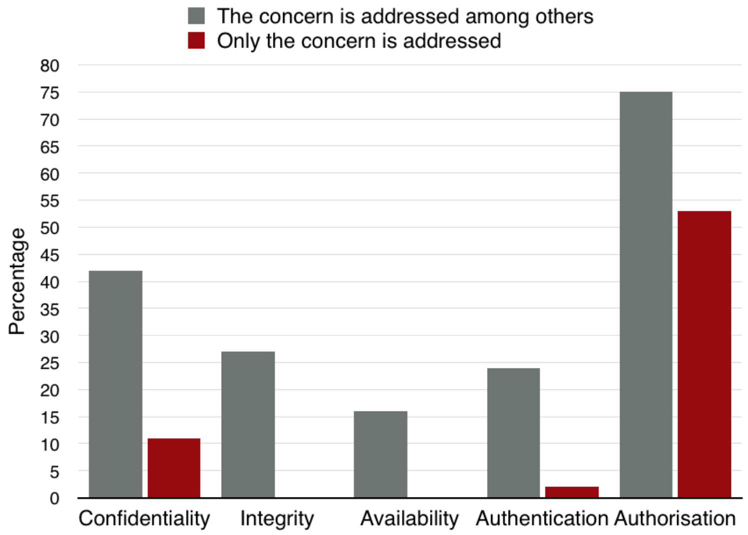


FIGURE 3.5: How much each concern is addressed in MDS?

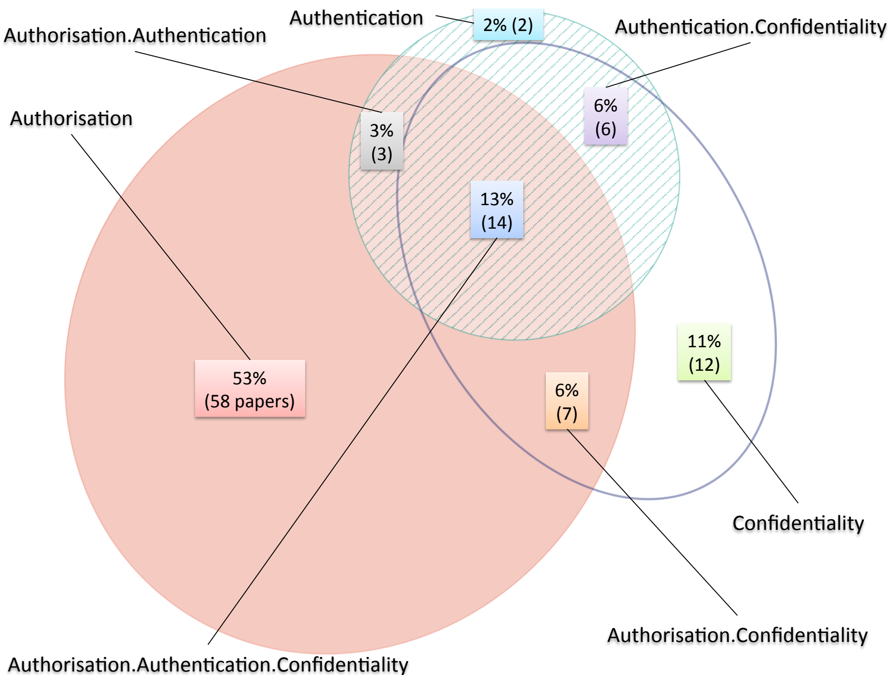


FIGURE 3.6: Intersection of Authentication, Authorisation, and Confidentiality

To answer the last two research questions, we reviewed the range of security topics, the scope of MDS research work and the quality of MDS research results to determine whether there are any observable limitations and open issues.

3.5 Results

First, in Section 3.5.1 we report on some statistic results according to the evaluation criteria. Then, the principal MDS approaches and other emerging/less common MDS

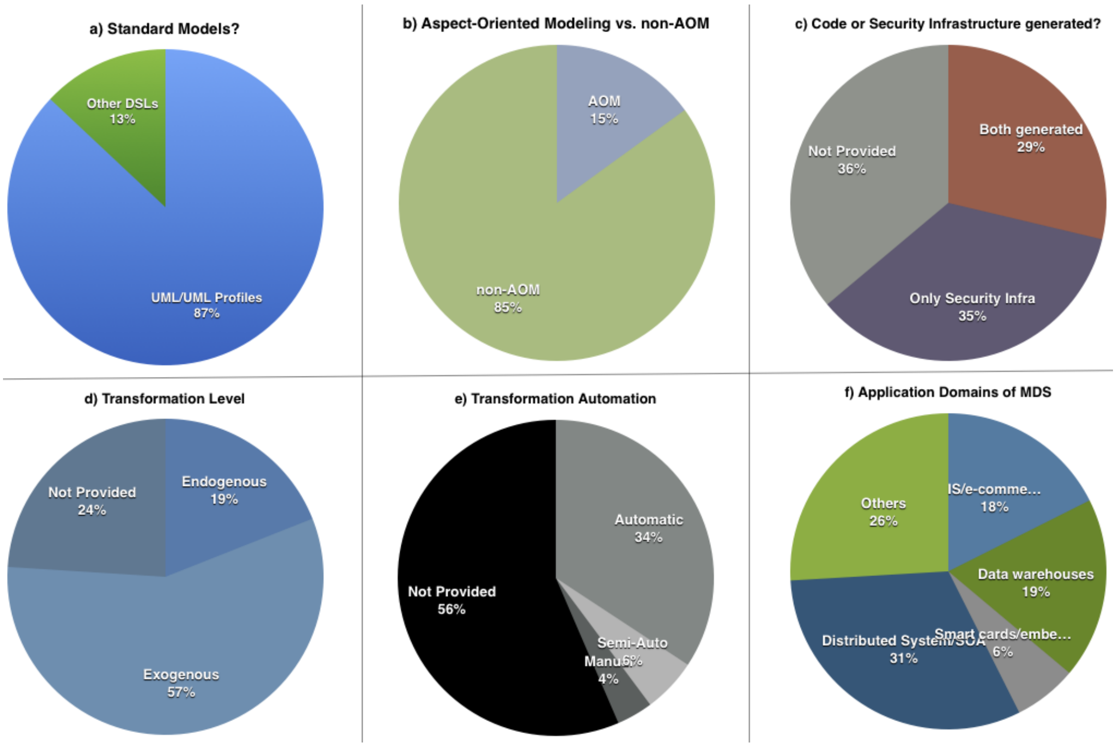


FIGURE 3.7: Statistics of some key MDS artefacts

approaches are revealed and described in Sections 3.5.2, 3.5.3 respectively. Finally, Section 3.5.4 analyses the trends of some key factors in MDS.

3.5.1 Results per Evaluation Criterion

An overview of the results can be seen in Figures 3.5, 3.6 3.7 and Tables 3.4, 3.5. Figs. 3.5, 3.6 show the statistics about how each security concern has been addressed by the primary MDS approaches. Fig. 3.7 visualises other key results for a representative set of evaluation criteria. Tables 3.4, 3.5 summarise all the values for all evaluation criteria. We present the results for each evaluation criterion as follows.

Security concerns/mechanisms: Fig. 3.5 shows the statistic of security concerns tackled by the reviewed MDS approaches. We can see that *authorisation* is addressed the most, by 75% of the examined MDS papers. Moreover, more than half of the MDS papers (53%) deal with *authorisation* only (see Figs. 3.5, 3.6). The second security concern in terms of receiving attention is *confidentiality* addressed by 42% of the examined MDS papers. 11% of the examined MDS papers tackle *confidentiality* solely (see Figs. 3.5, 3.6). Other security concerns, like *integrity*, *authentication*, and *availability* are, however, less tackled with 27%, 24%, and 16% correspondingly. These results suggest that more MDS research work should focus on particular security concerns like *integrity*, *availability*, and *authentication*.

We also would like to know how much multiple security concerns are tackled at the same time by the MDS approaches. Fig. 3.6 displays the statistic about how much three key security concerns (Authentication, Authorisation, and Confidentiality) are tackled solely and simultaneously. Only 13% of the examined MDS papers propose methodologies to tackle all three together. About 15% of the examined MDS papers deal with two concerns simultaneously: Authentication and Authorisation (3%), Authentication and Confidentiality (6%), Confidentiality and Authorisation (6%). Not only multiple security concerns are less tackled, but also rarely the inter-relations among multiple security concerns are formally taken into account in the reviewed MDS approaches. Future MDS approaches should address multiple security concerns simultaneously, systematically by formally specifying inter-security concern relations. The inter-relation among security concerns have to be taken into account while developing DSLs for specifying security requirements.

These first results are very interesting. Indeed, an open question is "why in MDS *authorisation* and *confidentiality* got more attention?". A possible answer could be that MDS is a relatively young research area with more "model-driven" than "security". MDS is the common name of the MDE approaches specifically focusing on secure systems development. Thus, among the authors of the published MDS papers, there are significantly more researchers with MDE background than security engineering background. Researchers that mainly work with MDE techniques may first address *authorisation* (e.g. AC) because it is closer to application logic and functional requirements than other security concerns. This could be linked to the nature of security concerns. Some security concerns (e.g. *authorisation*) are closer to the application level than others. MDE researcher might not be familiar with security concerns to be addressed at the network layer. Given the background of the authors of the most renowned MDS approaches, it might be that we need more interest in MDE from the security engineering community to see more MDS approaches dealing with security concerns like *integrity*, *availability*, and *authentication*. Therefore, we suggest that more effort should be put into communicating MDE techniques as well as MDS approaches to the security engineering community.

Modeling approaches: Fig. 3.7a shows that 87% of the examined papers used standard UML models and defined DSLs for security concerns using the profile and stereotype mechanisms of the UML. 13% used other DSLs (e.g. [167], [152], or [169]). Thus, we understand that standardised, common UML models are broadly used by MDS approaches. On the other hand, defining DSLs (either UML profiles or other DSLs) is also very popular to leverage MDE techniques for secure systems development. UML profiles and other kinds of DSLs have been developed to better capture the specific semantics of security concerns. In other words, defining DSLs plays a key role in MDS because that way allows expressing security concepts/elements more easily. However, using UML profiles is not the only way for developing DSLs in MDS approaches. DSLs which are not UML profiles are also recommended, especially DSLs that can deal with multiple security concerns in the same system.

15% of the papers discuss approaches that are based on AOM (Fig. 3.7b) where security concerns are specified as aspects and eventually woven into primary models. Even though the remaining 85% are not really aspect-oriented, most of them still follow the *separation of concerns* principle and really separate security concerns from the main business logic [11]. In most of the cases, security concerns were specified separately from the business logic in PIMs and transformed into PSMs that can be refined into security infrastructures (e.g. *XACML*) integrated with the systems.

Security concerns are often modelled and analysed with a DSL that is concern-specific. But, few MDS papers have well-defined semantics for their languages so that these languages can be used for formal analysis. Only some papers related to the UMLsec, SecureUML approaches (see Section 3.5.2) provide some formal basis for security analyses. This shows that further efforts are required to mature security-specific modelling languages to foster analyses. Most (89%) of the MDS papers use structural models. Behavioural models are used in 31% of the reviewed MDS papers. Other types of models like domain specific models accounted for 13%. Using solely one type of models could not be enough to be able to express multiple security concerns. Thus, very few modelling approaches propose to deal with multiple security concerns together like [204, 85]. Most of them are specific to address only one security concern solely.

Model-to-model transformations (MMTs) & tools: Table 3.5 shows that 74% of the papers clearly mentioned MMTs while 26% did not use or mentioned transformations, e.g., because of a manual integration of security. More specifically, 57% of the examined papers use exogenous transformations. Most of these were used to transform PIMs to PSMs (Fig. 3.7d). Security concerns were modelled using DSLs for each concern to obtain PIMs that were transformed into PSMs, which can be refined into code. 19% define endogenous MMTs that are used to weave/compose security models into base models defined using the same DSLs.

34% of the examined MDS papers implement automatic MMTs, 6% describe semi-automatic (interactive) MMTs, and only 4% are manual (Fig. 3.7e). But 56% do not specifically provide any implementation information about MMTs, e.g. some simply provide mapping rules for transforming models. Having automated MMTs is one of the key success factors of MDE [99] and MMTs play a crucial role in MDS as well.

[11] Note that in this paper we only classified a modelling approach as AOM if a concern is modelled as an aspect model that can be woven into a primary model. We explained this point in Section 3.4.

Table 3.4: Results classified by the evaluation criteria

Evaluation criteria		# papers	%
Security concerns (overlapping)	Confidentiality	45	**42**
	Integrity	29	**27**
	Availability	17	**16**
	Authenticity	26	**24**
	Authorisation	81	**75**
Aspect-Oriented Modeling/AOSD	Yes	16	**15**
	No	92	**85**
Standard models	Yes(UML/UML profiles)	94	**87**
	Other DSLs	14	**13**
Type of models (overlapping)	Structural	96	**89**
	Behavioural	33	**31**
	Others	14	**13**

Especially some important semantics of security mechanisms might be embedded in the MMTs. Providing MMTs implementation details in MDS is important to evaluate the efficiency of each approach. It can be also helpful for other researchers to learn from previous experiences in choosing or developing a suitable transformation engine for their work. 19% of the selected MDS papers describe their MMTs implementation using standard transformation languages like ATL and QVT. 81% of the papers only describe the transformation rules without implementation details, or use other transformation languages like graph-based transformations, or specific (Java-based) compilers/tools.

Model-to-text transformations (MTTs) & tools: Table 3.5 shows that 64% of the papers describe MTTs or the generation of code or security infrastructures. 36% of the papers do not describe MTTs in details. Some mainly used models for verifying or analysing implemented secure systems, e.g. UMLsec where code/security infrastructure generation is mainly mentioned in future work. Comparing the purposes of MTTs, we can see in Fig. 3.7c that there are nearly as many MDS papers (34%) that only generate security infrastructure, such as XACML or security aspects code, as the MDS papers that describe generation of both code and security infrastructure (29%).

The tools used for code generation are not shown in Table 3.5 because there are too many different tools. Besides Eclipse-based MTT engines like Xpand[12], there are many cases where ad-hoc self-developed engines (e.g. Java-based tools, parsers, etc.) are used. A reason for that could be that many "ad-hoc" tools are preferred because of their specific support for a specific security domain. Ark [233][13], for example, transforms an input

[12] https://www.eclipse.org/modeling/m2t/?project=xpand

[13] extends the code generation engine of the openArchitectureWare framework that was already migrated into Eclipse as Xpand

Table 3.5: Results classified by the evaluation criteria

Evaluation criteria		# papers	%
Transformations used	Yes	80	**74**
	No/Unknown	28	**26**
Transformations level	Endogenous	20	**19**
	Exogenous	62	**57**
	Not Provided	26	**24**
Transformations automation	Automatic	37	**34**
	Semi-automatic	6	**6**
	Manual	4	**4**
	Not Provided	61	**56**
Standard Transformations	ATL/QVT	20	**19**
	Others/not mentioned	88	**81**
Code generation mentioned	Yes	69	**64**
	No	39	**36**
Code + Security Infrastructures generated	Yes	31	**29**
	Only Security Infrastructure	37	**34**
	Not Provided	40	**37**
Application Domains	IS/e-commerce	19	**18**
	Data warehouses	20	**19**
	Smart cards/ embedded systems	7	**6**
	Distributed Systems/SOA	34	**31**
	Others	28	**26**
Type of validation	Controlled experiment	2	**2**
	Industry case studies	5	**5**
	Academic case studies	72	**67**
	Example only	23	**21**
	Not Provided	6	**5**

UML model designed with the proposed UML profile into a skeleton of application code (program code and deployment descriptor). More ad-hoc Java-based tools like the one in [51] generates code (XACML policy files) from the constraints specified in SECTET-PL The tool uses Antlr [197], a compiler program for the syntax analysis of the constraints.

In general, MMTs and MTTs are widely used in MDS to improve the productivity of the development process. Most of the primary MDS approaches do mention to leverage MMTs and/or MTTs by describing transformation rules/intentions. However, more than half of the primary MDS approaches did not provide implementation details of MMTs or MTTs. Not many primary MDS approaches use standard transformation languages/tools like ATL or QVT but rather ad-hoc tools like Java-based compiler/tools for engineering security into the system. With the progress in the maturity of standard

MMT and MTT tools, they should be leveraged more in the future MDS approaches. Most of the MMTs in the selected studies are *exogenous* used for transforming PIMs to PSMs. The main reason is that there are many approaches (e.g. dealing with access control) generating only security infrastructure. Access control models (PIMs) often used to generate XACML configuration files (PSMs) for enforcing security policy. Another reason could be the lack of *all-round* approaches for the whole development cycle of secure systems which in the end lead to automatic generation of both code and security infrastructure. An *all-round* approach could follow AOM paradigm to fully leverage the automation of MMTs and MTTs for composing, transforming and generating both code and security infrastructure. Developing tool chains (based on MMTs and MTTs) to derive from models to implementation code is also an important piece of future work. Few complete tool chains to automate (most of) the MDS development process have emerged, but are still rare.

Application domains: Fig. 3.7f shows the main application domains that have been secured by MDS approaches. In general, these are distributed systems or SOA (31%), information systems or e-commerce (18%), data warehouses (19%), and smart cards/embedded systems (6%). The remaining MDS papers do not clearly state a domain, or could be generically applicable for different application domains, such as [204, 126, 171, 98].

Evaluation methods: Most of the papers (67%) describe academic case studies used to evaluate their approaches. There are still quite many MDS papers (21%) which only provide "running examples" to illustrate their approaches. Few MDS papers show controlled experiments (2%) and industry case studies (5%) in the evaluation of their approaches. There are very few papers that provide an in-depth evaluation like [59], [218], and [37]. Therefore, we suggest that more effort should be put in evaluating MDS approaches, e.g., with empirical studies or benchmarks.

3.5.2 Principal MDS Approaches

Altogether, the synthesised data show that there are currently several MDS approaches that have been proposed, used, and discussed in multiple publications. We would like to identify the most influential MDS approaches in terms of numbers of publications and citations. In total, five primary MDS approaches, which are called principal MDS approaches, have been identified. They are summarised in Tables 3.6, 3.7. Each has at least 7 primary MDS papers in our final set. The details of each approach, except Secure data warehouses, can be found in [144]. Here we briefly present each approach, and then compare some key points among them.

SECTET firstly aimed at securing web services by leveraging the Object Constraint Language (OCL) for specifying RBAC [9]. Based on that, a complete configured security infrastructure (XACML policy files) is generated. Later on, the authors proposed a specification language namely SECTET-PL (OCL-based) which is part of the SECTET framework for model-driven security for B2B workflows. In this framework, Constraint based RBAC (CRBAC) can be specified and then transformed into low-level web services standard artefacts [11]. SECTET-PL is also used for modelling restricted (RBAC-based) delegation in Service Oriented Architecture [12]. Their modelling approach is extended in [90, 89]. MMT and MTT are both carried out in a complete model-driven framework [87, 51, 49]. SECTET mainly addresses RBAC as its security concern and focuses on generating security infrastructure (XACML), not all the source code. Recently, Memon et al. [150] and also Katt et al. [119] propose two pattern refinement approaches based on SECTET framework that allows flexible configurations of SOA security.

Secure data warehouses (DWs) are the motivation for the work of developing MDS techniques for secure database development. This MDS approach is very specific for developing secure DWs. Fernández-Medina et al. [73, 74] extend OCL and UML for secure database development [75]. Their approach also uses UML profiles for modelling security enriched PIMs as inputs for a model-driven framework to create secure DW solutions [76, 219]. Secure PIMs can be transformed to secure PSMs by a set of formally defined QVT rules [217, 218, 216]. These PSMs can then be used for generating code with security properties. A similar MDS approach for developing secure XML data warehouses is presented in [231, 230, 229, 228] More recently, the above mentioned techniques for secure DW development are also leveraged in a reverse engineering style to modernise legacy DWs [41].

SecureMDD is proposed for facilitating the development of smart card applications based on UML models. In SecureMDD, UML class diagrams are used for modelling static aspects while UML sequence and activity diagrams are used for modelling dynamic aspects of a system [156]. From platform-independent UML models (PIMs) of a system, its formal abstract state machine (ASM) specification and Java Card code are generated. The generated abstract state machine specification is used for formally proving the correctness of the generated code regarding the security properties of the system. Thus, their MDS approach integrates MDE techniques with semi-formal and formal methods for verification as well as the implementation of security-critical applications [158, 159, 161]. The authors illustrated that SecureMDD is applicable for the development of large and complex secure Smart Card applications as well [160]. The main limitations of SecureMDD are its specific application domain and the lack of analysis for consistency between the UML models and the ASM model.

Table 3.6: Summary of the Principal MDS Approaches

	Security Concerns	Modelling Approach				MMT		MTT		Verification	Application Domains	Validation
		Language	AOM	SOC	Type	Level	Impl	Both	Impl			
Sectet [10, 12, 11, 9, 8, 49, 50, 51, 90, 92, 88, 91, 119, 150, 89]	mainly authorisation (access control, delegation), integrity, confidentiality, non-repudiation,	UML profiles	X	✓	S	Exo	QVT	X	XPAND	X	e-government, e-health, e-education, web services, SOA	ACS
SecureDWs [42, 41, 43, 44, 75, 73, 74, 76, 219, 218, 225, 224, 230, 229, 231, 228, 232]	privacy, integrity, authentication, availability, non-repudiation, auditing, access control	UML profiles	X	✓	S	Exo	QVT	X	MOF, CASE tool	X	web applications, databases	IE, ACS

Note: Supported (✓); Partially supported (o); Not supported (X); Controlled experiment (CE);
Industrial case study (ICS); Academic case study (ACS); Illustrative example (IE); Not provided (NP);
Non-restrictive (NR); Self-developed (Self-); Endogenous (Endo); Exogenous (Exo); Structural (S); Behavioural (B); Others (O)

Table 3.7: Summary of the Principal MDS Approaches

	Security Concerns	Modelling Approach				MMT		MTT		Verification	Application Domains	Validation
		Language	AOM	SOC	Type	Level	Impl	Both	Impl			
SecureMDD [156, 158, 157, 159, 161, 160, 45]	cryptography (secrecy, integrity, confidentiality), application-specific security properties	UML profiles	X	✓	S, B	Endo, Exo	QVT	✓	XPAND	KIV theorem prover, test cases from UML specifications	smart card and service applications	IE, ACS, ICS
SecureUML [143, 28, 32, 24, 25, 31, 47, 52, 59, 65, 29]	access control	UML profiles	o	✓	S	Endo	o	o	ArcStyler, ActionGUI, compiler (self-)	SecureMova model-checker	web applications	IE, ACS, ICS
UMLsec [112, 113, 106, 108, 81, 114, 115, 174, 175, 105]	confidentiality, integrity, authenticity, authorisation, freshness, information flow, non-repudiation, fair exchange	UML profiles	X	o	S, B	Endo ([108, 81])	X	X	compiler (self-)	AICALL theorem prover	web applications, embedded systems, distributed systems	IE, ACS, ICS

Note: Supported (✓); Partially supported (o); Not supported (X); Controlled experiment (CE);
Industrial case study (ICS); Academic case study (ACS); Illustrative example (IE); Not provided (NP);
Non-restrictive (NR); Self-developed (Self-); Endogenous (Endo); Exogenous (Exo); Structural (S); Behavioural (B); Others (O)

SecureUML is the approach which aims at bridging the gap between security modelling languages and design modelling languages. First, UML and UML profile are used for modelling application with role-based access control that can lead to generated complete access control infrastructures [143]. Then, Basin et al. [32] propose a UML-based language (UML profiles) with different dialects, which forms modelling languages (such as SecureUML + ComponentUML) for designing secure systems. Access control infrastructures for server-based applications can be generated automatically from models. Their work mainly focuses on access control constraints based on RBAC in design models. Semantics of SecureUML (and ComponentUML) are provided by Brucker et al. [52] and Basin et al. [24, 25] which enable formal analysis of security-design models. Based on this work, Clavel et al. show and discuss their practical experience of applying SecureUML in industrial settings [59]. Recently, the work on SecureUML has been continued by combining SecureUML + ComponentUML with a language for graphical user interfaces (GUI), namely ActionGUI [29, 28]. These modelling languages with MMT enable the full generation of security-aware GUIs from models for data-centric applications with access control policies. Another recent work by Dios et al. [65] makes use of ActionGUI for model driven development of a secure eHealth application. The main limitation of SecureUML is its sole focus on access control.

UMLsec is one of the most well-known UML-based approaches in MDS proposed early by Jürjens [112] and Jürjens [113]. Security requirements, threat scenarios, security concepts, security mechanisms, security primitives can be modeled by using security-related stereotypes (UML profiles), tags, goal trees. and security constraints. Thus, it is possible to formally analyse UMLsec diagrams against security requirements regarding their dynamic behaviours. Not like SecureUML only focusing on authorisation (e.g. access control), UMLsec addresses multiple security concerns such as *confidentiality*, *integrity* [106]. Not to a great extent but AOM is also used in the UMLsec approach [108]. Later on, UMLsec is deployed by Best et al. [37] in an industrial context for designing and analysing designs of distributed information systems. On the other hand, relevant tools support for UMLsec are presented in [115]. To tackle also social challenges in security, UMLsec was combined with Secure Tropos [173] to take on security from requirement engineering phase [174]. This work is then extended and applied to two different industrial case studies [175]. A more recent work related to UMLsec is by Jürjens et al. [114] for incremental security verification for evolving UMLsec models. However, UMLsec lacks support for improving productivity of the development process in terms of automated model transformations. Even having a view from models to code but the lack of automated transformation(s) from models to implementation code is a miss in UMLsec. Other than that, UMLsec could be considered as the most complete and mature MDS approach that deals with multiple security concerns, from very early at the requirement engineering level, with transformations, formal analysis possibility, tools support, industrial case studies.

In general, the most common point among the principal MDS approaches is that they all propose to use UML profiles in their modelling phase. Even though not following truly AOM, defining UML profiles as DSLs for modelling security concerns still allows these principal MDS approaches to have separation of concerns. Except SecureUML which only addresses access control, other approaches are able to touch multiple security concerns. Structural models are mainly used in all five approaches. SecureMDD and UMLsec have also used behavioural models. Exogenous MMTs are defined in SECTET and SecureDWs to transform PIMs (UML models) to PSMs. SecureUML and UMLsec integrate security into systems specified in UML using endogenous MMTs. SecureMDD combine both kinds of MMTs in their development process. Some standard transformation tools are used (e.g. QVT and XPAND) among other self-developed tools (java-based compilers). With their formal background, SecureMDD, SecureUML and UMLsec provide some tools for formal verification of security properties. These three also have industrial case studies while SECTET and SecureDWs have not. Generally, each approach is quite specific to a application domain, e.g. SecureDWs for secure database development, or SecureMDD for secure smart card development.

3.5.3 Less common/emerging MDS Approaches

It would not be fair to only discuss about the above-mentioned principal MDS approaches. There are other less common or emerging MDS approaches that are also worth to get noticed and analysed. We discuss some representative ones here. For the full list, readers are referred to Tables 3.8, 3.9, 3.10, and 3.11. The less common or emerging MDS approaches here are simply classified into several groups as follows.

Pattern-based MDS: Based on domain-independent, time-proven security knowledge and expertise, security patterns can guide security at each stage of the development process. Some MDS approaches that leverage security patterns are remarkable. Abramov et al. [1, 2, 3] propose an MDS framework for integrating access control policies into database development. At the pre-development stage, organisational policies are specified as security patterns. Then, the specified security patterns guide the definition and implementation of the security requirements which are defined as part of the data model. The database code can be generated automatically after the correct implementation of the security patterns has been verified at the design stage. Their approach has been evaluated in a controlled experiment [3]. Also using security patterns but at a different level of abstraction, Kim et al. [126, 127] develop a pattern-based technique for systematic, model-driven development of secure systems focusing on access control. Because this work mainly focuses on the design stage, access control is specified as design pattern. Bouaziz et al. [46] introduce a security pattern integration process for component-based models. With this process, security patterns can be integrated in the whole development process, from UML component modelling until aspect code generation. Another pattern-driven approach is proposed by Schnjakin et al. [210] for facilitating the configuration of security modules for service-based systems. The proposed security advisor enables the transformation from the general security goals, via security patterns at different abstraction level, to concrete security configurations. Menzel et al. [154] uses the security configuration patterns to operate the transformation of architecture models annotated with security intentions to security policies. The patterns that provide expert knowledge on Web Service security can be specified using a DSL. As using cloud services provided by cloud providers is getting more popular, Moral-Garcia et al. [163] recently propose an enterprise security pattern for securing Software as a Service. The security solution provided by the pattern can be driven by making design decisions whilst performing the transformation between the solution models. Specifically, from a Computation Independent Model (CIM), different PIMs can be derived based on different design decisions with security patterns. Those PIMs are transformed into PSMs which are then transformed into Product Dependent Models.

MDS for Security@Runtime: Many modern applications such as cloud-based software-as-a-service (SaaS) applications require the dynamic adaptation or even evolution of both security and service at runtime. More and more (MDS) approaches have been being proposed in this area. Almorsy et al. [16] introduce an approach called Model Driven Security Engineering at Runtime (MDSE@R). MDSE@R is based on a UML profile with tool supports for separately specifying base system and security, and then merging those models into a joint system-security model. Because security and system models are separated and loosely coupled, they can evolve more easily. Security controls are enforced dynamically into the target system at the code level. After that, in [15] the same authors leverage the MDSE@R approach for multi-tenant, cloud-hosted SaaS applications. This allows dynamically engineering security for multi-tenant SaaS applications at runtime. Recently, Almorsy et al. [14] develop a new DSL called SecDVSL for specifying visually a variety of security concepts like objectives, threats, requirement, architecture, and enforcement controls. SecDVSL also allows maintaining traceability among these security concepts. Not specifically for SaaS applications but component-based architecture, Morin et al. [167] leverage the notion of model@run.time to enable dynamically enforcing role-based access control policies into component-based systems. In the follow-up work, Nguyen et al. [184] deal with not only access control policies but also the more complex, but essential, delegation of rights mechanism. The propose MDS framework allows dynamically enforcing/weaving access control policies with various delegation features into security-critical systems. This is done with a flexibly dynamic adaptation strategy. Another runtime-update of security policy-based approach is presented by Elrakaiby et al. [69]. The introduced DSL called *Security@Runtime* covers many of the security requirements of modern applications such as authorisation, obligation, and reaction policies. Xiao [237]'s work is on adaptive and secure multi-agent systems. The authors adopting the adaptive agent model to put forward a security-aware model-driven mechanism by using an extension of RBAC model.

MDS for Secure SOA: Many MDS approaches focus on securing service-oriented systems (SOSs). Gilmore et al. [86] show how services, service compositions, and non-functional properties can be modelled using their self-developed UML profile and its extension. They address non-functional properties in general where security is considered with performance and reliable messaging. The models are the input for the framework VIATRA[14] to derive deployment mechanisms using MMT and MTT. Wada et al. [233] also address non-functional aspects in SOA with a MDD framework and tool support. Their work is empirically evaluated to show the improvement in the reusability and maintainability of service-oriented applications. More specifically to integrate security-related non-functional aspects in the development of services, Gallino et al. [83] present their MDS solution using multiple domain-specific models independently addressing security aspects. Hoisl et al. [96, 97] propose an MDS approach based on SoaML for specification and the enforcement of secure object flows in process-driven SOA. [153, 152] introduce a security metamodel for SOA. This metamodel is the base for their MDS framework that allows modelling of security requirements in system design models. Going further than modelling, Nakamura et al. [179] propose an MDS tooling framework to generate Web services security configurations. In the same line, intermediate model structure is introduced by Satoh et al. [207, 208] to simplify the transformation rules for transforming a security policy written in WebService-SecurityPolicy into platform-specific configuration files.

Aspect-Oriented Modelling in MDS: AOM techniques would be ideal for MDS with fully separation of concerns support. With AOM, security concerns can be modelled separately, and then automatically composed into primary models. All of the reviewed MDS approaches in this category except [200, 243] tackle multiple security concerns. These approaches aim at dealing with multiple security concerns as one would expect from any AOM approach. Georg et al. [85] propose a methodology that allows not only security mechanisms but also attacks to be modelled as aspect models. The attacks models can be composed with the primary model of the application to obtain

[14] http://www.eclipse.org/viatra/

TABLE 3.8: Summary of the less common/emerging MDS Approaches

	Security Concerns	Modelling Approach				MMT		MTT		Verification	Application Domains	Validation
		Language	AOM	SOC	Type	Level	Impl	Both	Impl			
Pattern-Based [1, 2, 3]	access control	UML	X	✓	S	Exo	ATL	✓	SMT (self-)	✓	database	ACS; CE
Pattern-Based [46]	access control	UML	X	✓	S	Endo	ATL	✓	NP	X	component-based architecture	ACS
Pattern-Based [126, 127]	access control	UML	✓	✓	S, B	Endo, Exo	NP	X	X	✓	NR	ACS
Pattern-Based by Schnjakin et al. [210]	integrity, confidentiality, authentication, authorisation	BPMN	X	✓	O	NP	X	X	NP	X	service-oriented architectures	IE
Pattern-Based by Moral-Garcia et al. [163]	integrity, confidentiality, availability, authentication, authorisation	DSL	X	✓	O	Exo	X	X	NP	X	secure cloud computing	IE
Sec@runtime by Almorsy et al. [16, 15, 14]	integrity, confidentiality, availability, authentication, authorisation	UML	✓	✓	S, B	Endo	NP	o	NP	testing	cloud-based applications	ACS, CE
Sec@runtime by Elrakaiby et al. [69]	authorisation	UML, DSL	✓	✓	S, DSM	Exo	NP	X	NP	X	NR	ACS
Sec@runtime by Morin et al. [167]	authorisation	DSL	X	✓	DSM	Exo	Kermeta	o	Kermeta	X	component-based architecture	ACS

Note: Supported (✓); Partially supported (o); Not supported (X); Controlled experiment (CE);
Industrial case study (ICS); Academic case study (ACS); Illustrative example (IE); Not provided (NP);
Non-restrictive (NR); Self-developed (Self-); Endogenous (Endo); Exogenous (Exo); Structural (S); Behavioural (B); Others (O)

Table 3.9: Summary of the less common/emerging MDS Approaches

	Security Concerns	Modelling Approach				MMT		MTT		Verification	Application Domains	Validation
		Language	AOM	SOC	Type	Level	Impl	Both	Impl			
Sec@runtime by Nguyen et al. [184]	authorisation (access control, delegation)	DSL	X	✓	DSM	Exo	Kermeta	o	Kermeta	X	component-based architecture	ACS
Sec@runtime by Xiao [237]	authorisation	DSL	DSM	X	✓	Exo	NP	✓	JADE(self-)	X	NR	ACS
SecureSOA by Gallino et al. [83]	authorisation	UML	X	✓	S	Exo	NP	o	NP	X	SOA	ACS
SecureSOA by Gilmore et al. [86]	non-functional aspects	UML profiles	X	o	S, B, O	Exo	VIATRA	✓	VIATRA2	✓	distributed systems	ACS
SecureSOA by Hoisl et al. [96, 97]	confidentiality, integrity	UML	X	✓	S, B	Exo	NP	NP	NP	X	SOA	ACS
SecureSOA by Menzel et al. [154, 153, 152]	confidentiality, integrity, authentication, authorisation	DSL	X	✓	DSM	Exo	NP	o	NP	X	SOA	ACS
SecureSOA by Nakamura et al. [179]	confidentiality, integrity, availability, authentication	UML profile	X	✓	S	Exo	NP	o	NP	X	SOA	IE
SecureSOA by Satoh et al. [207, 208]	authentication	DSL	X	✓	DSM	Endo	NP	o	NP	X	SOA	IE
SecureSOA by Wada et al. [233]	confidentiality, integrity, authorisation	UML profile	X	✓	S	Exo	ark	✓	ark (self-)	X	SOA	CE
SecureSOA by Wolter et al. [236]	integrity, confidentiality, availability, authentication, authorisation	DSL	X	✓	DSM	Exo	NP	o	NP	X	SOA	IE

Note: Supported (✓); Partially supported (o); Not supported (X); Controlled experiment (CE);
Industrial case study (ICS); Academic case study (ACS); Illustrative example (IE); Not provided (NP);
Non-restrictive (NR); Self-developed (Self-); Endogenous (Endo); Exogenous (Exo); Structural (S); Behavioural (B); Others (O)

Table 3.10: Summary of the less common/emerging MDS Approaches

	Security Concerns	Modelling Approach				MMT		MTT		Verification	Application Domains	Validation
		Language	AOM	SOC	Type	Level	Impl	Both	Impl			
AOMsec by Georg et al. [85]	integrity, confidentiality, availability, authentication, authorisation	UML	✓	✓	S, B	Endo	NP	NP	NP	✓	NR	IE
AOMsec by Mouheb et al. [171] and Mouheb et al. [172]	confidentiality, authorisation	UML	✓	✓	S, B, O	Endo	QVT	✓	RSA	X	NR	ACS
AOMsec by Ray et al. [200]	authorisation (AC)	UML	✓	✓	S, O	Endo	NP	NP	NP	X	NR	IE
AOMsec by Sánchez et al. [204]	confidentiality, integrity, authorisation	UML profile	✓	✓	S, B	Exo	QVT	o	NP	X	NR	ACS
AOMsec by Zhu et al. [243]	confidentiality, integrity, availability	UML profile	✓	✓	S, B	Exo	NP	o	aspect code gen (self-)	✓	NR	ICS
Access Control oriented by Ahn et al. [6]	authorisation	UML	X	X	S, O	NP	NP	o	Octopus + Dresden OCL toolkit	X	NR	CE
Access Control oriented by Burt et al. [54]	authorisation	DSL	X	✓	DSM	Exo	NP	NP	NP	X	NR	IE
Access Control oriented by Fink et al. [78]	authorisation	DSL	X	✓	DSM	Exo	Graph Transformation	NP	NP	X	NR	ACS
Access Control oriented by Kim et al. [128]	authorisation (AC)	UML	✓	✓	S, B	Endo	IBM RSA	✓	IBM RSA	o	NR	ACS
Access Control oriented by Mouelhi et al. [169]	authorisation	DSL	X	✓	DSM	Exo	NP	o	NP	✓(testing)	NR	ACS

Note: Supported (✓); Partially supported (o); Not supported (X); Controlled experiment (CE); Industrial case study (ICS); Academic case study (ACS); Illustrative example (IE); Not provided (NP); Non-restrictive (NR); Self-developed (Self-); Endogenous (Endo); Exogenous (Exo); Structural (S); Behavioural (B); Others (O)

Table 3.11: Summary of the less common/emerging MDS Approaches

	Security Concerns	Modelling Approach				MMT		MTT		Verification	Application Domains	Validation
		Language	AOM	SOC	Type	Level	Impl	Both	Impl			
Access Control oriented by Kaddani et al. [116]	authorisation (OrBAC)	UML	X	✓	S	Exo	NP	o	NP	X	electrical grid	IE
Access Control oriented by Pavlich-Mariscal et al. [198]	authorisation, authentication	UML profile	X	✓	S	Endo	NP	✓	self-	o	NR	ACS
Access Control oriented by Sohr et al. [215]	authorisation	UML	X	✓	S, B, O	NP	NP	NP	NP	X	distributed systems	ACS
Access Control oriented by Schefer-Wenzl et al. [209]	authorisation	UML profile	X	✓	S, B, O	Exo	NP	NP	NP	X	NR	ACS
Access Control oriented by Bertolino et al. [35]	authorisation	DSL	X	✓	S, B	Exo	Kermeta	o	NP	model-based testing	NR	ACS
Usage Control by Neisse et al. [180]	authorisation (UCON)	DSL	X	✓	DSM, S, B, O	Exo	Java-based tool	o	Java-based tool (self-)	X	NR	ACS
ModelSec by Sánchez et al. [203]	integrity, confidentiality, availability, authentication, authorisation	DSL (SecML)	X	✓	DSM	Exo	RubyTL	o	MOFScript	X	NR	ACS
Secure Web Apps by Busch et al. [55]	integrity, confidentiality, availability, authentication, authorisation	UML profile	X	✓	DSM	Exo	NP	o	XPand	testing possible	web applications	ACS
SecEmbedded by Eby et al. [68]	confidentiality, availability	DSL	X	✓	DSM	Exo	NP	NP	NP	X	embedded systems	ACS

Note: Supported (✓); Partially supported (o); Not supported (X); Controlled experiment (CE);
Industrial case study (ICS); Academic case study (ACS); Illustrative example (IE); Not provided (NP);
Non-restrictive (NR); Self-developed (Self-); Endogenous (Endo); Exogenous (Exo); Structural (S); Behavioural (B); Others (O)

the misuse model. The authors then use the Alloy Analyser [15] to reason about the misuse model. If the misuse model shows that the application is compromised, some security mechanism must be incorporated into the application. The Alloy Analyser is used again to verify that the secured application model is now resilient to the attack. Mouheb et al. [171] and Mouheb et al. [172] develop a UML profile that allows specifying security mechanisms as aspect models. The aspect models often go together with their integration specification. Their approach allows security aspects to be woven automatically into UML design models (class diagrams, state machine diagrams, sequence diagrams, and activity diagrams) [171]. In [172], the authors present a full security hardening approach, from design to implementation. Not only restricted to security aspects, Sánchez et al. [204] propose a MDD approach for all early aspects, including security. The difference with other approaches is that they focus on aspect-oriented requirements specifications (models). These aspect-oriented requirements models are then automatically transformed into aspect-oriented architecture models. Not dealing with multiple security concerns, Ray et al. [200] introduce an AOM approach for addressing access control. Specifically, RBAC aspects can be modelled using parameterised UML models as patterns. This allows uniformly incorporate pervasive access control functionality into a design. The woven model can be analysed to check the correctness of incorporation. Zhu et al. [243] propose a model-based aspect-oriented framework for building intrusion-aware software systems. There, attack scenarios and intrusion detection aspects are modelled using an aspect-oriented UML profile. The intrusion detection aspect models are used to automatically generate aspect-oriented codes. The aspect-oriented codes are woven into the target systems using an aspect weaver to obtain the intrusion-aware software system. Recently, Horcas et al. [98] propose a hybrid AOSD and MDE approach for automatically weaving a customised security model into the base application model. By using the Common Variability Language (CVL) and ATL, different security concerns can be woven into the base application in an aspect-oriented way, according to weaving patterns. However, inter-security concern relations have not been taken into account.

MDS for Access Control: Section 3.5.1 shows that access control problem got the most attention from the MDS community. We discuss here some representative MDS approaches that specifically address access control. Ahn et al. [6] propose a framework for representing security model, specifying and validating security policy, and automatically generating security enforcement codes. This framework leverages the MDD approach together with a systematic tool to build secure systems. Also presenting a MDD approach for access control, Fink et al. [78] aim at developing access control policies for distributed systems using MOF and UML profiles. However, this approach does not work well with

[15] http://alloy.mit.edu

module-based system like systems based on SOAP [16]. Kim et al. [128] present a feature-based approach that enables systematic configuration of RBAC features for developing customisable access control-based enterprise systems. Feature modelling is used for effectively capturing the variabilities of the RBAC. UML models are used for specifying the static and behavioural properties of RBAC features. The composition method in their approach is used for building RBAC configuration, which also serves as a verification point for correctness of composition. Aiming at a full design-to-testing MDD process, Mouelhi et al. [169] introduce a generic access control metamodel. The generic access control policy model specified by the metamodel is automatically transformed into security policy for the XACML platform, and integrated in the target application using aspect-oriented programming. Model-based mutation testing makes the access control enforcement quantitatively testable. Pavlich-Mariscal et al. [198] propose a MD framework with a set of composable access control features that can be tightly integrated into the UML. At the code level, access control is map to the policy code which realises access control diagrams and features, and the enforcement code, to restrict access to methods based on information of the policy code. The degree of traceability of mappings is assessed. Recently, Schefer-Wenzl et al. [209] propose a full MDD approach for specifying and enforcing break-glass policies in process-aware information systems. By tackling a complex security exception handling mechanism like break-glass policies with MDS, this work shows developing DSLs for specific security concerns are a good way to capture well the semantics of these concerns. Based on that, a typical MDD process can be developed for derive security from specification to enforcement with tools support. Bertolino et al. [35] even go further in terms of tools support by providing a toolchain for designing, generating. and testing access control policies. This toolchain is the result of integrating specific tools for specific stages of the development cycle that have been developed in a collaborative research network. The research around UMLsec has also resulted in various tools support but not yet systematically formed a tool chain.

Miscellaneous: Neisse et al. [180] present one of few MDS approaches about usage control, the next generation of access control. Consisting of authorisations and obligations, high-level usage control policies are specified considering an abstract system model and automatically refined with the help of policy refinement rules to implementation-level policies. The work by Elrakaiby et al. [69] mentioned above can also be categorised as usage control. In the domain of securing embedded systems, the approach we reviewed is by Eby et al. [68]. The authors propose a framework to incorporate security modelling into embedded system design. Their security analysis tool is capable of analysing the flow of data objects through a system and identifying points that are vulnerable to attack. Not restricted to a particular application domain, ModelSec by Sánchez

[16] http://www.w3.org/TR/soap/

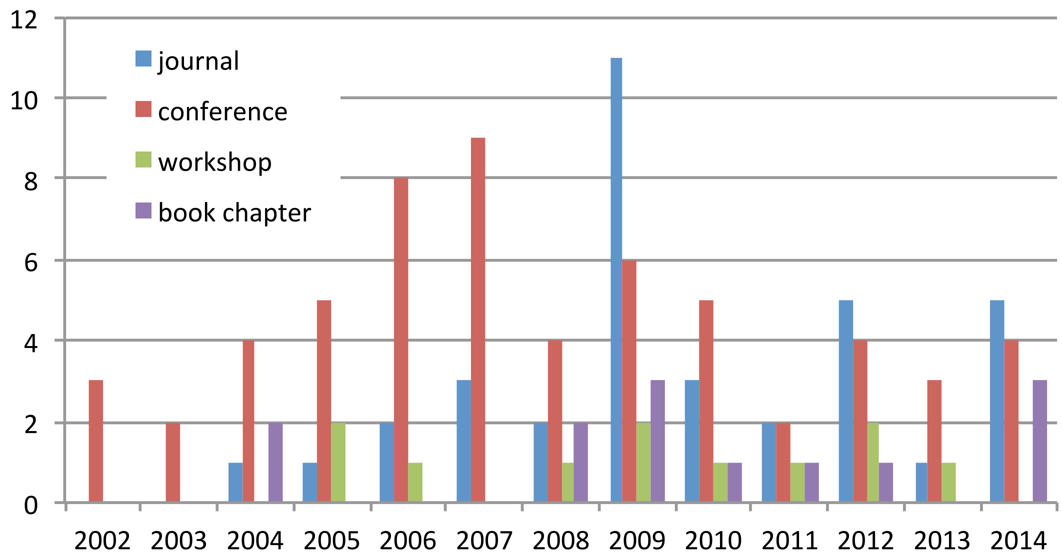


FIGURE 3.8: Trend of MDS publication

et al. [203] can deal with multiple security concerns in an integrated fashion, including privacy, integrity, access control, authentication, availability, non-repudiation, and auditing. MODELSEC supports defining and managing security requirements by building security requirements models for an application from which operational security models can then be generated. Recently, Busch et al. [55] present an MDS approach specific for securing web applications, tackling multiple security concerns. The graphical, UML-based Web Engineering (UWE) language is extended for specifying security concerns in web applications. Moreover, the approach is mapped to an iterative development cycle from requirement specification to testing and deployment with tools support.

3.5.4 Trend analysis of MDS approaches

In terms of publication, we can see in Fig. 3.8 there was a peak time for primary MDS publications in 2009. As we mentioned, the primary MDS approaches were first introduced from 2002. From 2002-2008, more primary MDS papers were published at conferences than journals. The number of primary MDS papers published at conferences were going up until 2007. In 2008, the number of primary MDS papers published at conferences decreased. One of the reasons could be primary MDS papers were under submission to journals. In 2009, there was a peak number of primary MDS papers published in journals. After the peak in 2009, the trend of primary MDS publications looks more stable for the period 2010-2014. From 2010 to 2014, less primary MDS papers were published than the previous 5-year period (2005-2009). However, the trend of publishing primary MDS papers in the period 2010-2014 seems more stable.

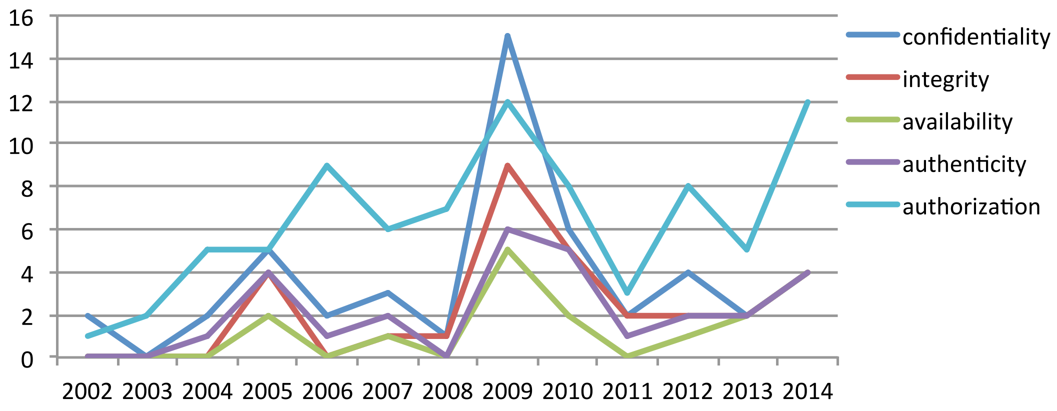


FIGURE 3.9: Trend of security concerns addressed by MDS studies

Similarly to the trend of publications, the trend of how security concerns have been addressed also has a peak time in 2009. Fig. 3.9 shows that, nearly all the time reviewed, authorisation is the concern that has been addressed the most. Only in 2009, confidentiality was tackled by more primary MDS papers than authorisation. The other concerns were always less focused than authorisation and confidentiality all the time reviewed. Until 2014, authorisation looks like still being addressed the most by the MDS research community. MDS researchers should pay more attention to the less tackled security concerns, and should aim at a solution addressing multiple security concerns simultaneously.

The trends of how MDE artefacts leveraged in the primary MDS approaches look well coupled with the number of primary MDS publications. The line of each artefact is very close to the others (see Fig. 3.10). This means that most primary MDS approaches did leverage the key artefacts of MDE in secure systems development. It is easily understandable that as long as we clearly define how an approach can be considered an MDS approach, most of the key MDE artefacts have to be leveraged in an MDS approach. This trend should hold in the future as well.

In terms of publication venues, Information and Software Technology (IST) journal and ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS) are so far the most popular venues for publication of primary MDS papers. Fig. 3.12 shows that at least 10 primary MDS publications have been found in each of these two venues. The next two attractive venues for primary MDS papers are ARES (security conference), and SoSym (MDE journal). Primary MDS papers were also published at some other general journals (Journal of Universal Computer Science) or domain specific conferences (IEEE International Conference on Web Services). The proceedings of Tutorial Lectures on Foundations of Security Analysis and Design (FOSAD) contains some significant primary MDS approaches as well. In general, except ARES

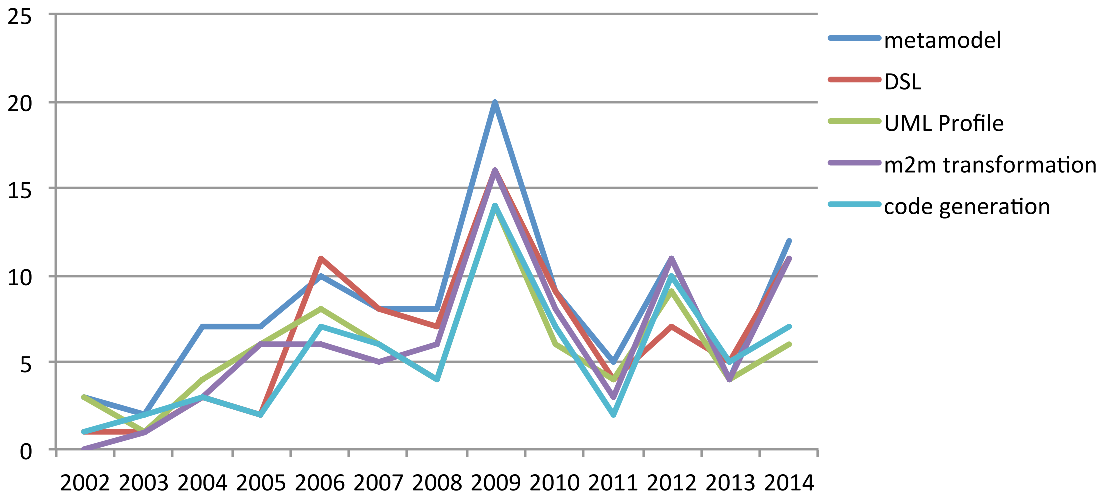


FIGURE 3.10: Trend of MDE artefacts leveraged by MDS studies

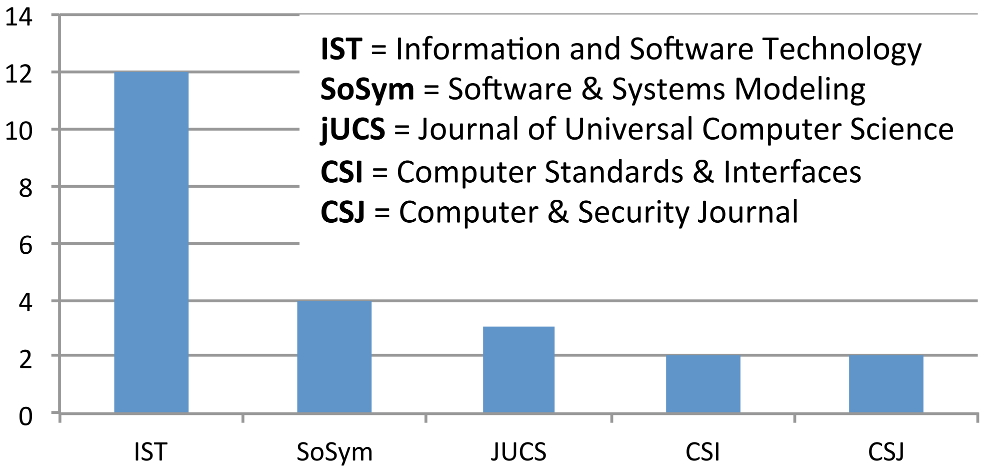
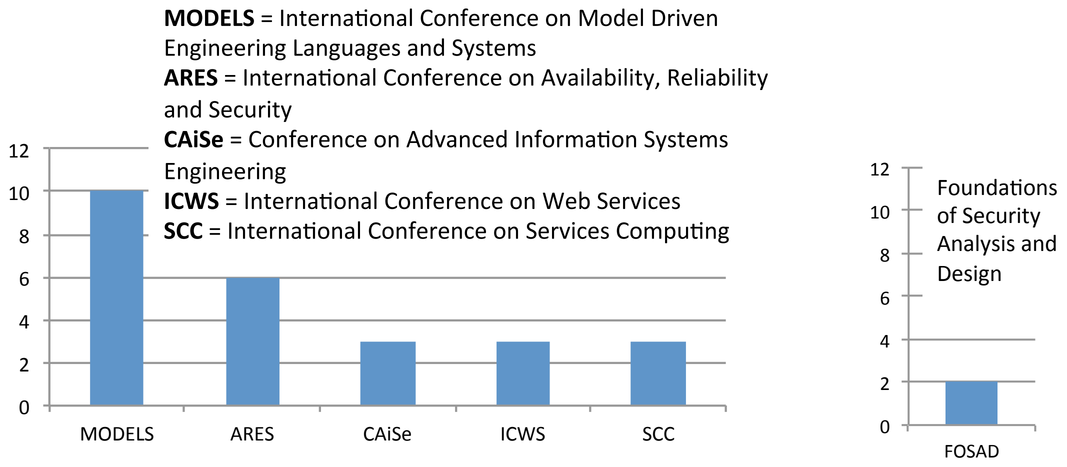


FIGURE 3.11: Number of papers for the journals with the most MDS papers found in this review

and CSJ, conferences and journals specific for security do not seem to be the common venues for MDS publications yet.

3.6 Threats to validity

We discuss the threats to validity of this SLR according to the lessons learned on validity in SLRs [129] and our own experience.



(A) conferences

(B) others

FIGURE 3.12: Number of papers for the conferences, workshops with the most MDS papers found in this review

3.6.1 The search process

To maximise the relevant articles returned by the search engines, we kept the search string not too specific but still reflecting what we wanted to search for. Moreover, the search string was used for searching not only in the titles, abstracts but also in the full text of an article. Only the search engine of Web of Knowledge (ISI) does not provide the option for searching in full text. This limitation could affect the search results returned by ISI. To minimise the possibility of missing relevant papers, we kept our search string generic so that we cover as many relevant papers as possible (more than 10 thousands relevant papers found). To complement for the automatic search, we have also conducted the manual search on relevant journals and proceedings of relevant conferences. Then, to mitigate the limitations of automatic and manual searches, we have adopted the snowballing strategy. Even though only three out of five steps of the snowballing strategy were adopted, those are the key steps. Moreover, we already conducted the extensive automatic and manual searches which covered thousands of relevant publications, and resulted in a large set of primary MDS papers. That is why conducting only three key steps of snowballing strategy would be fair enough. Another possible threat is that we did not extensively search for books related to MDS. However, we did include the option to also search for book chapters while performing automatic search. In fact, we found out some book chapters that got into our final selected papers for data extraction, e.g. [90], [106].

3.6.2 Selection of primary studies

A large part of the search and selection process was conducted by the first author. Some publications might have been missed. To mitigate this risk, every doubtful or "borderline" publication was not dismissed in the first place but rather being cross-checked and discussed by all the reviewers. Additionally, our clearly predefined review protocol with inclusion and exclusion criteria helped to reduce the reviewers' bias in the selection of primary studies.

The results of this SLR papers are based on the data extracted and synthesised from the selected MDS studies. Note that we have applied the citation criterion to estimate the quality and impact factors of the selected primary MDS studies. Even though this criterion is not too strict, applying it caused a number of MDS papers not to be included. We realised that some of the excluded MDS papers are related to the included primary MDS studies. To mitigate the risk of missing some important data of the primary MDS studies, we put back the excluded MDS papers that are related to the primary MDS studies. In total, we re-selected 15 MDS papers as the "sidekick" papers to be included in the final set for data extraction.

Some key selection criteria in this SLR are time-bound. The citation criterion for selecting primary MDS papers is based on the numbers of citations provided by Google Scholar engine. The selection of venues for conducting manual search is based on Microsoft Research ranking website. Google citations will change from time to time. Similarly, rankings of conferences and journals will change. Those time-bound metrics influence the reproduction of this SLR. So, some papers which were not selected as primary MDS papers because of the citation criterion would satisfy this criterion later on.

3.7 Related work

In [118], the authors present a survey on MDS. They propose an evaluation based on the work of Khwaja and Urban [120]. The study revealed that approaches that analyse implementations of modelled systems are still missing. Due to the fact that implementations are not generated automatically from formal specifications, verification of running code is reasonable. The main drawback of [118] is that it is not a SLR. As a result, there are some well-known approaches that are missing in [118], such as SecureUML [32].

In [27], Basin et al. went through a "Decade of Model-Driven Security" by presenting a survey focusing on their specific MDS approach called SecureUML. The authors claim that MDS has enormous potential, mainly because Security-Design Models provide a clear, declarative, high-level language for specifying security details. The potential is even more, when the security models rely on a well-defined semantics. The main drawback of [27] is that it only considers the work around SecureUML.

[227] is a survey of model-based security methodologies for distributed systems. The papers surveyed in [227] are not only about model-driven methodologies but also architecture-driven methodologies, pattern-driven methodologies, and agent-driven methodologies. Thus the focus is not specifically MDS but rather security engineering for distributed systems in general. Our review explicitly targets MDS methodologies as described in the previous sections.

In [144], five well-known MDS approaches, i.e. UMLsec, SecureUML, Sectet, ModelSec, and SecureMDD, are summarised, evaluated, and discussed. These five MDS approaches are also confirmed in this chapter. It can be seen that our SLR results are complementary to the contributions of the normal survey papers, e.g. [144], [227]. Those survey papers perform in depth analysis of some significant MDS approaches by elaborating one after another. But our SLR performs a SLR in both width and depth of MDS research which result in not only (evidently) significant MDS approaches but also emerging considerable MDS approaches. It is the first MDS literature review that systematically considers all relevant publications using explicit evaluation and extraction criteria. Furthermore, our SLR provides a detailed look at all the key artefacts of any MDS approaches such as modelling techniques, security concerns, how model transformations employed, how verification and validation methods used, and case studies, and application domains. We also provide a trend analysis for the development of MDS research area.

[103] is closer to our SLR. The authors propose three research questions with the goal to determine if the current MDS approaches focus on code generation and/or having empirical studies. The study shows that there is a need for more empirical studies on MDS (none exists), and that standardisation is key to achieve the objectives of MDD/MDA (which are increased portability and interoperability). However, [103] presents several drawbacks and differences from our work. First, their search strategy is very limited compared to our three-pronged search strategy. Second, concerning the SLR protocol, no evaluation criteria and data extraction strategy are given. Moreover, their exclusion criteria are very narrow. Consequently, the authors exclude significant papers in the field, e.g. UMLsec papers. Also, the authors exclude AOM approaches, because they consider that AOM does not consider security aspects as specific aspects (i.e. different from other aspects). Our work covers all the limitations of [103] and provides much more extensive SLR on the topic.

3.8 Conclusions

We have presented an extensive systematic literature review on the model-driven approaches for developing secure systems. The SLR is based on a rigorous three-pronged search process, which combined automatic search and manual search with snowballing strategy. Using 9 clearly predefined selection criteria, 108 MDS papers have been strictly selected, and then reviewed. From these primary MDS papers, we extracted and synthesised the data to answer three research questions: (RQ1) How do these approaches support the development of secure systems? (RQ2) What are the limitations? (RQ3) What are open issues to be further investigated?

(Our answers to RQ1.1) The results show that most MDS papers focus on *authorisation* (75 %) and *confidentiality* (42 %) while only few publications address other security concerns like *integrity*, *availability*, and *authentication*. Moreover, very few MDS papers deal with multiple security concerns simultaneously in a systematic way, e.g. only 9 % address authentication, authorisation, and confidentiality together. (RQ1.2) Most of the approaches try to separate security concerns from core business logic, but only few weave security aspects into primary models. The UML profile mechanism is often used for the definition of security-oriented DSLs, but some approaches have introduced non-UML based DSLs. It can be understandable that standardised, common UML models are broadly used by MDS approaches. Anyway, defining DSLs plays a key role in MDS because that way allows better capturing the specific semantics of security concerns. Still few security modelling languages are introduced with a thorough semantic foundation, which is needed for automated formal analyses. Most of the MDS papers use only structural models. Using solely one type of models could not be enough to be able to express multiple security concerns simultaneously. (RQ1.3 and RQ1.4) In MDS, MMTs and MTTs are often used but implementation details and tools are not often provided. Many examined MDS papers do not specifically provide any implementation information about MMTs. These papers just provide mapping rules for transforming models, or even without clearly defined transformation rules/mappings. Among the transformation tools provided or mentioned, not only general-purpose MMT and MTT tools but also many ad-hoc, specific (Java-based) tools are used. MMTs were mentioned in most of the identified MDS papers (74 %), but more than half of the papers do not provide detailed information on the used languages, tools, or transformation rules (56 %) and only a few mention standard transformation languages (19 %), such as ATL or QVT (RQ 1.4). More specifically, MTTs were mentioned slightly less often (64 %) than transformations to models and were used almost equally often to generate only security infrastructure (34 %) or also functional code (29 %). (RQ1.5) Most papers discuss illustrative examples or academic case studies (67 %) but do not mention in-depth evaluations, e.g. industrial case studies (5 %), controlled experiments (2 %) or common benchmarks. (RQ1.6) Although most papers do not mention a specific application domain there are domains that are discussed more frequently, such as distributed or service-oriented systems (31 %) and data warehouses (19 %).

Our answers to RQ2 and RQ3 can be derived from the details given in the answers to RQ1. (RQ2) More specifically, our SLR shows that many MDS approaches are limited to a specific, isolated security concern, especially access control. In many cases, the approaches are also too specialised to a certain application domain. Not only multiple security concerns are less tackled, but also the inter-relations among security concerns are rarely taken into account systematically in MDS approaches. Another important limitation is the lack of rigorous evaluations of claimed benefits and capabilities of MDS approaches. Last but not least, few complete tool chains to automate (most of) the MDS development process have emerged, but are very rare, and not yet adoptable by industry. (RQ3) All these findings urge for more attention from the MDS research community to the less tackled security concerns, to the solutions that can deal with multiple security concerns systematically. Leveraging Aspect-Oriented Modelling (AOM) techniques for better ensuring the separation-of-concern in MDS approaches should be promoted more. AOM could help to deal with multiple security concerns more systematically. Enhancing separation-of-concern in the development process, between engineering security and engineering main system functionalities, is important especially for developing complex software systems. Developing tool chains (based on MMTs and MTTs) to derive from models to implementation code is an important part of future work.

Independent of our initial research questions, our SLR revealed five significant MDS approaches that can be classified as more mature than the rest. But we also identified various emerging/less common MDS approaches that respond to recent developments, such as cloud-based environments. With trend analyses for the last twelve years we showed that there was a clear peak of publications on MDS in 2009, which mainly because of an increase in journal publications. Finally, our analysis of publication venues showed that the journal on Information and Software Technology and the MODELS conference published most of the identified MDS papers.

In future work, our SLR protocol and the list of finally selected MDS papers could be used for a follow-up SLR of MDS to identify papers that are published after this review. A reviewer would need to check again the citation criterion for those primary MDS papers using up-to-date citation numbers. After obtaining a subset of MDS papers from the original set, only forward snowballing would have to be conducted for this subset as backward snowballing cannot reveal newly published papers in references of old papers. After reviewing and selecting a new set of MDS papers from the result of forward snowballing, the full snowballing process could be performed on it to obtain a new final set. For the newly found papers in this final set, data extraction would have to be performed in order to obtain up-to-date results on new MDS publications.

Chapter 4

MDS-MoRe: MDS with Modularity and Reusability

Contents

This chapter presents an integrated framework (namely MDS-MoRe) which consists of different methodologies and techniques for addressing the three main open issues mentioned in Chapter 1. First, an overview of Model-Driven Security with Modularity and Reusability (MDS-MoRe) is given in Section 4.1. Section 4.2 presents the main stages of MDS-MoRe. Then, two different MDS approaches in the MDS-MoRe framework are presented in Section 4.3 and Section 4.4 respectively. Finally, we summarise all the main points of this chapter in Section 4.5.

4.1 Introduction

For addressing the three main open issues of MDS research mentioned in Chapter 1, a comprehensive MDS study called MDS-MoRe (**M**odel-**D**riven **S**ecurity with **Mo**dularity and **Re**usability) is proposed in this chapter. The MDS-MoRe study aims at tackling the three main open issues of MDS research above as well as promoting *modularity* and *reusability*. Taking into account security of complex systems makes the development of these systems much more complex. As indicated in [21], *"modularity is a concept that has* *proved useful in a large number of fields that deal with complex systems"*. Thus, we want to promote modularity in developing secure systems, from modelling till secure code generation and testing. On the other hand, design reuse has been long recognised as offering better payoff than code reuse [199]. To address multiple security concerns systematically, we want to promote a systematic reuse process of interrelated security patterns at the design level. Therefore, it is necessary for the proposed MDS-MoRe to leverage existing well-established techniques as well as develop relevant approaches to support model-driven development of secure systems with modularity and reusability. For that, (security) modelling techniques, model composition/transformation techniques, security (patterns) engineering techniques, and security testing techniques have to be analysed, leveraged, or developed to achieve the aim of MDS-MoRe.

In general, MDS-MoRe consists of two different MDS approaches. The former (see Chapter 5) develops a DSL for modelling a specific, complex security concern, i.e. delegation. Customised model-to-model transformations, code generation, and testing are also introduced to deal this specific, complex security concern from modelling till testing. The latter (see Chapter 6) leverages an AOM technique (RAM) to develop a *System of Security design Patterns* which can be used in systematically addressing multiple security concerns in developing secure systems. In the following section, the framework of MDS-MoRe study is described in detail. Moreover, the core phases of each main MDS approach such as modelling, model composition, code generation, testing, their purposes and their relations are presented.

4.2 The Model-Driven Framework of MDS-MoRe

Fig. 4.1 shows two different MDS approaches in the MDS-MoRe framework: one on the left hand side for dealing with a specific, complex security concern with extensive semantic, and another one on the right hand side for dealing with multiple security concerns systematically. The whole framework can roughly be divided into the following phases:

- **Modelling:** The task of this phase is to specify security concern(s) by using either domain-specific models or UML models. One important aspect of this phase is to really enforce separation of concerns between the security concerns and the core business logic of systems. Security concerns are modelled separately into security-oriented models. In the later phases, the security-oriented models will be integrated into the base model containing the business logic of the system to make the system secure.

- **Composing:** It is the process of transforming, weaving the security-oriented models into the base system model. The output of this phase is the composed model called security-enforced model.
- **Testing:** In this phase, test cases can be generated to evaluate the validation of the generated secure code. Testing techniques such as mutation analysis can be leveraged to improve the set of test cases.
- **Code Generation/Implementation:** From the security-enforced model, (partial) code generation can be performed to generate (the skeleton of) secure code.

Each MDS approach is presented separately according to these phases in the following sections.

4.3 MDS with Modularity

On the left hand side of Fig. 4.1, an MDS approach with modularity is presented. The framework is for dealing with a specific but complex security concern, i.e. delegation in access control management. Thus, a DSL for modelling delegation has been developed for capturing the complex semantics of delegation in access control management. The delegation models, access control models, and business logic models of a system can be specified separately. In fact, the target systems of this work are adaptive component-based systems in which leveraging modularity would enable the secure systems to evolve at runtime. On the other hand, mutation testing is employed for validating the implementation of delegation in access control management of the resulting system. The details of this MDS approach are presented in Chapter 5. The main ideas in each phase are described as follows, from modelling till testing and code generation.

4.3.1 Domain-Specific Modelling of Security Concerns

Access control is a popular security concern that has been focused in a majority of MDS studies as indicated in Chapter 3. So far, MDS studies mainly focus on static definitions of access control policies, without taking into account the more complex, but essential, delegation of rights mechanism. Delegation is a meta-level mechanism for administrating access rights, which allows a user without any specific administrative privileges to delegate his/her access rights to another user. As can be seen in Fig. 4.1, three DSLs have been developed for specifying access control, delegation and the business logic as separate concerns. A specific DSL is developed for capturing the

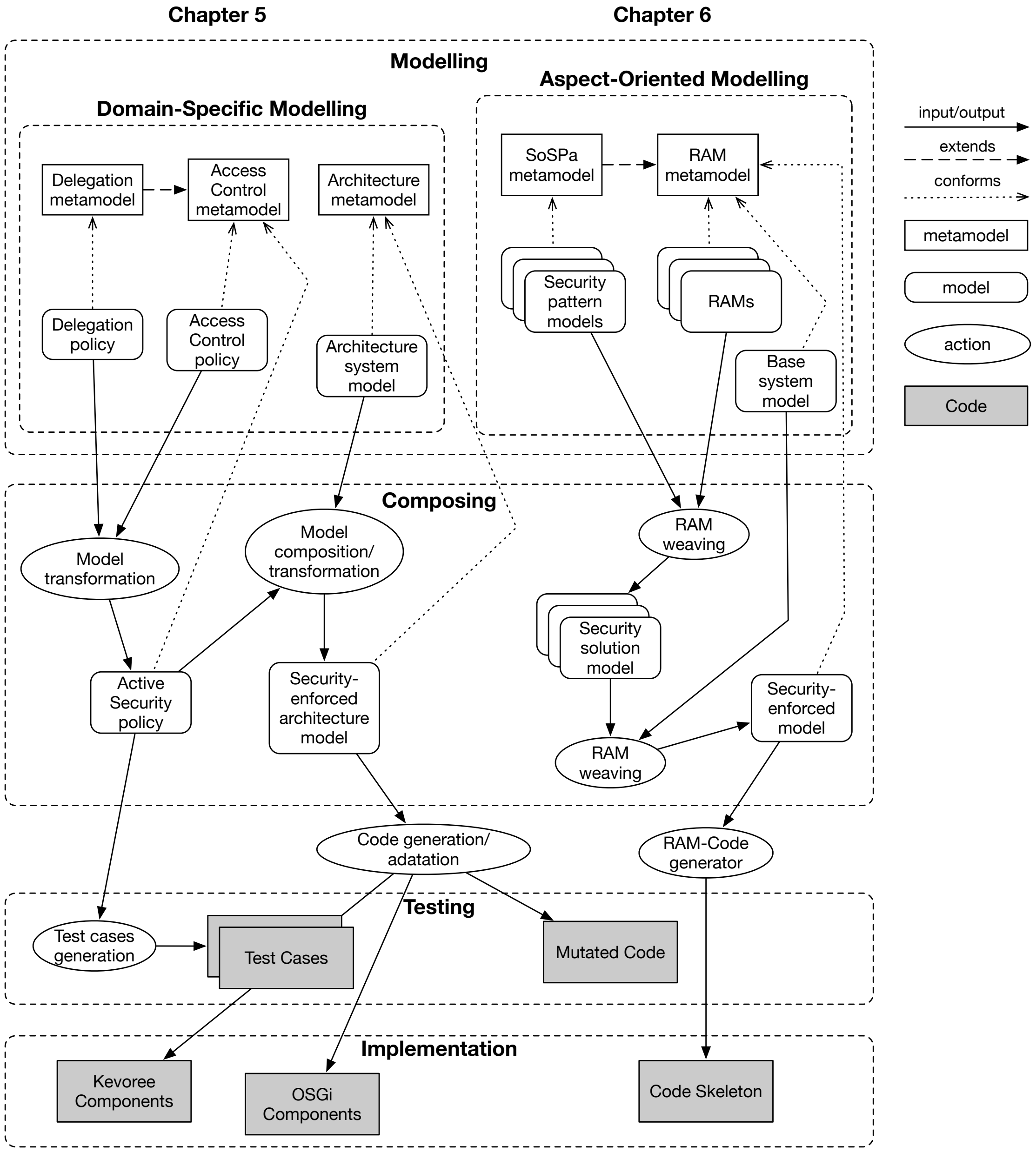


FIGURE 4.1: MDS-MoRe Framework

complex semantics of delegation in access control management. This DSL is an extension of another DSL which can be used to specify a Role-Based Access Control (RBAC) policy. Besides, the business logic of a target system is modelled by using the third DSL capturing component-based architecture. The output of this phase is the delegation policy model, the access control policy model, and the component-based architecture system model.

4.3.2 Transforming and Composing Models

Because of its complexity and relation to access control, delegation can be seen as a “meta-level” mechanism which impacts the existing access control policies similarly as an aspect can impact a base program. An ad-hoc MMT has been developed to transform the access control policy model according to the delegation rules specified in the delegation policy model. The result of this step is an active access control policy model reflecting both delegation rules and active access control rules. Then, the active access control policy model is transformed into a three-layer component based architecture model. This model is integrated into the architecture model of the base system for enforcing the security policy at the model level.

4.3.3 Testing

The meta-level characteristic together with the complexity of delegation itself make it crucial to ensure the correct enforcement and management of delegation policy in a system via testing. To this end, we adopt mutation analysis for delegation policies. In order to achieve this, a set of mutation operators specially designed for introducing mutants into the key components (features) of delegation is proposed. Test cases are derived from the security policy (see Fig. 4.1). The mutation operators are used to mutate the delegation policy and its enforcement which result in mutated code. The mutated code is then used to evaluate and improve the quality of the set of test cases.

4.3.4 Code Generation

Last but not least, (partial) code generation is another automated step to ensure the productivity of MDS in the secure systems development. From the security-enforced architecture model, the (partial) source code of the secure system in the target implementation platform can be generated. In fact, the DSLs are platform-independent. Because our target secure systems are of type component-based architecture, we show that the security-enforced architecture model can be used to generate components for two different adaptive execution platforms, i.e. Kevoree and OSGi. More details can be found in Chapter 5.

4.4 MDS with Reusability

On the right hand side of Fig. 4.1, another MDS approach promoting not only modularity but also reusability based on AOM is presented. The approach is for addressing multiple security concerns systematically. The catalogs of security patterns are the most accessible, well organised, documented resources of different security solutions for different security concerns, e.g. [72, 211, 220]. In our approach, we develop a unified *System of Security design Patterns* (SoSPa) based on RAM. Our SoSPa metamodel is an extension of RAM metamodel (see Fig. 4.1). The security pattern models, the supporting RAM models, and the base system model are specified at the modelling phase. At the composing phase, by using RAM weaver and a pattern refinement process, security solution models are constructed and woven into the base system model. The source code (skeleton) of the target system can be generated from the resulting model (security-enforced model). Chapter 6 presents the details of this MDS approach. The main ideas in each phase are described as follows.

4.4.1 Aspect-Oriented Modelling of Security Concerns

In SoSPa, security design patterns are collected, specified as reusable aspect models to form a coherent system of them that guides developers in systematically addressing multiple security concerns. More precisely, our MDS framework allows selecting, refining, composing security design patterns to systematically build security solution models, and then automatically integrating them into a target system design. Other RAM models and base system model are also specified in this phase.

4.4.2 Mapping and Weaving Models

In this phase, the mappings to integrate the newly built security solutions to a base system model are defined. Once all the necessary mappings and constraints are resolved, RAM weaver can execute the composition of the security solutions to a base system model to obtain the security-enforced system model.

4.4.3 Testing

We have some inspiration from the approach proposed by Kobashi et al. [133] where test templates could be introduced into the security design patterns for later validation of their application. This idea has not been realised yet in the work presented in this thesis. However, this is indeed a good point for future work.

4.4.4 Code Generation

The (skeleton) of source code can be generated automatically from the security-enforced model. This step is also supported by the RAM tool.

4.5 Summary

In this chapter, an integrated MDS framework with modularity and reusability, namely MDS-MoRe, is proposed for supporting the development of secure systems. MDS-MoRe is for addressing the three main open issues in the state of the art of MDS research. Two different MDS approaches are integrated in MDS-MoRe. Each MDS approach consists of the main MDS phases such as modelling, composing, testing, and code generation. The former leverages DSM for addressing a specific and complex security concern, i.e. delegation. The latter leverages UML, RAM, and a unified *System of Security design Patterns* to systematically address multiple security concerns. The details of these two MDS approaches can be found in Chapter 5 and Chapter 6 correspondingly.

Chapter 5

MDS with Modularity for Dynamic Adaptation of Secure Systems

Contents

Among the variety of models that have been studied in a *Model-Driven Security* perspective, one can mention *access control* models that specify the access rights. So far, these models mainly focus on static definitions of access control policies, without taking into account the more complex, but essential, *delegation* of rights mechanism. Delegation is a meta-level mechanism for administrating access rights, which allows a user without any specific administrative privileges to delegate his/her access rights to another user. This chapter gives a formalisation of access control and delegation mechanisms, and analyses the main hard-points for introducing various advanced delegation semantics in *Model-Driven Security*. Then, we propose a modular model-driven framework for 1) specifying access control, delegation and the business logic as separate concerns; 2) dynamically enforcing/weaving access control policies with various delegation features into security-critical systems; and 3) providing a flexibly dynamic adaptation strategy. We demonstrate the feasibility and effectiveness of our proposed solution through the proof-of-concept implementations of different component-based systems running on different adaptive execution platforms, i.e. OSGi and Kevoree.

5.1 Introduction

Software security is a polymorphic concept that encompasses different viewpoints (hacker, security officer, end-user) and raises complex management issues when considering the ever increasing complexity and dynamism of modern software. In this perspective, designing, implementing, and testing software for security is a hard task, especially because security is dynamic, meaning that a security policy can be updated at any time and that it must be kept aligned with the software evolution. As one of the key concerns in software security, managing *access control* to critical resources requires the dynamic enforcement of access control policies. Access control policies stipulate actors access rights to internal resources and ensure that users can only access the resources they are allowed to in a given context. A sound methodology supporting such security-critical systems development is extremely necessary because access control mechanisms cannot be "blindly" inserted into a system, but the overall system development must take access control aspects into account. Critical resources could be accessible to wrong (or even malicious) users just because of a small error in the specification or in the implementation of the access control policy.

Several design approaches like [167] [32] have been proposed to enable the enforcement of classical security models, such as *Role-Based Access Control* (RBAC) [77] [205]. These approaches bridge the gap from the high-level definition of an access control policy to its enforcement in the running software, automating the dynamic deployment of a given access control policy. Although such a bridge is a prerequisite for the dynamic administration of a given access control policy, it is not sufficient to offer the advanced administration instruments that are necessary to efficiently manage access control. In particular, *delegation* of rights is a complex dimension of access control that has not yet been addressed by the adaptive access control mechanisms. User delegation is necessary for assigning permissions from one user to another user. An expressive design of access control must extensively take into account delegation requirements.

Delegation models based on RBAC management have been characterised as *secure*, *flexible* and *efficient* access management for resource sharing, especially in a distributed environment. Flexible means that different subjects for delegation should be supported, i.e. delegation of roles, specific permissions or obligations. Also, different features of delegation should be supported, like temporary and recurrent delegation, transfer of role or permissions, delegation to multiple users, multi-step delegation, revocation, etc. However, the addition of flexibility for delegation must come with mechanisms to make sure that the security policy of the system is securely consistent. And last but not least, the administration of delegations must remain simple to be efficient. Thus, delegation is a complex problem to solve and to our best knowledge, there has been no complete approach for both specifying and dynamically enforcing access control policies by taking into account various features of delegation. Having such an expressive security model is crucial in order to simplify the administrative task and to manage collaborative work securely, especially with the increase in shared information and distributed systems.

Based on previous work [167], in this chapter we propose a new *Modular Model-Driven Security* solution to easily and separately specify 1) the business logic of the system without any security concern using a *Domain Specific Modelling Language* (DSML) for describing the architecture of a system in terms of components and bindings; 2) the "traditional" access control policy using a DSML based on a RBAC-based metamodel; 3) an advanced delegation policy based on a DSML dedicated to delegation management. In this third DSML, delegation can be seen as a "meta-level" mechanism which impacts the existing access control policies similarly as an aspect can impact a base program. The security enforcement is enabled by leveraging automated model transformation/-composition (from security model to architecture model). Consequently, in addition to [167], an advanced model composition is required to correctly handle the new delegation features. In this chapter, we claim that delegation needs to be clearly separated from access control because a delegation policy impacts access control rules. Therefore, delegation and access control are not at the same level and should be separated. This separation involves an advanced model composition approach to dynamically know, at any time, what is the set of new access controls that has to be considered, i.e., the "normal" access control rules as well as the access control rules modified by the delegation rules. From a more technical point of view, the security enforcement is dynamically done via automated model transformation/composition (from security model to architecture model) and the dynamic reconfiguration ability of modern adaptive execution platforms.

The remainder of this chapter is organised as follows. Section 5.2 presents the background on access control, delegation, and the security-driven model-based dynamic adaptation. Formal definitions of our access control model and formalisms of advanced delegation features are given in detail. Section 5.3 describes a running example. It will be used throughout this chapter to show the diverse characteristics of delegation and illustrate the various aspects of our approach. In Section 5.4, we first give an overview of our approach. Then, we formalise our delegation mechanisms based on RBAC and show how our delegation metamodel can be used to specify expressive access control policies that take into account various features of delegation. Based on the delegation metamodel, we describe our model transformation/composition rules used for transforming and weaving security policy into an architecture model. This section ends with a discussion of several strategies for dynamic adaptation and evolution of security policy. Section 5.5 describes how our approach has been applied and evaluated in the development of three different systems running on two different adaptive execution platforms. Next, related work is presented in Section 5.6. Finally, Section 5.8 concludes the chapter and discusses future work.

5.2 Background

This section introduces the main concepts which are used in this chapter. Firstly, formal definitions of *Access Control* and *Delegation* policies are presented. Based on these definitions, some key advanced delegation features are introduced formally. We keep all the definitions here generic so that they can be mapped into different security models like *Role-Based Access Control* (RBAC), *Organization-Based Access Control* (ORBAC) [117], *Discretionary Access Control* (DAC) [135], etc. These definitions also provide the basis for deriving mutation operators that can be used for testing delegation policy enforcement [191]. Then, a brief summary of previous work on dynamic security policy enforcement [167] is given.

5.2.1 Access Control

Access Control [101] is known as one of the most important security mechanisms. It enables the regulation of user access to system resources by enforcing access control policies. A policy defines a set of access control rules which expresses: who has the right to access a given resource or not, and the way to access it, i.e. which actions a user can access under which conditions or contexts.

Definition 5.1 (Access Control)**.** Let U be a set of users, P be a set of permissions, and C be a set of contexts. An access control policy AC is defined as a user-permission-context assignment relation: $AC \subseteq U \times P \times C$. A user u is granted permission p in a given context c if and only if $(u, p, c) \in AC$.

Additional details about contexts are given in next Sub-Section 5.2.2.1.

5.2.2 Delegation

In the field of access control, delegation is a very complex but important aspect that plays a key role in the administration mechanism [33]. A software system which supports delegation, should allow its users without any specific administrative privileges to grant some authorisations. There are two types of authorisation can be delegated: right and obligation. This thesis studies right delegation, not obligation delegation. In this thesis, "delegation" is understood as "right delegation".

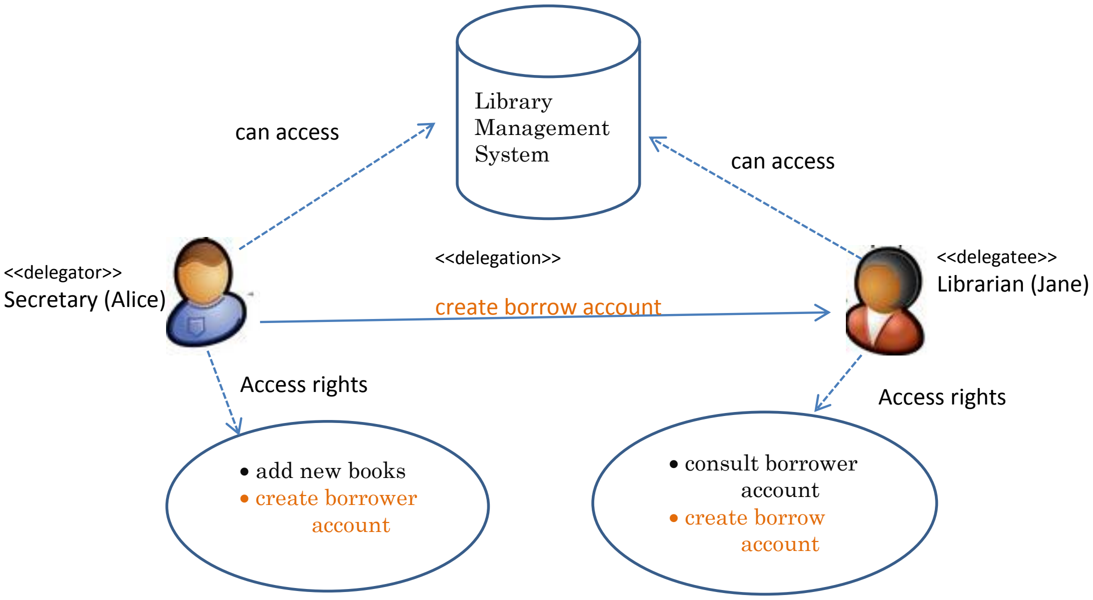


FIGURE 5.1: A simple example of Delegation Process

Delegation of rights allows a user, called the *delegator*, to delegate his/her access rights to another user, called the *delegatee*. By this delegation, the delegatee is allowed to perform the delegated roles/permissions on behalf of the delegator [61]. The delegator has full responsibility and accountability for the delegated accesses since he/she provides the accesses to the resources to other users, who are not initially authorised by the access control rules to access these resources. The basic concept of delegation is presented in Figure 5.1. In this example, Alice (*Secretary*) and Jane (*Librarian*) have access to the resources of LMS with their personal rights, i.e. Alice as a secretary can add new books, and create borrower account, while Jane as a librarian can consult borrower account. For some reason, Alice wants to delegate her permission of creating borrow account to Jane. As described by the arrow from Alice to Jane in Figure 5.1, Alice (delegator) is delegating the permission of "create borrower account" to Jane (delegatee). Once Alice has been delegating this permission to Jane, it is added to the access rights of Jane. By this delegation, Jane can create borrower account while being delegated by Alice on this permission.

A delegation policy can be considered as an administration-related security policy that is built on top of an access control policy. It is composed of delegation rules that can be specified at two levels: master-level and user-level. Basically, a delegation policy is two-fold:

1. It specifies who has the right to delegate which permission (for accessing to a given resource/action/subject) to whom, and in which context. We call this kind of rule *master-level delegation rule* as such a rule is normally defined by security officers. For example, a security officer can define a rule to specify that the head of a department at a university can *only* delegate the permission of updating personnel accounts to a professor.

Definition 5.2 (Master-Level Delegation Policy)**.** Let U be a set of users, P be a set of permissions, and C be a set of contexts. A master-level delegation policy MLD is defined as a user-user-permission-context assignment relation: $MLD \subseteq U \times U \times P \times C$ with the following meaning. A delegation of a permission p from a user u_1 to a user u_2 in a given context c **is allowed** if and only if $(u_1, u_2, p, c) \in MLD$.

2. It specifies who delegates to whom which permission, and in which context. We call this kind of rule *user-level delegation rule* as these rules are mostly defined by normal users. Note that user-level delegation rules must conform to master-level delegation rules. For example, *Bill* (the head of department) delegates his permission of updating personnel accounts to *Bob* (a professor) during his absence.

Definition 5.3 (User-Level Delegation Policy)**.** Let U be a set of users, P be a set of permissions, and C be a set of contexts. A user-level delegation policy ULD is defined as a user-user-permission-context assignment relation: $ULD \subseteq U \times U \times P \times C$ with the following meaning. A user u_2 **has** a permission p **by delegation** from a user u_1 in a given context c if and only if $(u_1, u_2, p, c) \in ULD$. It can be seen that all the delegations in ULD conform to the rules defined in the MLD. In other words, every delegation at the user-level can only be created if it conforms to the delegation rules defined at the master-level.

5.2.2.1 Context

A context is a condition or a combination of conditions in which an access control/delegation rule is active, i.e. enforced in the running system. Cuppens et al. discuss five different kinds of contexts in [63]. These kinds of contexts include temporal context, spatial context, user-declared context, prerequisite context, and provisional context. Temporal delegation is delegation within a time constraint, for example delegation is active for two days, or delegation is active for the time the delegator is on vacation. The spatial context relies on the delegator/delegatee's location, e.g. a delegated permission is only active when the delegatee is at office. User-declared context is related to the purpose of the delegator/delegatee, e.g. a delegator may state that his/her delegatee cannot further delegate his/her permissions to someone else. Prerequisite context allows delegation when some precondition is satisfied and the provisional context depends on the previous actions that delegator/delegatee has performed on the system. Moreover, it is possible for a security rule to have a complex context, which is a composition of contexts. Our security model supports context composition using conjunction &, disjunction $\oplus$, and negation $\bar{\ }$.

Note that every access control rule and delegation rule defined in this chapter is always associated with a context c. By default, if not specified explicitly, a context c is at least composed of a condition, called *Default*, i.e. always true.

5.2.3 Advanced Delegation Features

Delegation is a powerful and very useful way to augment access control policy administration. On one hand, it allows users to temporarily modify the access control policy by delegating access rights. By delegation, a delegatee can perform the delegated job, without requiring the intervention of the security officer. On the other hand, the delegator and/or some specific authorised users should be supported to revoke the delegation either manually or automatically. In both cases, the administrative task can be simplified and collaborative work can be managed securely, especially with the increase in shared information and distributed systems [5]. However, the simpler the administrative task can be, the more complex features of delegation have to be properly specified and enforced in the software system. To the best of our knowledge, there is no approach for both specifying and dynamically enforcing access control policies taking into account all delegation features like temporary delegation, transfer delegation, multiple delegation, multi-step delegation, etc.

In this section, we define the most well-known complex delegation features and formally specify them w.r.t. the definitions of access control and delegation policies. In the following definitions, we use *pre*, *body*, and *post* to respectively specify the state of the policy before changing, the state while it is being changed by the function (the delegation rule is being enforced), and the state after changing.

5.2.3.1 Monotonicity of Delegation

Monotonicity of delegation refers to whether or not the delegator can still use the permission while delegating it [61]. If the delegator can still use the permission while delegating it, the delegation is called grant delegation. Of course, the delegatee can use the permission while it is delegated to him. This is monotonic because available authorisations (in the set AC) are increased due to successful delegation operations. Again, note that every delegation can only be performed if and only if it satisfies the master-level delegation policy.

Definition 5.4 (Grant Delegation)**.** $grantDelegation(u_1, u_2, p, c) : -$
pre $(u_1, p, c) \in AC \wedge (u_2, p, c) \notin AC \wedge (u_1, u_2, p, c) \in MLD$
body $AC := AC \cup \{(u_2, p, c)\}$; $ULD := ULD \cup \{(u_1, u_2, p, c)\}$ end
post $(u_1, p, c) \in AC \wedge (u_2, p, c) \in AC \wedge (u_1, u_2, p, c) \in ULD$

Vice versa, if the delegator can not use the permission while delegating it, the delegation is called transfer delegation. As such, this is non-monotonic because available authorisations (in AC) are not increased due to successful delegation operations.

Definition 5.5 (Transfer Delegation)**.** $transferDelegation(u_1, u_2, p, c) : -$
pre $(u_1, p, c) \in AC \wedge (u_2, p, c) \notin AC \wedge (u_1, u_2, p, c) \in MLD$
body $AC := AC \setminus \{(u_1, p, c)\}$; $AC := AC \cup \{(u_2, p, c)\}$; $ULD := ULD \cup \{(u_1, u_2, p, c)\}$
end
post $(u_1, p, c) \notin AC \wedge (u_2, p, c) \in AC \wedge (u_1, u_2, p, c) \in ULD$

5.2.3.2 Temporary Delegation

This is also a very common feature of delegation needed by users. When revocation is handled automatically, the delegation is called temporary. In this case, the delegator specifies the temporal conditions in which this delegation applies: only at a given time, after or before a given time, or during a given time interval. The temporal conditions may correspond to a day of the week, or to a time of the day, etc. If the temporal context is not used, the delegation needs to be revoked manually.

Definition 5.6 (Temporary Delegation)**.** Let c be a given context of a delegation (either grant delegation or transfer delegation). A delegation is specified as temporary if its context c is associated with a time constraint. The delegation will only be active while the time constraint is satisfied.

For example, if the context is *vacation period*, a delegator *Bill* could have an associated delegation rule with the following context:

$c := c\&vacation\_period(startDate, endDate)$
where $vacation\_period(startDate, endDate) : -$
$startDate \leq endDate \wedge afterDate(startDate) \wedge beforeDate(endDate)$

Here, $afterDate(date)$ returns $true$ iff $date$ is equal or later than the current date. Similarly, $beforeDate(date)$ returns $true$ iff $date$ is equal or earlier than the current date.

5.2.3.3 Multiple Delegation

A permission can be delegated to more than a delegatee at a given time. However, the number of times that a permission is concurrently delegated have to be controlled. Multiple delegation refers to the maximum number of times that a permission can be delegated at a given time.

Definition 5.7 (Multiple Delegation)**.** Let N_m be the maximum number of times that a permission can be concurrently delegated. N_m is predefined by the security officer. The number of concurrent delegations in which the same role or permission is delegated at a given time, in a given context can not exceed N_m.

We introduce a counting function to count the number of delegations of a permission which is delegated by a delegator in a given context. The number returned by this function is always updated according to the change in the delegation policy, i.e. the number of delegation rules related to permission p.

$countDelegation(u, p, c) := |\{(u, v, p, c) \mid \forall v \in U : (u, v, p, c) \in ULD\}|$

If the number of concurrent delegations of the same permission at a given time, in a given context has not exceeded N_m, then this permission is still allowed to be delegated.

$grantDelegation(u_1, u_2, p,$
$c\&countDelegation(u_1, p, c) < N_m) : -$ pre $(u_1, p, c) \in AC \wedge (u_2, p, c) \notin AC \wedge (u_1, u_2, p, c) \in$
$MLD \wedge countDelegation(u_1, p, c) < N_m$
body $AC := AC \cup \{(u_2, p, c)\}$; $ULD := ULD \cup \{(u_1, u_2, p, c)\}$ end
post $(u_1, p, c) \in AC \wedge (u_2, p, c) \in AC \wedge (u_1, u_2, p, c) \in ULD$

5.2.3.4 Multi-step Delegation

This characteristic refers to the maximum number of steps (N_s, normally specified by a security officer) that a permission p can be re-delegated, counted from the first delegator of this permission. So if $N_s = 0$ that means the permission p can not be re-delegated anymore.

Definition 5.8 (Multi-step Delegation)**.** Let $N_s \geq 0$ be the maximum number of steps that a permission p can be re-delegated. A permission p can only be delegated *iff* $N_s > 0$.

First, let us define a helper function that returns the number of times a permission p is re-delegated in a given context c. $stepCounter(u_0, p, c) := N_s$ where u_0 is the first delegator of p in the delegation chain: u_0 delegates p to ... in a given context c; ... re-delegates p to u_1 in context c; and u_1 re-delegates p to u_2 in context c. Here, "..." is the users in the middle of the delegation chain, u_1 is the current last delegatee of this chain, and u_2 is the next delegatee if $stepCounter(u_1, p, c) \geq 1$.

If there exists a predefined maximum number of steps N_s for a permission p as described above, the delegation is specified as following.

$grantDelegation(u_1, u_2, p, c$
$\&stepCounter(u_1, p, c) \geq 1) : -$
pre $(u_1, p, c) \in AC \wedge (u_2, p, c) \notin AC \wedge (u_1, u_2, p, c) \in MLD \wedge stepCounter(u_1, p, c) \geq 1$
body $AC := AC \cup \{(u_2, p, c)\}$; $ULD := ULD \cup \{(u_1, u_2, p, c)\}$; $stepCounter(u_2, p, c) := stepCounter(u_1, p, c) - 1$ end
post $(u_1, p, c) \in AC \wedge (u_2, p, c) \in AC \wedge (u_1, u_2, p, c) \in ULD$

5.2.3.5 Delegation Revocation

Delegation supports a revocation feature in which a delegation can be revoked and permissions are returned back to the original user.

Definition 5.9 (Delegation Revocation)**.** Delegation revocation is the ability for any delegation can be manually revoked by authorised users.

The revocation of a grant delegation means to deny access of the delegatee to the delegated permission.

Definition 5.10. $revokeGrantDelegation(u_1, u_2, p, c) : -$
pre $(u_1, p, c) \in AC \wedge (u_2, p, c) \in AC \wedge (u_1, u_2, p, c) \in ULD$
body $AC := AC \setminus \{(u_2, p, c)\}$; $ULD := ULD \setminus \{(u_1, u_2, p, c)\}$ end
post $(u_1, p, c) \in AC \wedge (u_2, p, c) \notin AC \wedge (u_1, u_2, p, c) \notin ULD$

The permission to be revoked is deleted from the access rights of the delegatee. To revoke a transfer delegation, it is not only to deny access of the delegatee to the delegated permission but also to re-grant access to the delegator who is temporarily not having this access.

Definition 5.11. $revokeTransferDelegation(u_1, u_2, p, c) :-$
pre $(u_2, p, c) \in AC \wedge (u_1, p, c) \notin AC \wedge (u_1, u_2, p, c) \in ULD$
body $AC := AC \setminus \{(u_2, p, c)\};\ AC := AC \cup \{(u_1, p, c)\};\ ULD := ULD \setminus \{(u_1, u_2, p, c)\}$
end
post $(u_1, p, c) \in AC \wedge (u_2, p, c) \notin AC \wedge (u_1, u_2, p, c) \notin ULD$

We have presented formal definitions of access control, delegation, and various delegation features. These definitions are generic (at the conceptual level) so that they can be mapped into different security models like RBAC, ORBAC, DAC, etc. Section 5.4 shows how these formal concepts can be implemented (based on RBAC) using MDE techniques.

5.2.4 Security-Driven Model-Based Dynamic Adaptation

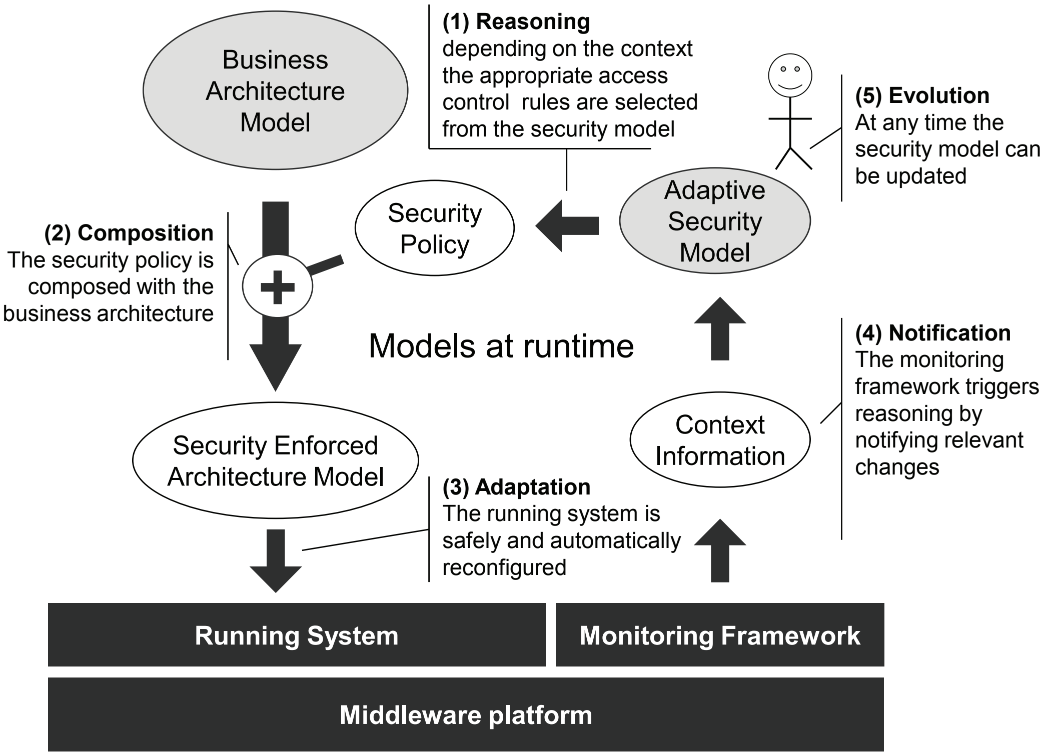

Business Architecture Model
(1) Reasoning depending on the context the appropriate access control rules are selected from the security model
(5) Evolution At any time the security model can be updated
Security Policy
Adaptive Security Model
(2) Composition The security policy is composed with the business architecture
Models at runtime
(4) Notification The monitoring framework triggers reasoning by notifying relevant changes
Context Information
Security Enforced Architecture Model
(3) Adaptation The running system is safely and automatically reconfigured
Running System
Monitoring Framework
Middleware platform

FIGURE 5.2: Overview of the Model-Driven Security Approach of [167]

In [167], the authors have proposed to leverage MDE techniques to provide a very flexible approach for managing access control. The different steps of this approach are summed up in Figure 5.2. On one hand, access control policies are defined by security experts, using a DSML, which describes the concepts of access control, as well as their relationships. On the other hand, the application is designed using another DSML for describing the architecture of a system in terms of components and bindings. This component-based software architecture only contains the business components of the application, which encapsulate the functionalities of the system, without any security concern. Then, the authors define mappings between both DSMLs describing how security concepts are mapped to architectural concepts. These mappings are used to fully generate an architecture that enforces the security rules. When the security policy is updated, the architecture is also updated. Finally, the proposed technique leverages the notion of *models@runtime* [164] in order to keep the architectural model (itself synchronised with the access control model) synchronised with the running system. This way, the running system can be dynamically updated in order to reflect changes in the security policy. Only users who have the right to access a resource can actually access this resource.

5.3 A Running Example

In this section, we give a motivating example which will be used throughout this chapter for describing the diverse characteristics of delegation and illustrating the various aspects of our approach.

Let us consider a *Library Management System* (LMS) providing library services with security concerns like access control and delegation management. There are two types of user account: *personnel accounts* (director, secretary, administrator and librarian) are managed by an administrator; and *borrower accounts* (lecturer and student) are managed by a *secretary*. The *director* of the library has the same accesses as a secretary, but additionally, he can also consult the personnel accounts. The *librarian* can consult the borrower accounts. A secretary can add new books in the LMS when they are delivered. *Lecturers* and *students* can borrow, reserve and return books, etc. In general, the library is organised with the following entities and security rules.

Roles (users): access rights (e.g. working days)
Director (*Bill*): consult personnel account, consult, create, update, and delete borrower account.
Secretary (*Bob* and *Alice*): consult, create, update, and delete borrower account, add book.
Administrator (*Sam* and *Tom*): consult, create, update, and delete personnel account.
Librarian (*Jane* and *John*): consult borrower account, find book by state, find book by keyword, report a book damaged, report a book repaired, fix a book.

Lecturer (*Paul*) and **Student** (*Mary*): find book by keyword, reserve, borrow and return book.

Resources and actions to be protected

Personnel Account: consult, create, update, and delete personnel account.
Borrower Account: consult, create, update, and delete borrower account.
Book: report a book damaged, report a book repaired, borrow a book, deliver a book, find book by keyword, find book by state, fix a book, reserve a book, return a book

In this organisation, users may need to delegate some of their authorities to other users. For instance, the director may need the help of a secretary to replace him during his absence. A librarian may delegate his/her authorities to an administrator during a maintenance day.

It is possible to only specify role or action delegations by using the DSML described in [167]. For instance, a role delegation rule can be created to specify that *Bill*, the director (prior to his vacation) delegates his role to *Bob*, one of his secretaries. But it is impossible for *Bill* to define whether or not *Bob* can re-delegate the *director* role to someone else (in case *Bob* is also absent for some reason). The role delegation of *Bill* to *Bob* is also handled manually: it is enforced when *Bill* creates the delegation rule and only revoked when *Bill* deletes this rule. There is no way for *Bill* to define a temporary delegation where its active duration is automatically handled. Obviously the DSML described in [167] is not expressive enough to specify complex characteristics of delegation.

There are many delegation situations that should be supported by the system. We give some delegation situations of the LMS as follows:
1. The director (*Bill*) delegates his role to a secretary (*Bob*) during his vacation (the delegation is automatically activated at the start of his vacation and revoked at the end of his vacation).
2. A secretary (*Alice*) delegates her task/action of create borrower account to a librarian (*Jane*).
3. A secretary (*Bob*) transfers his role to an administrator (*Sam*) during maintenance day. In case of a transfer delegation, the delegator temporarily loses his/her rights during the time of delegation.
4. The role administrator is not delegable.
5. The permission of deleting borrower account is not delegable.
6. The director can delegate, on behalf of a secretary, the secretary's role (or some his/her permitted actions) to a librarian (e.g. during the secretary's absence).

7. If a librarian empowered in role *secretary* by delegation is no longer able to perform this task, then he/she can delegate, again, this role to another librarian.
8. The secretary empowered in role *director* by delegation is not allowed to delegate/-transfer, again, this role to another secretary.
9. A secretary is allowed to delegate his/her role to a librarian only and to one librarian at a given time.
10. A secretary is allowed to delegate his/her task of book delivery to a librarian only and scheduled on every Monday.
11. *Bill* can delegate his role and permitted actions only to *Bob*
12. *Bob* is not allowed to delegate his role.
13. *Alice* is not allowed to delegate her permitted action of book delivery.
14. Users can always revoke their own delegations.
15. The director can revoke users from their delegated roles.
16. A secretary can revoke librarians empowered in *secretary* role by delegation, even if he/she is not the creator of this delegation (e.g. the creator is the director or another librarian).

This running example shows the two levels of delegation rules as defined in the previous section: user-level (rules defined by a user: e.g. situations 1, 2, 3) and master-level (rules defined by a security officer: e.g. 4, 5, 6). Obviously, delegation rules at user-level have to conform to rules at master-level. For example, the security officer can define that users of role *director* are able to delegate on behalf of users of role *secretary*. Then at user-level, *Bill* (director) can create a delegation rule to delegate, on behalf of *Alice*, her role (secretary) to *Jane* (librarian).

5.4 Model-Driven Adaptive Delegation

5.4.1 Overview of Our Approach

In our approach, as noted in Section 5.2, delegation is considered as a "meta-level" mechanism which impacts the existing access control policies, like an aspect can impact a base program. We claim that to handle advanced delegation rules, an ideal solution is to logically separate the delegation rules from the access control policy, each being specified in isolation, and then compose/weave them together to obtain a new access control policy (called active security policy) reflecting the delegation-driven policy (Figure 5.3). We present our metamodel (DSML) for specifying delegation based on RBAC in Section 5.4.2.

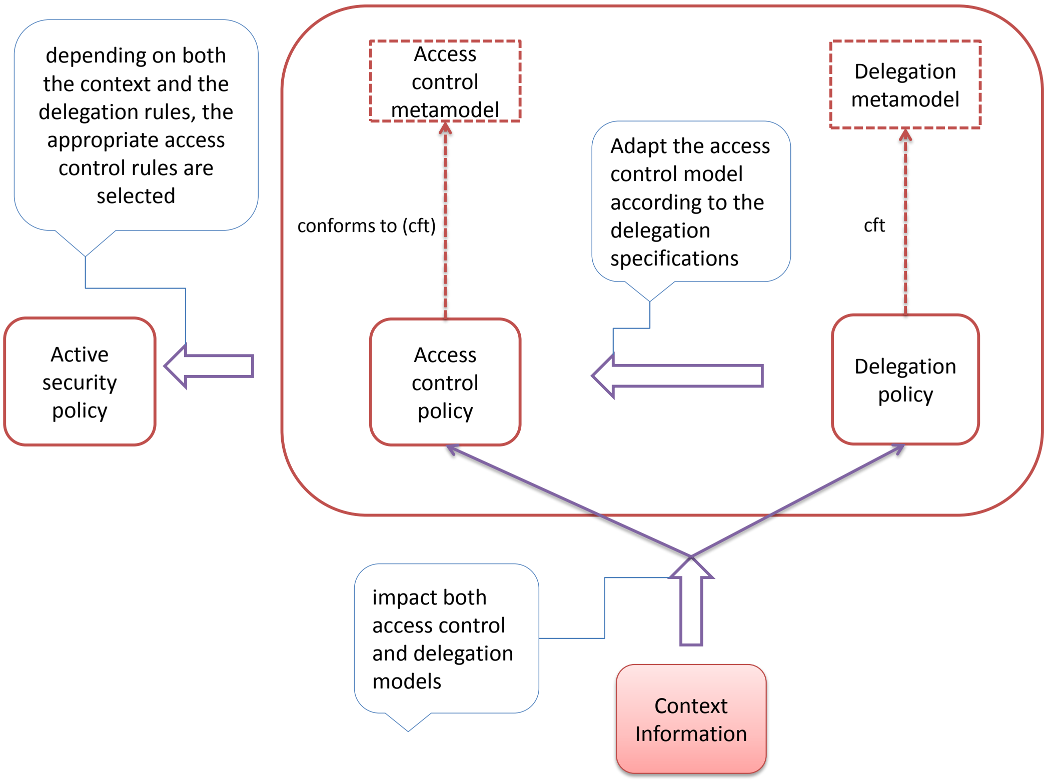

depending on both the context and the delegation rules, the appropriate access control rules are selected
Access control metamodel
Delegation metamodel
conforms to (cft)
Adapt the access control model according to the delegation specifications
cft
Active security policy
Access control policy
Delegation policy
impact both access control and delegation models
Context Information

FIGURE 5.3: Delegation impacting Access Control

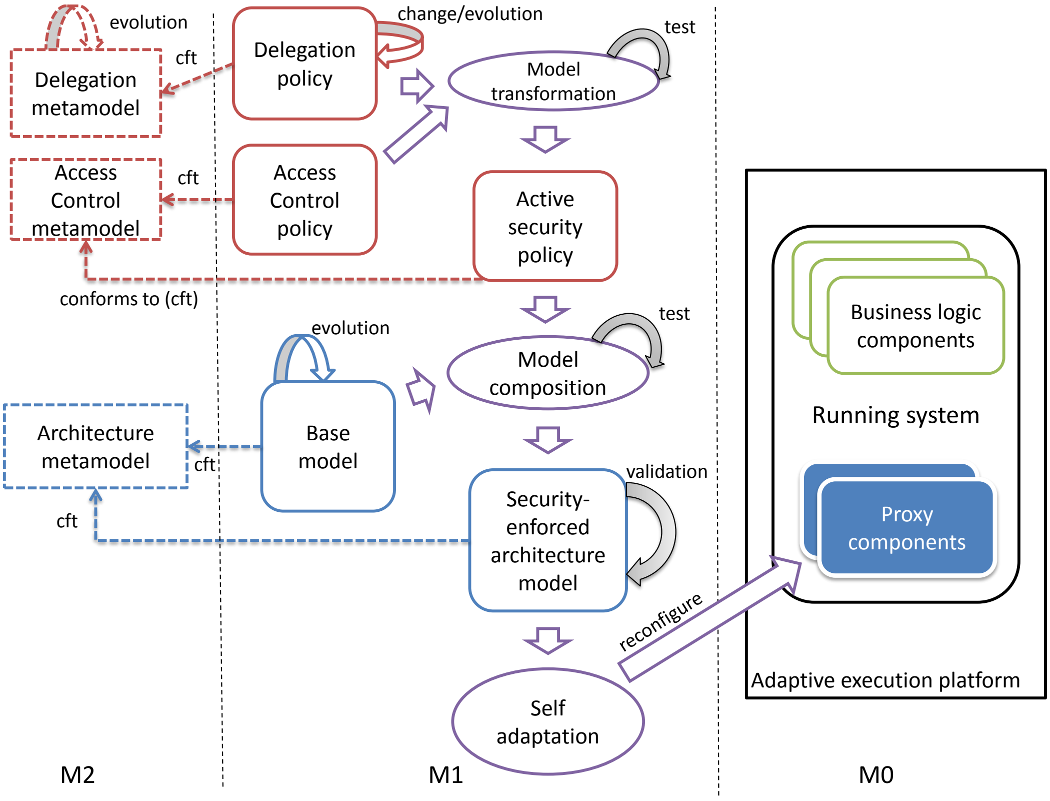


FIGURE 5.4: Overview of our approach

The separation of concerns is not only between delegation and access control, but also between the security policy and the business logic of the system. Figure 5.4 presents a wider view of the overall approach. In order to enforce a security policy for the system, the core business architecture model of the system is composed with the active security policy previously obtained. The architecture model is expressed in another DSML, called architecture metamodel (an architecture modelling language described in [167]). The idea is to reflect security policy into the system at the architecture level. Section 5.4.3 defines transformation rules to show how security concepts are mapped into architectural concepts.

The security-enforced architecture model obtained above is a pure architecture model which by itself reflects how the security policy is enforced in the system. Our model-driven framework to reflect security policy at the architecture model is *generic*, meaning that from the security-enforced architecture model of the system, it is possible to enforce security policy for running systems on different execution platforms. In Section 5.5, we show how our approach is applied for two different adaptive execution platforms, i.e. OSGi [222] and Kevoree [80] [1]. It is important to note that the security-enforced architecture model is not used for generating the whole system but only the proxy components. These proxy components can be adapted and integrated with the running system at runtime to physically enforce the security policy. The adaptation and integration can be done by leveraging the runtime adaptation mechanisms provided by modern adaptive execution middleware platforms. The approach of possibly generating proxy components overcomes some main limitations of [167]. Section 5.4.4 is dedicated to discuss our strategy for adaptation and evolution of the secure systems.

5.4.2 Delegation Metamodel

Our metamodel, displayed in Figure 5.5, defines the conceptual elements and their relationships that can be used to specify access control and delegation policies which are defined in Section 5.2. Because the delegation mechanism is based on RBAC, we first explain the main conceptual elements of role-based access control. Then, we show how our conceptual elements of delegation, based on the RBAC conceptual elements, can be used to specify various delegation features which are defined in Section 5.2.

As shown in Figure 5.5, the root element of our metamodel is the *Policy*. It contains *Users*, *Roles*, *Resources*, *Rules*, and *Contexts*. Each user has one role. A security officer can specify all the roles in the system, e.g. *admin*, *director*, etc., via the *Role* element. In order to specify an access control policy, the security officer should have defined in

[1] www.kevoree.org, last access October 2013

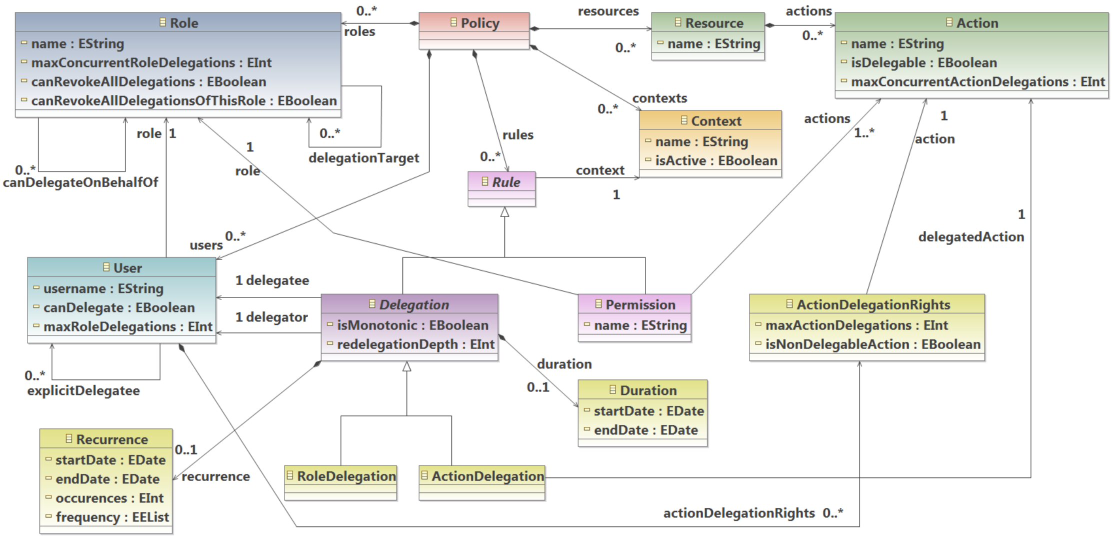


FIGURE 5.5: The Delegation metamodel

advance the *resources* that must be protected from unauthorised access. Each resource contains some *actions* which are only accessible to authorised users. These protections are defined in rules: *permission rules* and *delegation rules*. Permission rules are used to specify which actions are accessible to users based on their *roles*. That means, without delegation rules or user-specific access control rules, every user is able to access the actions associated with his/her role only. Delegation rules are used to specify which actions are accessible to users by delegation. There are two basic types of delegation:

- **Role delegation**: When users empowered in role(s) delegated by other user(s), they are allowed to access not only actions associated with their roles but also actions associated with the delegated role(s).

- **Action delegation**: Instead of delegating their roles, users may want to delegate only some specific actions associated with their roles.

Another important aspect of our access control and delegation framework is the notion of *context* which has been introduced in Section 5.2.2.1. It can be seen from our metamodel that every permission/delegation rule is associated with a context. A rule is only active within its context. The concept of context actually provides our model with high flexibility. Security policies can be easily adapted according to different contexts.

The full metamodel for specifying delegation is displayed in Figure 5.5. It depicts the features that are supported by our delegation framework. All delegation management features are developed based on two basic types of delegation mentioned above. In the following, we show how the delegation features can be specified, w.r.t. our metamodel. In other words, this is how the formal definitions in Section 5.2 are actually implemented.

- **Temporary delegation**: This is one of the most common types of delegation used by users. It describes when the delegation *starts* to be active and when it *ends*. The delegator can specify that the delegated role/action is authorised only during a given time interval, e.g. situation 1 of the running example in Section 5.3. Actually, this can be specified using the recurrence of delegation described below, but we want to define it separately because of its common use.

- **Monotonicity (Transfer of role or permissions)**: A property *isMonotonic* can be used to specify if a delegation is monotonic or non-monotonic. The former (*isMonotonic* = *true*) specifies that the delegated access right is available to both the delegator and delegatee after enforcing this delegation. As defined in Section 5.2, this delegation is called a grant delegation. The latter (*isMonotonic* = *false*) means the delegated role/action is transferred to the delegatee, and the delegator temporarily loses his rights while delegating, e.g. situation 3. In this case, the delegation is called a transfer delegation.

- **Recurrence**: It refers to the repetition of the delegation. A user may want to delegate his role to someone else for instance every week on Monday. Recurrence defines how the delegation is repeated over time. It is similar to what is implemented in calendar system and more precisely the icalendar standard (RFC2445[2]). It has several properties; the *startDate* and *endDate* are the starting and ending dates of the recurrence. In addition, the *startDate* defines the first occurrence of the delegation. The *frequency* indicates one of the three predefined types of frequency, daily, weekly or monthly. The *occurrences* is the number of times to repeat the delegation. If the *occurrences* is for instance equals to 2 it means that it should only be repeated twice even when the *endDate* is not reached. An example of this delegation is situation 10 of the running example.

- **Delegable roles/actions**: These kinds of delegation define which roles or actions can be delegated and how (master-level). A policy officer can specify that a role can only be delegated/transferred to specific role(s), e.g. situation 9. If no *delegationTarget* is defined for a role, this role cannot be delegated/transferred, e.g. situation 4. If a role or action (*isDelegable* = *false*) is not delegable, it should never be included in a delegation rule. Moreover, a role can also be delegated by a user not having this role but his/her own role is specified as can delegate on behalf of a user in this role (*canDelegateOnBehalfOf* = *true*), e.g. situation 6.

[2] http://www.rfc-editor.org/info/rfc2445

- **Multiple delegations**: It should be possible to define the max number of concurrent delegations in which the same role or action can be delegated at a given time (master-level delegation rule). The properties *maxConcurrentRoleDelegations* and *maxConcurrentActionDelegations* define how many concurrent delegations of the same role/action can be granted, e.g. situation 9. Moreover, it is possible to define for each specific user a specific maximum number of concurrent delegations of the same role/action: *maxRoleDelegations* and *maxActionDelegations*.

- **User specific delegation rights**: All user-specific elements are used to define more strict rules for a specific user rather for his/her role. There are other user-specific delegations than *maxRoleDelegations* and *maxActionDelegations*. It is possible to define that a specific user is allowed to delegate his role/permitted action(s) or not (*canDelegate* = *true* or *false*), e.g. situation 12. The property *isNonDelegableAction* specifies an action that a specific user cannot delegate, e.g. situation 13. Moreover, the security officer can define to which explicit user(s) only (*explicitDelegatee*) a user can delegate/transfer his role to, e.g. situation 11.

- **Multi-step delegation**: It provides flexibility in authority management, e.g. situations 7, 8. The property *redelegationDepth* is used to define whether or not the role/action of a delegation can be delegated again. When a creator creates a new delegation, he/she can specify how many times the delegated role/action can be re-delegated. If the *redelegationDepth* = 0, it means that the role/action cannot be delegated anymore, e.g. situation 8. If the *redelegationDepth* > 0, that means the role/action can be delegated again and each time it is re-delegated, the *redelegationDepth* is decreased by 1.

- **Revocations**: All users can revoke their own delegations, e.g. situation 14. Security officer may set *canRevokeAllDelegations* = *true* for a role with a super revocation power in such a way that a user empowered in this role can revoke all delegations, e.g. situation 15. Moreover, a role can also be defined such that every user empowered in this role can revoke any delegation from this role (*canRevokeAllDelegationsOfThisRole* = *true*), even he/she is not the delegator of the delegation, e.g. situation 16.

Moreover, each possible instance of the security policy has to satisfy all necessary validation condition expressed as OCL invariants. For example, we can make sure that no delegation is out of target, meaning that delegatee's role has to be a delegation target of delegator's role:

context Delegation **inv** NoDelegationOutOfTarget: self.delegator.role.delegationTarget −>exists (t | t = self.delegatee.role)

Or to check that for every user, the number of concurrent role delegations cannot be over its thresholds:

context User **inv** NoRoleDelegationOverMax: RoleDelegation.allInstances −>select (d | d.delegator = self) −>size() ≤ self.role.maxConcurrentRoleDelegations and RoleDelegation.allInstances −>select (d | d.delegator = self) −>size() ≤ self.maxRoleDelegations

Other examples are to restrict the value of the *redelegationDepth* must not be negative, or *startDate* cannot be later than *endDate*:

context Delegation **inv** NonNegativeDeleDepth: self.redelegationDepth ≥ 0

context Duration **inv** ValidDates: self.startDate ≤ self.endDate

5.4.3 Transformations/Compositions

After specifying a security policy by the DSML described in Section 5.4.2, it is crucial to dynamically enforce this policy into the running system. Transformations play an important role in the dynamic enforcement process. Via model transformations, security models containing delegation rules and access control rules are automatically transformed into component-based architecture models. Note that instances of security models and architecture models are checked before and after model transformations, using predefined OCL constraints.

The model transformation is executed according to a set of transformation rules. The purpose of defining transformation rules is to correctly reflect security policy at the architectural level. Based on transformation rules, security policy is automatically transformed to proxy components, which are then integrated to the business logic components of the system in order to enforce the security rules. The metamodel of component-based architecture can be found in [167] and an instance of it can be seen in Figure 5.8. We first describe the transformation that derives an access control model according to delegation rules (step 1), and then describe another transformation to show how security policy can be reflected at the architecture level (step 2). Moreover, we also show an alternative way of transformation that combines two steps into one.

5.4.3.1 Adapting Role-Based Access Control policy model to reflect delegation (step 1):

Within the security model shown in Figure 5.3, delegation rules are considered as "meta-level" mechanisms that impact the access control rules. The appropriate access control rules and delegation rules are selected depending on the context information and/or the

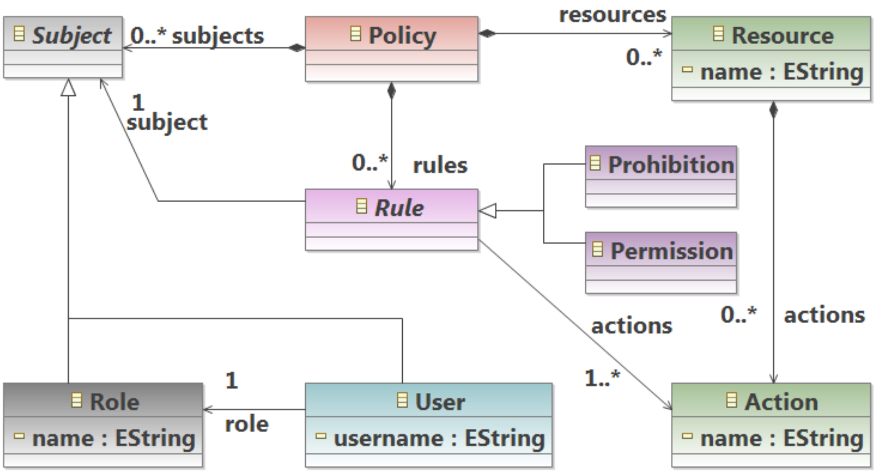


FIGURE 5.6: A pure RBAC metamodel

request of changing security rules coming from the system at runtime. According to the currently active context (e.g. WorkingDays), only *in-context* delegation rules and *in-context* access control rules of the security model (e.g. rules that are defined with context = WorkingDays) are taken into account to derive the active security policy model (Figure 5.3). Theoretically, we could say that delegation rules impact the core RBAC elements in the security model in order to derive a pure RBAC model (without any delegation and context elements) which conforms to a "pure" metamodel of RBAC (Figure 5.6). Delegation elements of a security policy model are transformed as follows:

A.1: Each **action delegation** is transformed into a new permission rule. The *subject* of the permission is *user* (delegatee) object. The set of *actions* of the permission contains the delegated action.

A.2: Each **role delegation** is transformed as follows. First, a set of actions associated to a role is identified from the permissions of this role. Then, each action is transformed into a permission like transforming an **action delegation** described above.

A.3: A **temporary delegation** is only taken into account in the transformation if it is in active duration defined by the **start** and **end** properties. In fact, when its active duration starts the (temporary) action/role delegation is transformed into permission rule(s) as described above. When its active duration ends the temporary delegation is removed from the policy model.

A.4: If an action delegation is of type transfer delegation (**monotonic**), then it is transformed into a permission rule and a prohibition rule. The *subject* of the permission is the *user*-delegatee object. The set of *actions* of the permission contains the delegated action. The *subject* of the prohibition is the *user*-delegator object. The set of *actions* of the prohibition contains the delegated action.

A.5: If a role delegation is of type transfer delegation, then it is also transformed into a permission rule and a prohibition rule. The *subject* of the permission is the *user*-delegatee object. The set of *actions* of the permission contains the delegated actions. The delegated actions here are the actions associated with this role. The *subject* of the prohibition is the *user*-delegator object. The set of *actions* of the prohibition also contains the delegated actions.

A.6: If a delegation rule is defined with a **recurrence**, based on the values set to the recurrence, the delegation rule is only taken into account in the transformation within its *fromDate* and *untilDate*, repeated by *frequency* and limited by *occurrences*. In other words, only active (during recurrence) delegation rules are transformed.

A.7: (User-specific) If a user is associated with any **non-delegable action**, the action delegation containing this action and this user (as delegator) is not transformed into a permission rule. Similarly, if a user is specified as he/she **cannot delegate** his/her role/action, no role/action delegation involving this user is transformed.

A.8: (Role/action-specific) Any delegation rule with a **non-delegable** role/action will not be transformed. In fact, a delegation rule is only transformed if it satisfies (at least) both user-specific and role/action-specific requirements.

A.9: Only a role delegation to a user (delegatee) whose role is in the set of **delegationTarget** will be considered in the transformation.

A.10: Before any delegation is taken into account in the transformation, it has to satisfy the requirements of **max concurrent action/role delegations**. Note that the user-specific values have higher priorities than the role-specific values.

A.11: A delegation is only transformed if its **redelegationDepth** > 0. Whenever a user empowered in a role/an action by delegation re-delegates this role/action, the newly created delegation is assigned a **redelegationDepth** = the previous **redelegationDepth** - 1.

After transforming all delegation rules, we obtain a pure RBAC model which reflects both the delegation model and access control model. This pure RBAC model is then transformed into a security-enforced architecture model as described next.

5.4.3.2 Transformation of Security Policy to Component-based Architecture (step 2):

The transformation rules are defined below. The goal is to transform every security policy model (pure RBAC model obtained in step 1) which conforms to the metamodel shown in Figure 5.6 to a component-based architecture model which conforms to the metamodel described in [167]. However, both the security policy model and the *base model* provided by a system designer are used as inputs for the model transformation/-composition. Via a graphical editor, the security designer must define in advance how

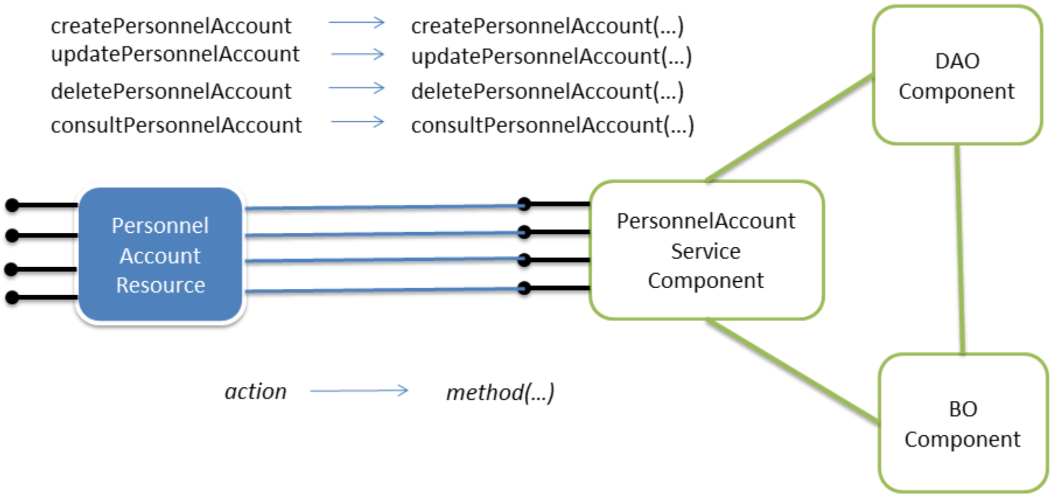

createPersonnelAccount → createPersonnelAccount(...)
updatePersonnelAccount → updatePersonnelAccount(...)
deletePersonnelAccount → deletePersonnelAccount(...)
consultPersonnelAccount → consultPersonnelAccount(...)
Personnel Account Resource
PersonnelAccount Service Component
DAO Component
BO Component
action → method(...)

FIGURE 5.7: Mapping Resources to Business Logic Components

the resource elements in the policy model are related to the business components in the *base model*. Figure 5.7 shows how each action in the policy can be mapped to the Java method in the business logic.

Because the *base model* already conforms to the architecture metamodel, we now only focus on transforming the security policy model into the security-reflected architecture model. As we know, this transformation/composition process will also weave the security-reflected elements into the base model in order to obtain the security-enforced architecture model.

The core elements of RBAC like *resource*, *role*, and *user* are transformed following these transformation rules. All the transformation rules make sure that the security policy is reflected at the architectural level.
R-A.1: Each *resource* is transformed into a component *instance*, called a *resource* proxy component. According to the relationship between the resource elements in the policy model and the business components in the *base model*, each *resource* proxy component is connected to a set of business components via bindings. To be more specific, each action of a resource element is linked to an operation of a business component (Figure 5.7). By connecting to business components, a *resource* proxy component provides and requires all the services (actions) offered by the resource.
R-A.2: Each *role* is also transformed into a *role* proxy component. According to the granted accesses (permission rules associated with this role) to the services provided by the resources, the corresponding *role* proxy component is connected to some *resource* proxy component(s) (Figure 5.8). A *role* proxy component is connected to a *resource* proxy component by transforming granted accesses into ports and bindings. Each (active) access granted to a role is transformed into a pair of ports: a client port associated

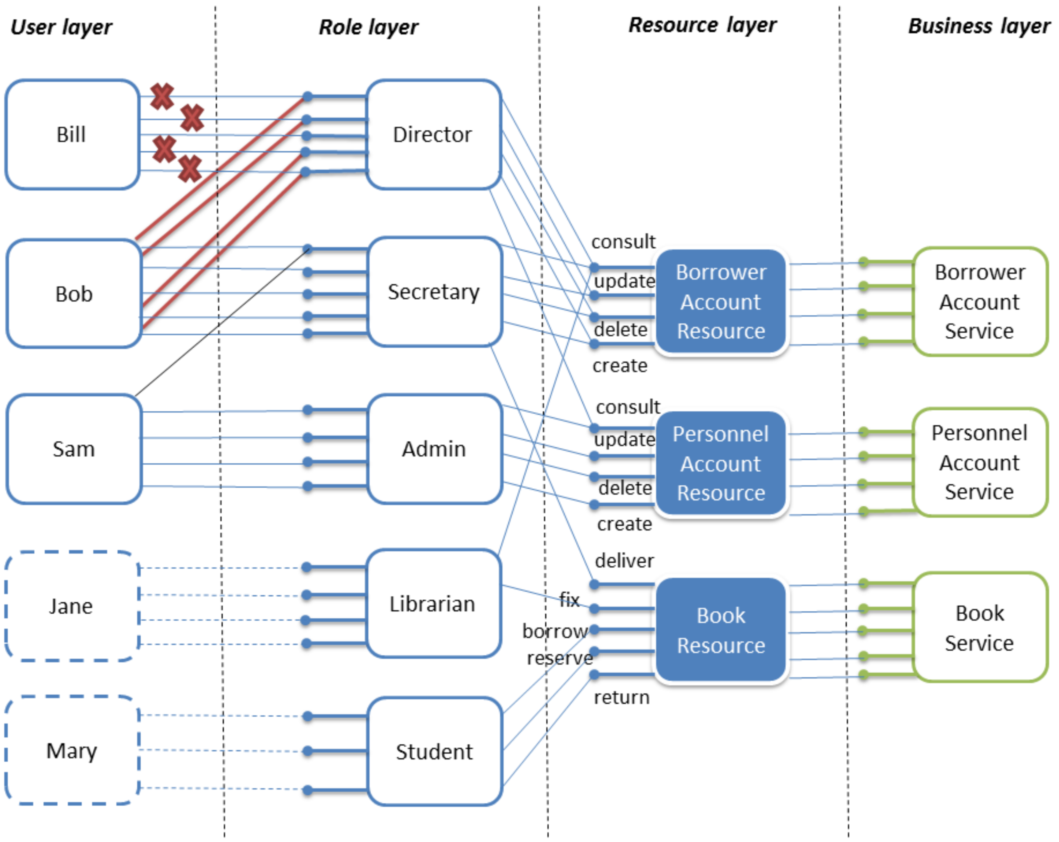


Figure 5.8: Architecture reflecting security policy before and after adding a delegation rule (bold lines)

with the *role* proxy component, a server port associated with the *resource* proxy component, and a binding linking these ports.

R-A.3: Each *user* element defined in the policy model is also transformed into a *user* proxy component. Because each user must have one role, each *user* proxy component is connected to the corresponding *role* proxy component. However, each user may have access to actions associated to not only his/her role but also to actions associated to other roles by delegation. Thus, each *user* proxy component may connect to several *role* proxy components. The connection is established by transforming each access granted to a user into a pair of ports: a client port associated with the *user* proxy component, a server port associated with the corresponding *role* proxy component (providing the access/port), and a binding linking these ports (Figure 5.8). Actually, the granted accesses are calculated not only from *permission* rules but also from *prohibition* rules. Simply, the granted accesses that equal permissions exclude prohibitions.

In our approach, revocation of a delegation simply consists in deleting the corresponding delegation rule. In this way, the revocation is reflected at the architectural level and physically enforced in the running system. Moreover, both the delegator and delegatee elements will be removed if these users are not involved in any delegation rules. As described above, user elements are transformed into proxy components. However, it is important to stress that only users involved in delegation rules (e.g. Bill, Bob and Sam in Figure 5.8) are created in the security policy model and transformed into proxy components. Users who are not involved in any delegation rules (e.g. Jane and Mary in Figure 5.8), are manipulated as session objects which directly access the services offered by the corresponding role proxy components.

Two steps described above are two separate model transformations that are mainly used to explain how delegation can be considered as a "meta-level" mechanism for administrating access rights. The first model transformation is to transform a delegation-driven security model into a pure RBAC model. The second model transformation is to transform the RBAC model into an architecture model. In fact, these two steps could be done in only one model transformation that directly transforms the delegations, the access control policy and the business logic model into an architecture model reflecting the security policy. However, this alternative way (described in the following) has the disadvantage of losing the intermediate security model (the active security policy) that could be useful for traceability purpose.

5.4.3.3 An alternative way using only one transformation:

In this approach, we have to define different transformation rules to transform directly every security policy model which conforms to the metamodel, shown in Figure 5.5, to a component-based architecture model which conforms to the architecture metamodel described in [167]. Core elements of RBAC like *resources*, *roles*, and *users* are transformed following these transformation rules:
R-B.1: Each *resource* is transformed into a component *instance*, called a *resource* proxy component (already presented).
R-B.2: Each *role* also is transformed into a *role* proxy component (already presented). The only difference here is that the *context* has to be taken into account (in the step 2 of transformation mentioned earlier, no *context* existed because *context* was already dealt with in the step 1). Because every permission is associated with a *context*, we only transform permissions with the *context* that is active at the moment.
R-B.3: Each *user* element defined in the policy model is also transformed into a *user* proxy component. However, the connection (via bindings) from a *user* proxy component to the *role* proxy component(s) is not only depended on the user's role but also delegation rules that the corresponding user involved in. The transformation of delegation rules is presented below.

All the transformation rules above make sure that access control rules are reflected at the architecture level. However, the delegation rules will impact this transformation process in order to derive the security-enforced architecture model reflecting both access control and delegation policy. Delegation elements of a policy model are transformed as follows:

R-B.4: Each action involved in an **action delegation** is transformed into a pair of ports and a binding. A client port (representing the required action) is associated with the user (*delegatee*) proxy component. The binding links the client port to the corresponding server port (representing the same action provided) that associated with the *role* proxy component reflecting the role of the *delegator*.

R-B.5: Each **role delegation** is transformed in a similar way as **action delegation**. First, a set of actions associated to a role can be identified from the permissions of this role. Then, each action in the set is transformed into a pair of ports and a binding as transforming an action delegation.

R-B.6: A **temporary delegation** is only transformed into bindings if it is still in active duration defined by **start** and **end** properties.

R-B.7: If a delegation is of type **transfer delegation**, then both user elements (delegator and delegatee) are transformed into delegator and delegatee proxy components as described above. The delegator proxy component is not connected to the corresponding role proxy component because he/she already transferred his/her access rights to the delegatee. Figure 5.8 shows a change in the architecture when *Bill* transfers his role to *Bob*.

R-B.8: If a delegation is defined with a **recurrence**, based on the values set to recurrence, the delegation rule is only active during the recurrence (similar to A.6).

R-B.9: If a user is associated with any **non-delegable action**, the delegation of this action is not taken into account while doing the transformation. Similarly, if a user is specified as he/she **can not delegate** his/her role/action, no delegation requested by this user will be transformed.

R-B.10: Only a role delegation to a user (delegatee) whose role is in the set of **delegationTarget** will be consider in the transformation.

R-B.11: Before any delegation is taken into account in the transformation, it has to satisfy the requirements of **max concurrent action/role delegations**. Note that the user-specific values have higher priorities than the role-specific values.

R-B.12: A delegation is only transformed if its **redelegationDepth** > 0. Whenever a user empowered in a role/an action by delegation re-delegates this role/action, the newly created delegation is assigned a **redelegationDepth** $=$ the previous **redelegationDepth** - 1.

By taking into account delegation rules while transforming access control rules of policy model into security-enforced architecture model, both delegation and access control rules are reflected at the architecture level.

5.4.4 Adaptation and Evolution Strategies

The model transformation/composition presented in Section 5.4.3 ensures that the security policies are correctly and automatically reflected in an architectural model of the system. The key steps to support delegation (i.e. specifications and transformations) are already presented in Sections 5.4.2 and 5.4.3. The last step consists in a physical enforcement of the security policy by means of a dynamic adaptation of the running system. In this section, our adaptation and evolution strategies are discussed.

5.4.4.1 Adaptation

The input for the adaptation process is a newly created security-enforced architecture model (Figure 5.9). First, this new architecture model is validated using invariant checking [165]. This valid architectural model actually represents the new system state the runtime must reach to enforce the new security policy of the system. According to the classical MAPE control loop of self-adaptive applications, our reasoning process performs a comparison (using EMFCommpare) between the new architecture model (target configuration) and the current architecture model (kept synchronised with the running system) [166]. This process triggers a code generation/compilation process, and also generates a safe sequence of reconfiguration commands [165]. Actually, the code generation/compilation process is only triggered if there are new proxy components, e.g. new user proxy components involved in delegation, that need to be introduced into the running system. The dynamic adaptation of the running system is possible thanks to modern adaptive execution platforms like OSGi [222] or Fractal [53], and most recently Kevoree [80], which provide low-level APIs to reconfigure a system at runtime. The running system is then reconfigured by executing the safe sequence of commands, compliant to the platform API, issued by the reasoning process. In an optimised model@runtime platform like Kevoree, all we need to do is to provide the reconfiguration script (Kevoree script) for the platform. The reasoning process is taken care of by the platform. In fact, the generation/compilation phase if needed could be time consuming. However, this phase has no impact on the running system, which remains stable until being adapted by executing the reconfiguration script. Thus, the actual adaptation phase lasts for only several milliseconds.

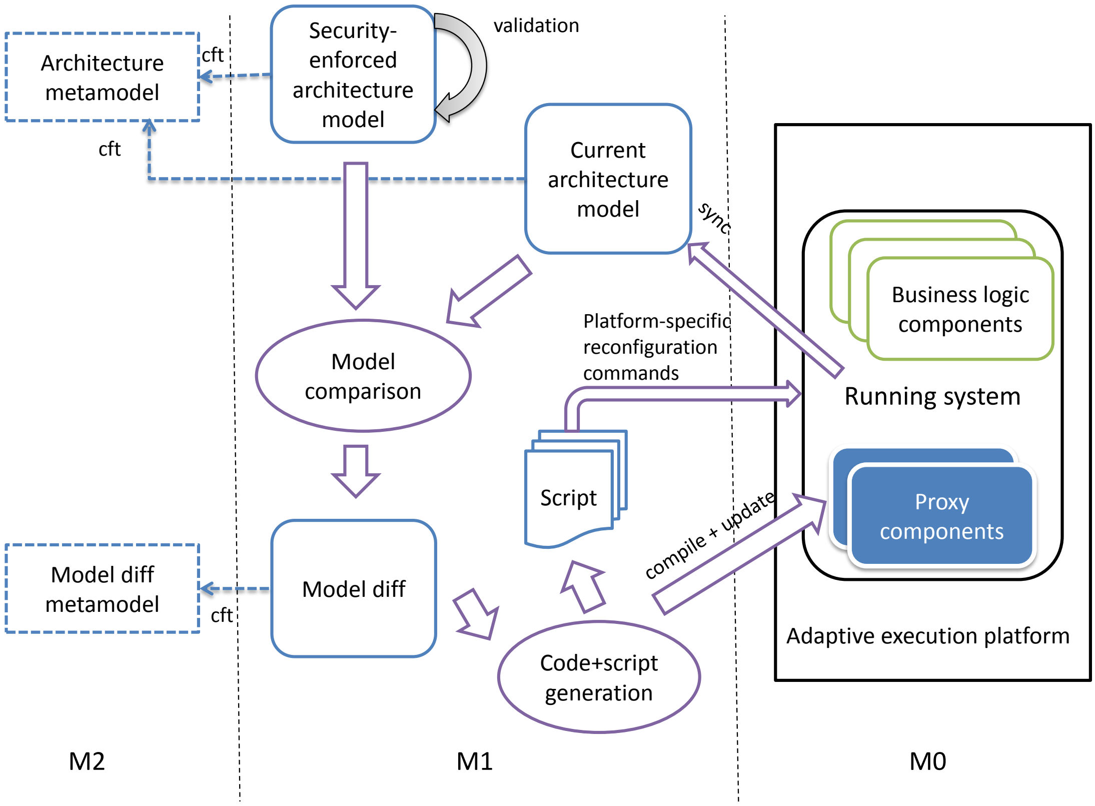


Figure 5.9: Overview of our adaptation strategy

In [167], the adaptation is entirely based on executing platform-specific reconfiguration scripts specifying which components have to be stopped, which components and/or bindings should be added and/or removed. This results in several limitations regarding delegation mechanisms:

L.1: Using only reconfiguration scripts implies to create all the potentially needed ports (used for bindings between *user* proxy components) beforehand. But all the combinations of users, roles, resources, actions could lead to a combinatorial explosion and make it infeasible for implementation.

L.2: In [167], the delegation between users are reflected using bindings connecting one *user* proxy component to another. But this approach is not suitable for supporting complex delegation features. For example, a transfer delegation will be reflected by adding bindings between the *delegator* and *delegatee* but removing bindings between *delegator* and the corresponding *role* proxy component. Consequently both *delegator* and *delegatee* cannot access the resource, which does not correctly reflect a transfer delegation.

L.1 can be solved by the automatic re-generation of proxy components and bindings between them according to changes in the architectural model. Moreover, as mentioned in Section 5.4.3.2, only users involved in a delegation are transformed into *user* proxy components with necessary ports and bindings. In this way, only required ports and bindings are created dynamically. **L.2** is solved by our model transformation approach. All complex delegation features are considered as “meta-level” mechanisms that impact access control rules. In this way, a transfer delegation will be reflected by adding bindings between the *delegatee* and the corresponding delegated *role* proxy component, but removing bindings between *delegator* and the corresponding *role* proxy component.

Our adaptation strategy could take more time than simply running a reconfiguration script because of the generation and compilation time of newly generated proxy components. But the process of generating and compiling new proxy components does not in fact harm the performance because each proxy component is very light-weight and only necessary proxy components are generated (see Section 5.5). Moreover, for each specific security policy, it is possible to think in advance and prepare as many proxy component types as possible. This strategy could make the generation/compilation phase unnecessary for most of the cases, except some major evolution of the business logic and/or the security policy.

5.4.4.2 Evolution

In [167], the evolution of the security policy is not totally dealt with. It is possible to run a reconfiguration script to reflect changes like adding, removing and updating rules. But adding a new user, role or resource requires the generation and compilation of new proxy components, which is impossible using only reconfiguration scripts. Thus, our strategy of automatically generating and compiling proxy components (see Section 5.5) is more practical w.r.t. evolution.

Another important aspect of evolution relates to the addition, removal or update of resources and actions in the business logic. The base architecture model can be updated with changes in the business logic, e.g. when a new resource is added. On the other side, security officers can manually update the mappings (Figure 5.7) following changes of resources/actions in the base architecture model. By composing the security model with the base architecture model as described earlier, the security policy is evolved together with the business logic of the system.

5.5 Implementation and Evaluation

This section shows how the steps described in Figure 5.4 have been implemented. In order to prove that our approach is generic, we target two different adaptive execution platforms: OSGi (more detail in Section C.1) and Kevoree (more detail in Section C.2).

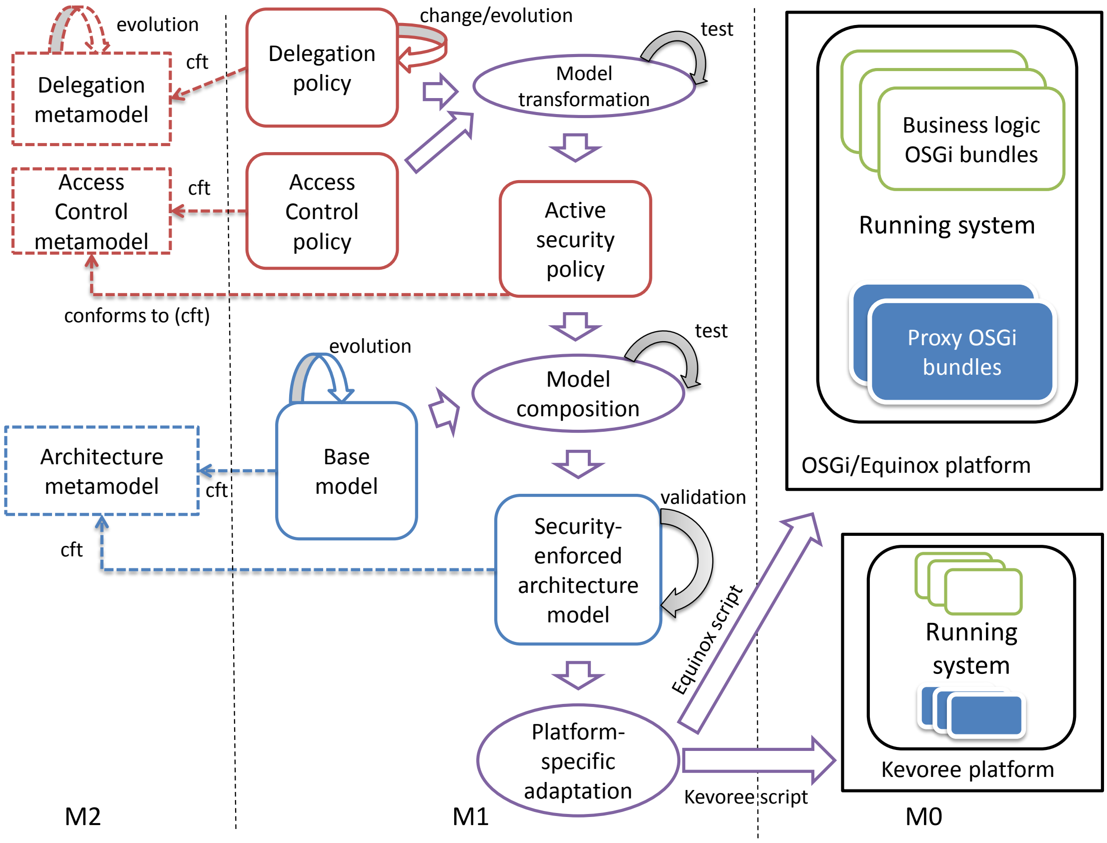


FIGURE 5.10: OSGi and Kevoree as adaptive execution platforms

Figure 5.10 shows that our metamodels and model-to-model transformation/composition are generic, i.e. independent of execution platforms. Only the adaptation process (e.g. the reconfiguration script) and the running system are platform-specific. The description of three case studies used in our experiments are given in Section 5.5.1. We evaluate our proof-of-concept implementations and discuss the results in Section 5.5.2. The business logic of these case studies are the same for the OSGi and Kevoree adaptive execution platforms.

5.5.1 Case Studies

To evaluate the feasibility of our approach, we have applied it on three different Java-based case studies, which have also been used in our previous research work on access control testing [170]:

1) LMS: as described in our running example.

2) VMS[3]: The Virtual Meeting System offers simplified web conference services. The virtual meeting server allows the organisation of work meetings on a distributed platform. When connected to the server, a user can enter (exit) a meeting, ask to speak, eventually

[3] For more information about VMS (server side), please refer to http://franck.fleurey.free.fr/VirtualMeeting.

speak, or plan new meetings. There are three resources (Meeting, Personnel Account, User Account) and six roles (Administrator, Webmaster, Owner, Moderator, Attendee, and Non-attendee) defined for this system with many access control rules, and delegation situations between the users of each role.

3) ASMS: The Auction Sale Management System allows users to buy and sell products online. Each user in the system has a profile including some personal information. Users wanting to sell a product (sellers) are able to start a new auction by submitting a description of the product, the starting and ending date of the auction. There are five resources (Sale, Bid, Comment, Personnel Account, User Account) and five roles (Administrator, Moderator, Seller, Senior Buyer, and Junior Buyer) defined for this system, also with many access control rules, and delegation situations between users of each role.

Table 5.1: Size of each system in terms of source code

	# Classes	**# Methods**	**# LOC**
LMS	62	335	3204
VMS	134	581	6077
ASMS	122	797	10703

Table 5.2: Security rules defined for each system

	# AC rules	**# Delegations**	**Total**
LMS	23	4	27
VMS	36	8	44
ASMS	89	8	97

Table 5.1 provides some information about the size of these three systems (the number of classes, methods and lines of code). In terms of security policies, Table 5.2 shows the number of access control (AC) rules and delegation rules defined for each system, used in our experiments.

All these systems are designed as component-based systems. The business components of each system contain the business logic, e.g. Book Service component, Personnel Account component, Meeting, Sale, Authenticate component, Data Access Object components, etc. To enable dynamic security enforcement for a system, the resources (components that have to be controlled) are specified in the base model, and mapped to the resources of the security policies. Our metamodels are applicable for different systems without any modification or adaptation. The structure of delegation and access control policies for all case studies is the same, only roles, users, resources, actions are specific to each case study. The proxy components are automatically generated and synchronised with the security policy model via model transformations and reconfiguration at runtime. The

TABLE 5.3: Performance of weaving Security Policies using Kermeta and ATL

	# Rules	**Kermeta 1.4.1**	**Kermeta 2.0.6**	**ATL 3.2.1**
LMS	27	4s	1.836s	0.048s
VMS	44	7s	2.161s	0.055s
ASMS	97	18s	2.834s	0.140s

model-to-model transformation and model-to-text transformation (code generation) can be implemented correspondingly using transformation engines like Kermeta [177] (or ATL[4]), and Xpand [130].

5.5.2 Evaluation and Discussion

There are two kinds of response time we would like to measure in our case studies: the authorisation mechanism and the dynamic adaptation according to changing security policies. The experiments were performed on Intel Core i7 CPU 2.20 GHz with 2.91 GB usable random-access memory running on Windows 7. The number of security rules defined for each system in our experiments is indicated in Table 5.2. Because all our access control and delegation rules are transformed to proxy components reflecting our security policy, response times to an access request only depends on method calls between these proxy components and business components (Figure 5.8). Unsurprisingly, response time to every resource access is a constant, only about 1 millisecond, because the access is already possible or not by construction. In other words, our 3-layered architecture reflecting security policy enables very quick response, independently from the number of access control and delegation rules.

For experimenting with performance of adapting the running system, we have implemented the model transformation/composition rules using not only Kermeta but also ATL. Regarding the adaptation process, Table 5.3 shows results of each case study for performing the model transformations of security policies mentioned in Table 5.2, using Kermeta 1.4.1, Kermeta 2.0.6, and ATL 3.2.1 correspondingly. Note that these model-to-model transformations are generic, platform-independent w.r.t the implementation platform of the running system. Thus, the same model-to-model transformations are used in both cases of implementation platform, i.e. OSGi and Kevoree. At first, we used Kermeta 1.4.1 to implement our model transformations. However, the performance of using Kermeta 1.4.1 shown in Table 3 was disappointing. It took more than 18 seconds to weave 97 security rules in case of the ASMS. To know if this performance problem is inherently linked to our approach or simply linked to the use of Kermeta 1.4.1, we decided to also implement our model transformations using ATL 3.2.1. Our experiments

[4] http://www.eclipse.org/atl/

show that the implementation using ATL 3.2.1 is much more efficient. We can conclude that the initial performance issue was due to Kermeta 1.4.1. Then, we have tried to use Kermeta 2.0.6 that is the latest version of Kermeta at this moment, compiled to byte code, which means much better performances. As can be seen from Table 5.3, the results of using Kermeta 2.0.6 are much better compared to using Kermeta 1.4.1.

Note that the transformation, code generation and compilation are performed “offline” meaning that the running system is not yet adapted. The actual adaptation happens when the newly compiled proxy components are integrated into the running system to replace the current proxy components. This actual adaptation process takes only some milliseconds by using the low-level APIs to reconfigure a system at runtime provided by the modern adaptive execution platforms, i.e. OSGi [222] and Kevoree [80]. Right after the new proxy components are up and running, the new security policy is really enforced in the running system.

More details on the implementations on different platforms OSGi and Kevoree can be found in Appendix C.

5.6 Related Work

There is substantial work related to delegation as an extension of existing access control models. Most researchers focused on proposing models solely relying on the RBAC formalism [205], which is not expressive enough to deal with all delegation requirements. Therefore, some other researchers extended the RBAC model by adding new components, such as new types of roles, permissions and relationships [22, 242, 5, 61, 178]. In [33], the authors proposed yet another delegation approach for role-based access control (more precisely for ORBAC model) which is more flexible and comprehensive. However, no related work has provided a model-driven approach for both specifying and dynamically enforcing access control policies with various delegation requirements. Compared to [167], we extend the model-based dynamic adaptation approach of [167] with some key improvements. More specifically, we propose a new DSML for delegation management, but also new composition rules to weave delegation in a RBAC-based access control policy. In addition, we present a new way (by generating proxy) to implement the adaptation of the security-enforced architecture of the system. Indeed, we provide an extensive support for delegation as well as co-evolution of security policy and security-critical system. That means our approach makes it possible to deeply modify the security policy (e.g. according to evolution of the security-critical system) and dynamically adapt the running system, which is often infeasible using the other approaches mentioned above.

In addition, several researchers proposed new flexible access control models that may not include delegation, but allow a flexible and easy to update policy. For instance, Bertino *et al.* [34] proposed a new access control model that allows expressing flexible policies that can be easily modified and updated by users to be adapted to specific contexts. The advantage of their model resides in the ability to change the access control rules by granting or revoking the access based on specific exceptions. Their model provides a wide range of interesting features that increase the flexibility of the access control policy. It allows advanced administrative functions for regulating the specification of access controls rules. More importantly, their model supports delegation, enabling users to temporarily grant other users some of their permissions. Furthermore, Bertolissi *et al.* proposed DEBAC [36] a new access control model based on the notion of event that allows the policy to be adapted to distributed and changing environments. Their model is represented as a term rewriting system [23], which allows specifying changing and dynamic access control policies. This enables having a dynamic policy that is easy to change and update.

As far as we know, no previous work tackled the issue of enforcing adaptive delegation. Some previous approaches were proposed to help modelling more general access control formalisms using UML diagrams (focusing on models like RBAC or MAC). RBAC was modelled using a dedicated UML diagram template [127], while Doan *et al.* proposed a methodology [66] to incorporate MAC in UML diagrams during the design process. All these approaches allow access control formalisms to be expressed during the design. They do not provide a specific framework to enable adaptive delegation at runtime. Concerning the approaches related to applying MDE for security, we can cite UMLsec [107], which is an extension of UML that allows security properties to be expressed in UML diagrams. In addition, Lodderstedt *et al.* [143] propose SecureUML which provides a methodology for generating security components from specific models. The approach proposes a security modelling language to define the access control model. The resulting security model is combined with the UML business model in order to automatically produce the access control infrastructure. More precisely, they use the Meta-Object facility to create a new modelling language to define RBAC policies (extended to include constraints on rules). They apply their technique in different examples of distributed system architectures including Enterprise Java Beans and Microsoft Enterprise Services for .NET. Their approach provides a tool for specifying the access control rules along with the model-driven development process and then automatically exporting these rules to generate the access control infrastructure. However, they do not directly support delegation. Delegation rules should be taken into account early and the whole system should be generated again to enforce the new rules. Our approach enables supporting directly the delegation rules and dynamically enforcing them by reconfiguring the system at runtime.

5.7 Testing Delegation Policy Enforcement via Mutation Analysis

In this section, we present a mutation analysis approach for testing delegation policy enforcement described before. Delegation may be viewed as an exception made to an access control policy in which a user gets right to act on behalf of other users. This "meta-level" characteristic together with the complexity of delegation itself make it crucial to ensure the correct enforcement and management of delegation policy in a system via testing. To this end, we adopt mutation analysis for delegation policies. In order to achieve this, a set of mutation operators specially designed for introducing mutants into the key components (features) of delegation is proposed. Our approach consists of analysing the representation of the key components of delegation, based on which we derive the suggested set of mutation operators. These operators can then be used to introduce mutants into delegation policies and thus, enable mutation testing.

5.7.1 Introduction

In the field of access control, delegation is a very complex but important aspect that plays a key role in the administration mechanism [33]. Delegation is a process of delegating access rights from one user (called delegator) to another user (delegatee). The management and enforcement of a delegation policy is crucial because delegation can be considered as administrative mechanism of access control. In a process of delegation, user gets exceptional capabilities to act on behalf of other users. Any discrepancy in delegation can cause a malicious user to get access to the protected resources of the system. It may cause privacy issues in case the data is used without the consent of authorised user. Therefore the enforcement of delegation in the system should be free from any disparity. Testing is a way to get confidence on the correctness of security policies enforcement, including delegation policies enforcement.

Software Testing aims at finding errors by executing tests against the flaws of the system. It is a way to establish confidence on the correctness of the system behaviour and to ensure its quality. Although work is being done using formal verification [4, 94] and static/dynamic program analysis based techniques [142], testing approaches are still in need. As formal verification identifies design flaws but, some of the underlying implementation defects still remain undetected. To this end, some work have been done on testing access control policies [138, 147, 148, 238]. However, none of these works aims at testing delegation policies. This forms the main issue addressed by the present section.

Mutation testing has been applied on XACML policies [56] and OrBAC policies [141]. Additionally, generic policy mutants and Obligation specific mutants have also been proposed in [146] and in [70], respectively. In this lines, this section proposes the use of mutation analysis for testing the behaviour of a system with respect to its delegation policy. This practice provides an effective way to specialise the testing process to the delegation enforcement mechanism.

Delegation policy enforcement has certain testing challenges that makes its testing task interesting and hard. It requires verifying the mapping between different delegation elements such as delegator, delegatee and role or permission. Additionally, since delegation is often context dependent, it is vital to ensure that a specific delegation rule is enforced correctly in its specified context. Moreover, advanced (complex) delegation features (like monotonicity, temporary delegation, multiple delegation, multi-step delegation, etc.) pose further difficulties to the testing process. However, as a key role in the administration mechanism of access control, the more complex delegation features that a system supports, the easier (and more secure) its administration task is. Thus, testing delegation policy enforcement has to take into account various advanced delegation features that this section is dealing with.

Testing of a delegation policy enforcement is the process of ensuring the correct enforcement of its delegation rules in a system. Mutation analysis for delegation policy involves creating mutants of a policy by injecting defects into the delegation rules of the policy. The system implementation is then checked against the enforcement of the mutated policy versions. In view of the request-response nature of a delegation, policy mutants can be recognised as killed or live. In this section, we introduce the issues of testing delegation policy enforcement. Ultimately, we suggest the use of mutation testing for their effective test. Briefly, our main contributions consist of 1) a formal specification of access control and delegation policy supporting advanced delegation features; and 2) a set of mutation operators derived from the process of analysing different aspects of delegation.

The rest of this section is organised as follows. Section 5.7.2 illustrates the aspects of testing a delegation policy via a running example. In Sections 5.7.3 and 5.7.4 the proposed set of mutation operators and a demonstration example of their application are given respectively. Finally Sections 5.7.5 discuses some related work and Section 5.7.6 summaries the main points.

5.7.2 Some Simplified Examples of Delegation for Testing

We reuse the running example about a Library Management System presented in Section 5.3. Some delegation situations are described as follows to illustrate our testing approach. We refer to these delegations as delegation 1, 2 and 3.

1. Bill (has role *Director*) delegates his permission of consulting personal account to Bob (*Secretary*).
2. Alice (*Secretary*) transfers her permission of creating borrower account to Jane (*Librarian*). Alice cannot use this permission while delegating it.
3. Bill (*Director*) delegates his permission of consulting personal account to Bob (*Secretary*) **during his vacation** (the delegation is automatically activated at the start of his vacation and revoked at the end of his vacation).

These delegation situations also will be used further in testing their correct enforcement via mutation analysis.

5.7.3 Mutation Operators

In this section, we define mutation operators for delegations. The operators are defined solely for delegation in an access control policy and we do not consider other aspects of policy such as obligations. Testing of access rights and obligations is done separately in several works [70], [146]. We categorise the operators into two broad categories: basic delegation operator and advanced delegation operator.

Note that we omitted the notion of role in Section 5.2 because we focused on the formalisation of different features of delegation where they can be specified without the definition of role. However, in practice, the notion of role is commonly used for supporting natural abstractions like sets of permissions. Role-Based Access Control (RBAC) introduces a set of role and decomposes the relation AC into user-role assignment $UR \subseteq U \times R$, and role-permission-context assignment $RPC \subseteq R \times P \times C$. Thus, $AC = UR \circ RPC$. For evaluation of operators, we will derive our delegation operators based on RBAC but we consider aspects that are commonly found in most access control and delegation models.

5.7.3.1 Basic Delegation Mutation Operators

Basically delegation are of two types: permission delegation and role delegation.
- Permission delegation: Users can delegate specific permission(s) that are associated

with their own roles.
- Role delegation: Role delegation means a user empowered in some role(s) can delegate his role to other user (s). In this way the delegatee can use permissions of his role and permissions of role that is delegated to him. We introduce the following operators based on the basic types of delegation.

Permission Delegation Operator Permission delegations are the most frequent type of delegations used in access control. Permissions are delegated between two subjects, like in LMS, one such delegation is Alice (secretary) delegated her permission to create borrow account to Jane (librarian). The Permission Delegation Operator (PDM) will mutate the permission delegation rule by replacing the permission being delegated by another permission of the delegator.

Definition 5.12. $PDM(u_1, u_2, p_{1a}, c) : -$
pre $(u_1, u_2, p_{1a}, c) \in ULD \wedge (u_1, r_1) \in UR \wedge (u_2, r_2) \in UR \wedge (r_1, p_{1a}, c) \in RPC \wedge (r_1, p_{1b}, c) \in RPC$
body $ULD := ULD \setminus \{(u_1, u_2, p_{1a}, c)\} \cup \{(u_1, u_2, p_{1b}, c)\}$; $AC := AC \cup \{(u_2, p_{1b}, c)\}$ end
post $(u_2, p_{1b}, c) \in AC \wedge (u_1, u_2, p_{1b}, c) \in ULD$

Role Delegation Operator The Role Delegation Operator (RDM) is used to simulate errors in delegation of roles. The RDM operator will mutate the delegation rule by replacing the delegator by some other user having different role. For example in the LMS system, one delegation is: Bill (Director) delegates his role to Bob (Secretary). The most important aspect in such a delegation is to establish correct link between delegator-role-delegatee. This means that the right delegator delegates the right role to the delegatee.

Definition 5.13. $RDM(u_1, u_2, r_1, c) : -$
pre $(u_1, r_1) \in UR \wedge (u_2, r_2) \in UR \wedge (u_3, r_3) \in UR \wedge r_1 \neq r_2 \neq r_3 \wedge (u_1, u_2, r_1, c) \in ULD$
body $ULD := ULD \setminus \{(u_1, u_2, r_1, c)\} \cup \{(u_3, u_2, r_3, c)\}$; $UR := UR \cup \{(u_2, r_3)\}$ end
post $(u_2, r_3) \in UR \wedge (u_3, u_2, r_3, c) \in ULD$

5.7.3.2 Advanced Delegation Operators

Advanced delegation operators are described w.r.t advanced properties of delegation such as transfer delegation, temporary delegation, user-specific delegation, role-specific delegation, etc., we introduce mutants to simulate errors that affects the correctness of these types of delegation.

Monotonic Delegation Operators A delegation can be monotonic or non-monotonic. The former specifies that the delegated access right is available to both the delegator and delegatee after enforcing this delegation. The latter means that the delegated role/permission is transferred to the delegatee, and the delegator temporarily loses his rights while delegating them. In this case, the delegation is called a transfer delegation. For instance in LMS, a secretary (Bob) transfers his role to an administrator (Sam) and Bob no longer can use his role during the delegation period.

By mutating the property *monotonicity* of a delegation, we can simulate errors such as the enforcement of a transfer delegation is implemented as a grant delegation, or vice versa, a grant delegation is changed to a transfer delegation. The Transfer to Grant Delegation Operator (T2G) and the Grant to Transfer Delegation Operator (G2T) are defined for performing these mutations.

Definition 5.14. $G2T(u_1, u_2, p, c\&IsMonotonic) :-$
pre $(u_1, p, c) \in AC \wedge (u_1, u_2, p, c\&IsMonotonic) \in ULD$
body $ULD := ULD \setminus \{(u_1, u_2, p, c\&IsMonotonic)\} \cup \{(u_1, u_2, p, c\&IsNonMonotonic)\}$
end
post $(u_1, p, c) \notin AC \wedge (u_1, u_2, p, c\&IsNonMonotonic) \in ULD$

Definition 5.15. $T2G(u_1, u_2, p, c\&IsNonMonotonic) :-$
pre $(u_1, p, c) \notin AC \wedge (u_1, u_2, p, c\&IsNonMonotonic) \in ULD$
body $ULD := ULD \setminus \{(u_1, u_2, p, c\&IsMonotonic)\} \cup \{(u_1, u_2, p, c\&IsMonotonic)\}$ end
post $(u_1, p, c) \in AC \wedge (u_1, u_2, p, c\&IsMonotonic) \in ULD$

Context-based Delegation Operators Delegations are always applied in some context. Test should guarantee correct implementation of context of delegations. Delegation contexts can be temporal, spatial, provisional, pre-requisite or a user declared context. The context is tested by reducing the scope of the context (CR), by extending the scope of the context (CE) and by negating the original context (CN).

Here, we only deal with temporal delegation because of its popularity [63]. The other types of context-based delegations are not considered. We introduce the Temporal Delegation Operator (TDM) to mutate the duration of temporal delegation. To be more specific, we mutate the temporal delegation rule by reducing or expanding the duration of a temporal delegation. For example, we apply TDM for the temporal delegation defined in Definition 5.6 as follows.

Definition 5.16. $TDM(startDate, endDate) :-$
pre $vacation\_period(startDate, endDate)$
body $startDate := laterStartDate \wedge laterStartDate > startDate \wedge laterStartDate < endDate$ end
post $vacation\_period(laterStartDate, endDate)$

Role-Specific Delegation Operators A security policy could allow users having a specific role can only delegate to other users having some role that belongs to the possible delegation targets of that role. This can be seen as the case where the security policy ensures that no conflict of interest can occur by delegation. Roughly speaking, one example is in any case a student must never have permissions of both roles *student* and *lecturer* because he can edit his own grades. That means a lecturer does not have right to delegate his role to his student. To test this kind of restriction we propose the following operators. The Role Delegation Off-Target 1 Operator (RDOT$_1$) will replace the delegatee by another user whose role is not a possible target of the delegator's role. That means there exists a delegation rule at master level saying that a user of this role is a possible delegatee of a user of another role: $(u_1, u_2, r, c) \in MLD$. Note that to keep it general, we use the definition of master-level delegation rule to refer to all kinds of delegation rule specifying constraints, e.g. role-specific delegation, multiple delegation, multi-step delegation, etc.

Definition 5.17. $RDOT_1(u_1, u_2, r, c) : -$
pre $(u_1, u_2, r, c) \in MLD \wedge (u_1, u_3, r, c) \notin MLD \wedge (u_1, u_2, r, c) \in ULD$
body $ULD := ULD \setminus \{(u_1, u_2, r, c\} \cup \{(u_1, u_3, r, c)\}$ end
post $(u_1, u_3, r, c) \notin MLD \wedge (u_1, u_3, r, c) \in ULD$

Vice versa, the Role Delegation Off-Target 2 Operator (RDOT$_2$) will replace the delegator by another user whose role's delegation targets set does not contain the delegatee's role.

Definition 5.18. $RDOT_2(u_1, u_2, r, c) : -$
pre $(u_1, u_2, r, c) \in MLD \wedge (u_3, u_2, r, c) \notin MLD \wedge (u_1, u_2, r, c) \in ULD$
body $ULD := ULD \setminus \{(u_1, u_2, r, c\} \cup \{(u_3, u_2, r, c)\}$ end
post $(u_3, u_2, r, c) \notin MLD \wedge (u_3, u_2, r, c) \in ULD$

Permission-Specific Delegation Operators

In this category, we discuss permission specific operator. The Non-Delegable Permission Delegation Operators (NDPD) will mutate a permission delegation by changing the delegated permission from delegable to non-delegable.

Definition 5.19. $NDPD(u_1, u_2, p, c) : -$
pre $(u_1, u_2, p, c) \in MLD \wedge (u_1, u_2, p, c) \in ULD$
body $MLD := MLD \setminus \{(u_1, u_2, p, c\}$ end
post $(u_1, u_2, p, c) \notin MLD \wedge (u_1, u_2, p, c) \in ULD$

Multiple Delegation Operator Tests should reveal problems in ensuring that the number of concurrent delegations of a specific role/permission does not exceed the threshold defined for it at the master level. To simulate the case where adding a new delegation will make that number exceeds a pre-define threshold of a role/permission, we introduce Multiple Delegation Operator (MultiD).

Definition 5.20. $MultiD(countDelegation(u_1, p, c) < N_m) : -$
pre $(u_1, u_2, p, c) \in MLD \wedge countDelegation(u_1, p, c) = N_m$
body $ULD := ULD \cup \{(u_1, u_2, p, c)\}$ end
post $(u_1, u_2, p, c) \in ULD \wedge countDelegation(u_1, p, c) = N_m + 1$

Multi-step Delegation Operator Tests should also reveal problems in re-delegation of permissions and roles. Some roles and permissions can be defined as re-delegable only after a limited number of times. We mutate the policy by re-delegating a role or permission that is not re-delegable and vice versa. In the mutated policy the delegatee will take the role/permission of delegator. Similarly the role and permissions that should not be redelegated are mutated by re-delegating them. The Re-delegation Operator (ReD) add a new delegation rule into the policy where the delegating permission/role must not be re-delegated any more ($stepCounter = 0$).

Definition 5.21. $ReD(stepCounter(u_1, p, c)) : -$
pre $(u_1, u_2, p, c) \in MLD \wedge stepCounter(u_1, p, c) = 0$
body $ULD := ULD \cup \{(u_1, u_2, p, c)\}$; $stepCounter(u_2, p, c) := stepCounter(u_1, p, c) - 1$ end
post $(u_1, u_2, p, c) \in ULD \wedge stepCounter(u_2, p, c) = -1$

Delegation Removal Operators Tests should be able to detect that a delegation rule is missing. We introduce the Delegation Removal Operator (DR) that removes one of the delegation rules.

Definition 5.22. $DR(u_1, u_2, p, c) : -$
pre $(u_1, u_2, p, c) \in ULD$
body $ULD := ULD \setminus \{(u_1, u_2, p, c\}$ end
post $(u_1, u_2, p, c) \notin ULD$

All the delegation mutants defined in this section are derived from basic delegation types as well as various (advanced) well-known delegation features. Thus, we believe they are representative for testing delegation policies. A demonstration example of using some of these mutants is given in the next section.

5.7.4 Demonstration of Testing Delegation Policy Enforcement

In this section, we demonstrate how the proposed mutation analysis approach has been applied on the prototype system LMS discussed in Section 5.5.

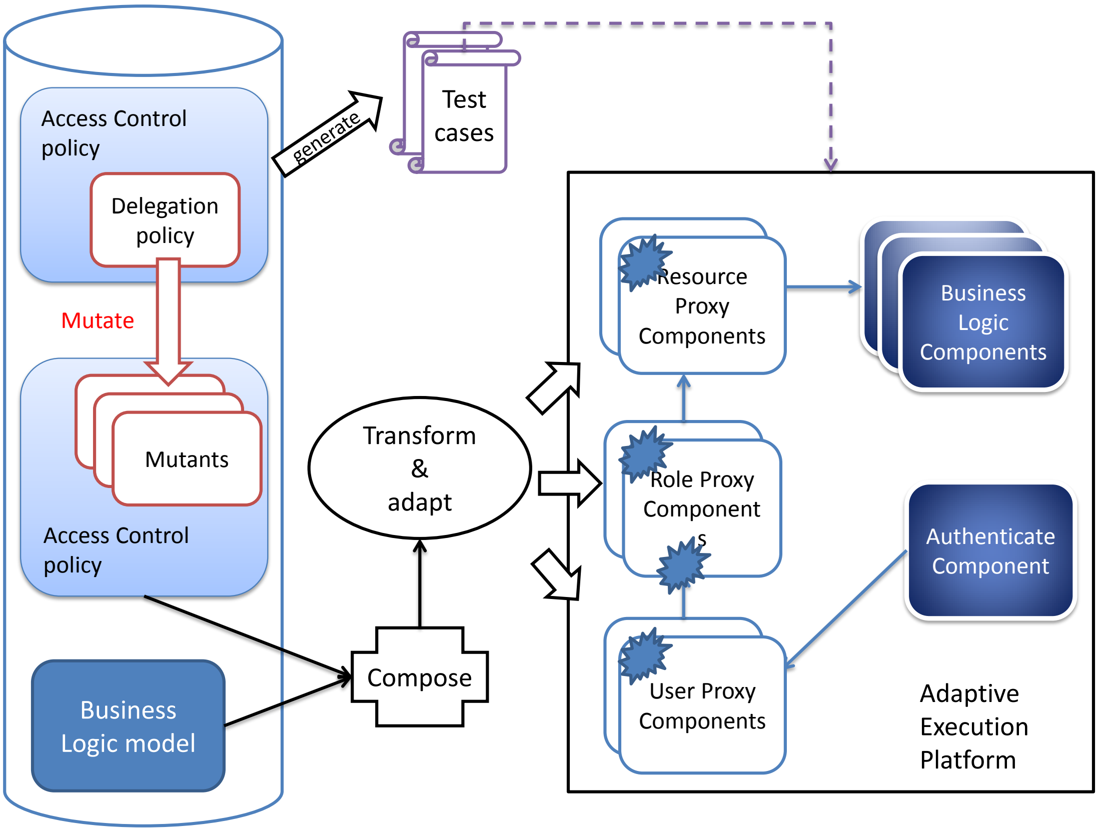


FIGURE 5.11: Mutation Process for testing Delegation Policy enforcement

Figure 5.11 gives an overview of the mutation process for testing delegation. Based on the mutation operators presented in Section 5.7.3 our delegation policy can be easily mutated. Via the model-driven framework described above, those mutated policies can be automatically enforced in the running system.

We performed mutation analysis on the examples described in Section 5.3. We used those three delegation situations to demonstrate the application of the proposed approach. We chose five test cases to test these three types of delegation of our running example. Table 5.4 gives an overview of them.

In case of delegation 1, Bill (*Director*) delegates his permission *consultPersonnelAccount* to Bob (*Secretary*). This simple delegation is mutated in two different ways, first replacing Bill by Sam who is an administrator (by using RDM) and then by replacing *consultPersonnelAccount* with another permission (using PDM), e.g. *consultBorrowerAccount*. We created one test case (TC1) to test if Bob can do *consultPersonnelAccount* after enforcing the delegation rule. The test case was able to kill the second mutant because Bob was not delegated *consultPersonnelAccount* but *consultBorrowerAccount*. However, it did not kill the first mutant because Sam also has the right *consultPersonnelAccount* as Bob. Thus, Sam's rights including *consultPersonnelAccount* have been delegated to Bob, that made him accessible to *consultPersonnelAccount*.

TABLE 5.4: Some preliminary mutation analysis results

Test Case	Killed Mutants	Live Mutants
Test Case-1	PDM (wrong permission mutant)	RDM (delegator fault)
Test Case-2	PDM (wrong permission)	RDM (delegator replaced)
Test Case-3	T2G (wrong type of delegation)	PDM, RDM (wrong delegator, permission)
Test Case-4	PDM (permission replaced)	TDM (CE,CR)
Test Case-5	TDM (CE,CR)	PDM, RDM

In case of delegation 2, we use two test cases to test this delegation. One (TC2) is used to test the correct enforcement of the delegated permission (*createBorrowerAccount*). It resembles the test case for delegation 1, in which we test right allocation of permission. Similarly, it detects mutant in where a wrong permission is inserted (PDM) but fails to kill mutant when the delegator is replaced (RDM). The other test case (TC3) is used to test transfer (monotonicity) property of the delegation, i.e. Alice cannot use *createBorrowerAccount* while transferring it to Jane. For this case, we used T2G to change the type of delegation 2 from transfer to grant. TC3 did kill the mutant of T2G but could not kill the other mutants of PDM and RDM because in those mutants Alice cannot use *createBorrowerAccount*.

We test delegation 3 in a temporal context. We chose two test cases (TC4 and TC5) to kill the four mutants of this delegation. Two mutants are created by injecting permission and delegator related faults (PDM and RDM). The other two mutants are created by reducing and expanding the duration of a temporal delegation (TDM). We created a test case (TC4) similar to TC1. The other test case (TC5) is designed to detect if enforcement of delegation 3 is correct during its active duration. TC5 kills TDM but leaves undetected the mutants of PDM and RDM. Vice versa, TC4 cannot kill the mutant of TDM.

5.7.5 Related Work

Testing of access control policy is a relatively new domain. Some work is available on testing of access control policy with XACML. Martin and Xie [146] propose a fault model for access control policies based on XACML. This fault model was then used to measure the effectiveness of their test cases. The fault model considers access control features composed of permissions and prohibitions. Hu et al. [56] elaborate the conformance checking of access control policy. Other approaches try to test a policy by using state machines. The main focus of such approaches is the generation of test cases for access control policies [141], [148]. To the best of our knowledge, there has not been any approach dedicated to testing delegation enforcement taking into account an extensive delegation model like ours.

Le Traon et al. [139] establish a mutation analysis approach for performing security testing. Xu et al. [238] propose a model based testing approach for access control policies. They use both the policy and its contracts implemented in a number of industry usable languages. The test adequacy is also measured through mutation analysis. El Rakaiby et al. [70] use mutation analysis to test obligations. Their approach (not model-based) is on the same lines with the one proposed here except that their focus is obligation and our focus is delegation. However, there is a big difference because delegation and obligation are two different aspects. Especially, because of the "meta-level" character of delegation w.r.t access control, applying mutation analysis for testing delegation enforcement has to take into account different delegation features in the whole process. Moreover, our mutation analysis approach bases on our model-driven framework showed in Section 5.4 making it easy to leverage model-based testing techniques.

5.7.6 Summary

The process of delegating access rights forms one of the central issues in the administration of an access control policy. The delegation of authority gives users more rights over protected resources so its undisputed implementation and enforcement require a rigorous testing process. In view of this, the present section suggests the use of mutation analysis to test the delegation policy enforcement. To achieve this, a careful and formal analysis of different delegation features was performed. Based on this analysis a set of mutation operators specific for delegation has been derived.

In future, a thorough empirical study using the proposed mutation operators is planed. This will lead in identifying the most useful and effective mutation operators. Moreover, we aim at investigating ways to automatically generate test cases for killing the proposed mutants. Towards this direction we aim at extending existing approaches [195, 194] to deal with delegations. Finally, the integration of model-based testing with the suggested mutation approach will be also researched.

5.8 Conclusions

In this chapter, we have proposed an extensive *Model-Driven Security* approach for adaptive delegation in access control management. By giving a formalisation of access control and delegation mechanisms, we introduced various advanced delegation features that would provide secure, flexible, and efficient access control management. It has been shown that these advanced delegation features can be specified using our delegation DSML. Our DSML supports complex delegation characteristics like temporary, recurrence delegation, transfer delegation, multiple and multi-step delegation, etc. We have also shown that revocation can be dealt with in a simple manner. Another main contribution of this chapter is our adaptive delegation enforcement in which delegation is considered as a "meta-level" mechanism that impacts the access control rules. A complete model-driven framework has been proposed to enable dynamic enforcement of delegation and access control policies that allows the automatic configuration of the system according to the changes in delegation/access control rules. Moreover, our framework also enables an adaptation strategy that better supports co-evolution of security policy and business logic of the system. The model-driven framework proposed in this chapter can be applied for securing (distributed) systems running on different adaptive execution platform like OSGi (Equinox), or an optimised models@runtime framework such as Kevoree. Our approach has been validated via three different case studies with consideration of performance and extensibility issues. Furthermore, a mutation testing approach has been integrated into this work that could provide the basis for building a tool chain of MDS, from modelling till testing.

Chapter 6

MDS with Reusability based on A System of Security Design Patterns

Contents

Model-Driven Security (MDS) for secure systems development still has limitations and open issues to be more applicable in practice. Our systematic review of MDS shows that current MDS approaches have not dealt with multiple security concerns systematically. Besides, catalogs of security patterns which can address multiple security concerns have not been applied efficiently. This chapter presents an MDS approach based on a System of Security design Patterns (SoSPa). In SoSPa, security design patterns are collected, specified as reusable aspect models (RAM) to form a coherent system of them that guides developers in systematically addressing multiple security concerns. SoSPa consists of not only interrelated security design patterns but also a refinement process towards their application. We applied SoSPa to design the security of crisis management systems. The result shows that multiple security concerns in the case study have been addressed by systematically integrating different security solutions based on SoSPa.

This work promotes not only modularity but also reusability with SoSPa and RAM in the model-driven development of secure systems.

6.1 Introduction

Model-Driven Security (MDS) emerged more than a decade ago as a specialised *Model-Driven Engineering* approach for secure systems development, but still has limitations and open issues to be more applicable. Our recent systematic review of MDS [182] shows that multiple security concerns have not been addressed systematically by existing MDS studies. Indeed, interrelations or dependencies among security solutions have not been considered formally, systematically by current MDS approaches. Developing modern secure systems must always address multiple security concerns to minimise different security leaks and to make these systems resilient to different security attacks. A solution to address a specific security concern often depends on other solutions addressing other security concerns. For instance, most authorisation mechanisms depend on authentication mechanisms because before an authorisation decision, the authorisation mechanism should have known the identity of the requester. Authentication mechanisms often rely on encryption mechanisms, especially for distributed systems. Furthermore, there could be a lot of different variations of security solutions to address the same security concern. All urge for an MDS approach that can systematically address multiple security concerns, considering interrelations among security solutions and their variants.

On the other hand, from security engineering's point of view, one of the best practices is the use of security patterns to guide security at each stage of the development process [211]. Patterns are applied in the different architectural levels of the system to realise security mechanisms. So far, catalogs of security patterns are the most accessible, well organised, documented resources of different security solutions for different security concerns, e.g. [72, 220, 211]. But the results of two relevant empirical studies [241, 240] have shown that using existing catalogs of security patterns does neither improve the productivity of the software designer, nor the security of the design. Indeed, security patterns could be applied at different levels of abstraction, e.g. architectural design rather than detailed design. Moreover, the levels of quality found in security patterns are varied, not equally well-defined like software design patterns [95]. Particularly, many security patterns are too abstract or general, without a well-defined, applicable description. There is also a lack of coherent specification of interrelations among security patterns, and with other quality properties like performance, usability. In some catalogs, each security pattern is described having its related patterns, and possible impacts on other quality properties mentioned, but without any more practical details. To the best of our knowledge, none of existing MDS approaches has proposed a *System of Security design Patterns* which provides not only well-defined security design patterns but also interrelations among security patterns that can guide developers in systematically dealing with multiple security concerns.

In this chapter, we propose an MDS framework based on a *System of Security design Patterns* (SoSPa) that allows practitioners to systematically address multiple security concerns in secure systems development. Our security patterns in SoSPa are theoretically based on well-known security design patterns (e.g. in [72, 220, 211]). They are collected, specified as reusable aspect models (RAM) [122] to form a coherent system of them. While not only specifying security patterns at the abstract level like in security patterns catalogs, SoSPa also provides a refinement process supported by RAM to derive the detailed security design patterns closer to implementation. In other words, a software designer can reuse our security design patterns that are specified at different abstraction levels as RAM models. By using SoSPa, an integrated security solution dealing with multiple security concerns can be systematically engineered into a system. Not only the security design patterns but also their interrelations are specified in SoSPa. Based on SoSPa, conflicts and inconsistencies among the applied security solutions in a system design can be detected, resolved, or eliminated systematically. This may help to improve the security in a system design against different security threats. Because we propose an MDS development framework, SoSPa is built on a meta-model, which is extended from RAM meta-model. Our MDS framework allows selecting, refining, composing security design patterns to systematically build security solution models, and then automatically integrating them into a target system design. The contribution of this chapter is three fold: 1) hierarchical RAM models with a refinement process for specifying security design patterns from abstract level till detailed design level; 2) explicitly specified interrelations among security design patterns for systematically dealing with multiple security concerns; 3) an MDS framework supporting secure systems development based on SoSPa.

In the remainder of this chapter, Section 6.2 provides some fundamental concepts and motivational examples for our MDS approach. Then, we present our MDS approach based on SoSPa in Section 6.3, and some key security design patterns of SoSPa in Section 6.4. Section 6.5 shows how our approach has been evaluated and discussed. The position of this work compared to related work is given in Section 6.6. Finally, Section 6.7 presents our conclusions and future work.

6.2 Background and Motivational Examples

We briefly recall RAM approach and why RAM is a good candidate for specifying *a system of security design patterns*. Then, we discuss some motivational examples for our work.

6.2.1 Reusable Aspect Models

RAM [122] is an aspect-oriented multi-view modelling approach with tool support for aspect-oriented design of complex systems. In RAM, any concern or functionality that is *reusable* can be modelled using class, sequence, and state diagrams in an aspect (RAM) model. A RAM model can be (re)used within other models via its clearly defined *usage and customisation interfaces* [13]. The usage interface of a RAM model consists of all the *public* attributes and methods of the class diagrams in the model. The customisation interface of a RAM model consists of all the *parameterised* model elements (marked with a vertical bar |) of the partially defined classes and methods in the model. A RAM model can be (re)used by composing the *parameterised* model elements with the model elements of other models. A RAM model can also reuse other RAM models in a hierarchical way. RAM weaver is used to flatten aspect hierarchies to create the composed design model.

We find that security patterns can be well specified by using RAM approach. RAM's multi-view modelling ability make it possible to capture even complex semantics of security patterns. The hierarchical modelling support of RAM enables a refinement process in which security patterns can be refined from abstract level till detailed design level. Fig. 6.1 shows a RAM model of a *Session* pattern which reuses the generic RAM model of *Map* in Fig. 6.2 (or *ZeroToManyAssociation* alternatively, Fig. 6.3) [122]. The RAM model of *Map* is composed of a generic data container `|Data` using a "Map" structure to store data in pairs of `|Key` and `|Value`. *Session* reuses *Map* by composing parameterised elements of *Map* with model elements of *Session* as can be seen in the "*Map instantiation*" box. For example, the mapping `|Value` to `|Attribute` means that attributes in a session are stored as `|Value` objects managed by the Map structure. The `|Attribute` element itself is a parameter. Any object can be stored in a session by mapping it to the `|Attribute` parameter. The RAM model of *Session* itself has the customisation interface comprises the parameterised classes `|Session`, `|SessionID`, `|Attribute`, and `|AttributeType`. For example, `|SessionID` is a parameterised class which will be instantiated as a unique identity associated with a session.

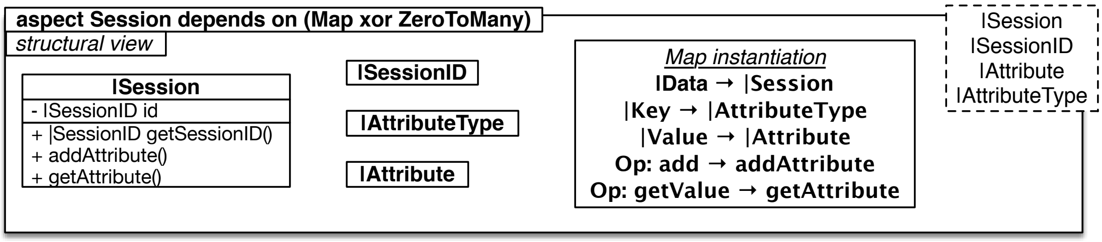


FIGURE 6.1: Aspect *Session* pattern

6.2.2 Motivational Examples

This section first recalls the case study of designing and developing Crisis Management Systems (CMS), particularly a Car Crash Crisis Management System (CCCMS) described in [125]. Next, CMS's potential misuse cases related to multiple security concerns are described. Then, we show why dealing with multiple security concerns systematically is very hard, even by leveraging security patterns in existing catalogs.

Briefly mentioning, in CMS a crisis can be created, processed by executing the rescue missions defined by a super observer, and then assigning internal and/or external resources. Fig. 6.4 shows a partial design of CMS, for creating a rescue mission. CMS are also security-critical systems whose different users must be authenticated, authorised to execute different tasks, sensitive data being communicated via different networks must be protected, and responsibility of users must be clearly traced. In [125], only a simple use case for CMS user authentication is provided. We show different security threats/misuse cases of CMS (informally) as follows.

Misuse cases related to user accounts, access control: [MUC-A1] An attacker impersonates a *CMSEmployee* after obtaining the user password by guess and try; [MUC-A2] A colleague of an authenticated user misuses the system on the authenticated user's working device while it is being left unattended and accessible; [MUC-A3] A *CMSEmployee* gains disallowed access to the protected resources. *CMSEmployees* must only have access rights according to their assigned roles; [MUC-A4] A *CMSEmployee* has access rights to the system as a *FirstAidWorker* but also a *SuperObserver*. Conflict-of-interest roles, e.g. *FirstAidWorker* and *SuperObserver*, cannot be assigned to the same user.

Misuse cases related to accountability data: [MUC-B1] A *CMSEmployee* has received mission information concerning a car crash but ignores or overlooks some crucial information, and does not accept the mission. The rescue mission fails. When confronted, the *CMSEmployee* denies having received the mission information. [MUC-B2] A *SuperObserver* wrongly created a rescue mission which led to its failure. The *SuperObserver* manages to delete the corresponding log entry of his wrong action, and denies it.

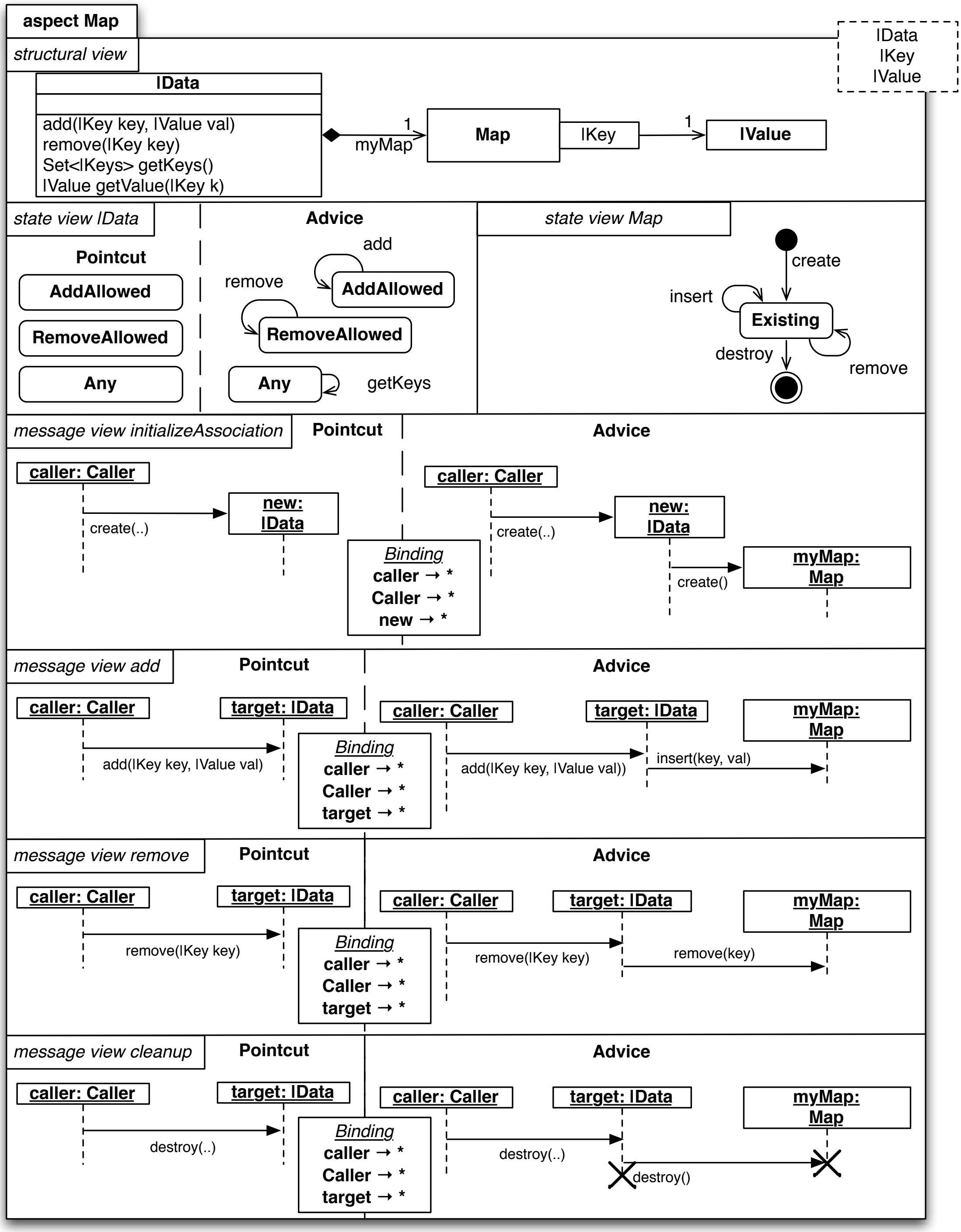


FIGURE 6.2: Aspect *Map* [122]

Misuse cases related to transmitted data: [MUC-C1] An attacker intercepts usernames and passwords barely transmitted from client to server to impersonate a valid *CMSEmployee*; [MUC-C2] An attacker intercepts the mission information about a crisis barely transmitted from the system to *CMSEmployee*. Similarly, an attacker may intercept victim's identity and/or medical history information transmitted from the *HospitalResourceSystem* to the CMS and/or from the CMS to a *FirstAidWorker*. Advanced

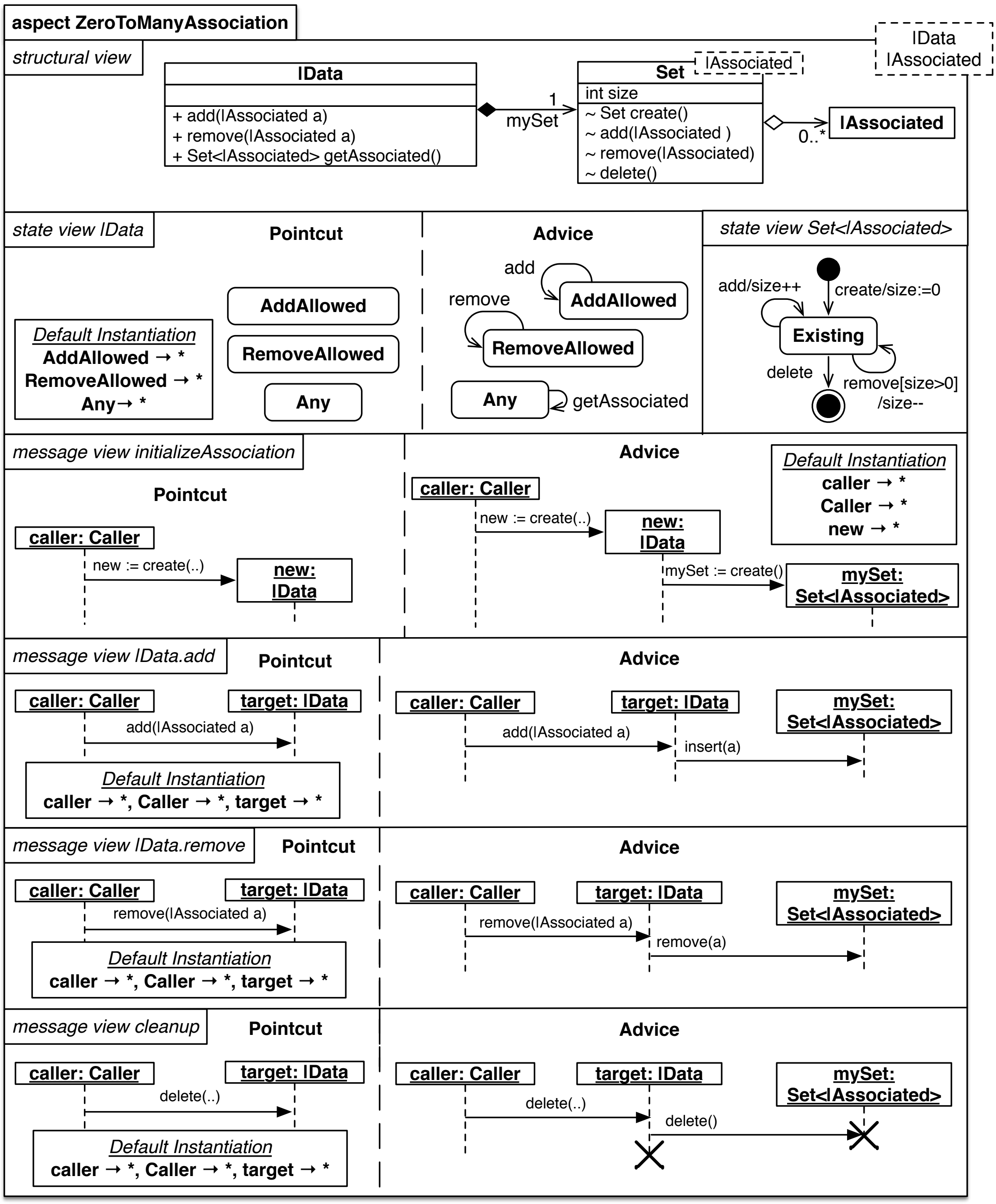


FIGURE 6.3: Aspect *ZeroToManyAssociation* [122]

attacker even could modify the transmitted victim's medical history information.

For most security experts, not saying security novices, finding the best possible solutions addressing multiple security concerns, e.g. described in the misuse cases, and integrating them properly into the CCCMS would be a very big challenge. Developing and integrating different security solutions addressing multiple security concerns into a system is hard, making them work together consistently is harder. Leveraging security patterns seems to be a good approach for practitioners because security patterns are

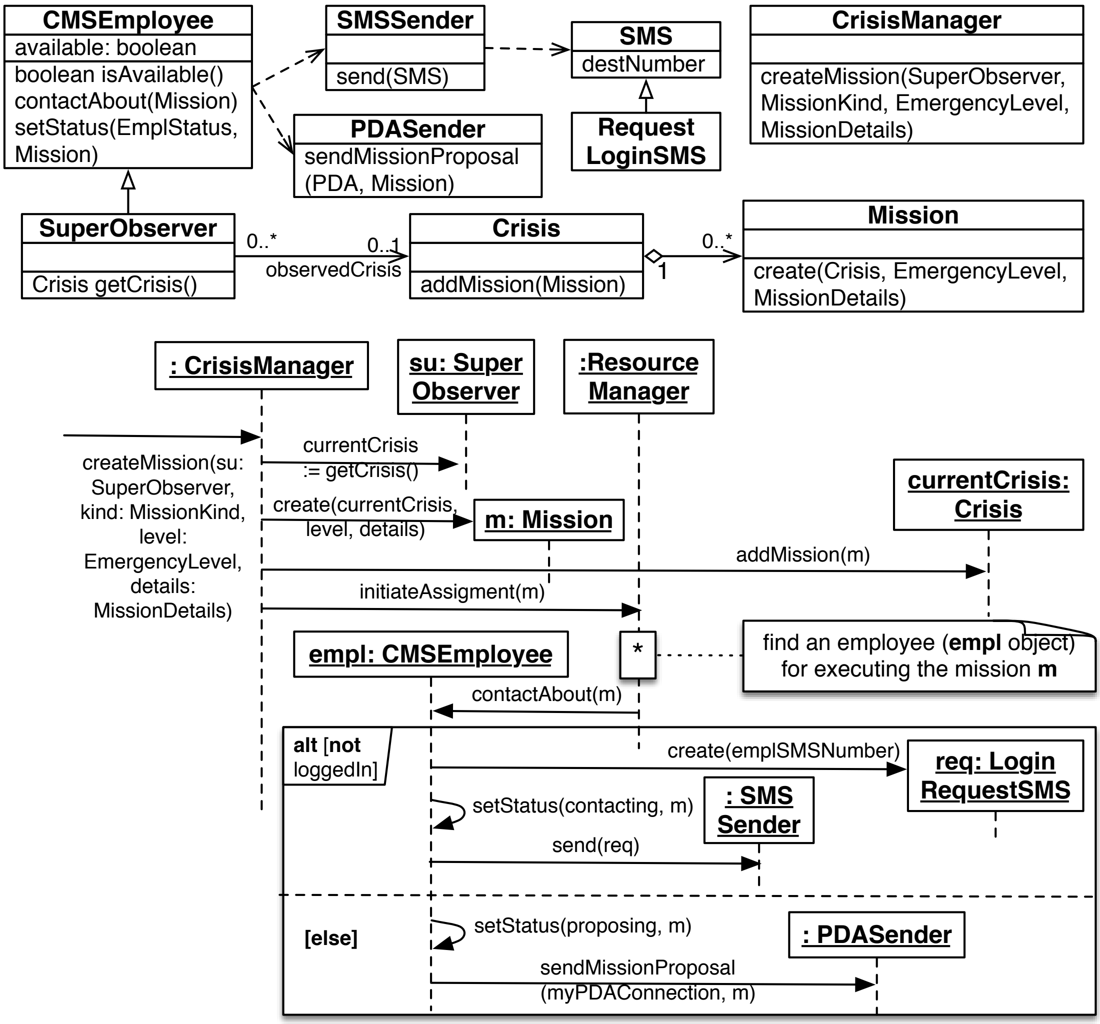


FIGURE 6.4: A partial design of CMS with *createMission* function [125]

fairly well documented to address different security concerns. Moreover, some security patterns also contain some informal inter-pattern relations, and constraints regarding other quality properties such as performance to guide the patterns selection process. For tackling [MUC-A1], one may decide to use the patterns in [211], [72], e.g. to ensure the complexity of user passwords, make password reset frequently, combine user passwords with one-time-password (OTP). [MUC-A2] and [MUC-A3] can be mitigated by using the patterns of access control in [72, 211, 220]. For tackling [MUC-B1] and [MUC-B2], the *Audit Intercepter* and/or *Secure Logger* patterns [220], or the *Security Logger and Auditor* pattern [72] can be used. For tackling [MUC-C1] and [MUC-C2], one may decide to use the *Secure Channel* pattern or *Secure Pipe* pattern [220], or the *TLS* pattern in [85].

But there is still a big gap between the intention and practical application of security patterns. Security patterns are often too abstract with good intention but no clear semantics that make them difficult to be implemented and applied, especially together.

One can see that in existing catalogs of security patterns, e.g. [72, 220, 211], interrelations among patterns and other constraints are only briefly mentioned but not concretely specified to be applicable. All of these could lead to inappropriate implementation and application of security patterns. For example, not well-thought design decision could lead to a weak user passwords authentication solution that allows a *FirstAidWorker* to guess and successfully impersonate a *SuperObserver*. Improperly integrating authentication, user session, and authorisation solutions could lead to access rights misused, and sensitive data leaked. More tricky, wrongly implementing an encryption channel for data transmission and also an auditing mechanism that intercepts and records the transmitted data may result in encrypted log entries that are useless for auditing purposes. Similarly, constructing a logging solution for accountability must be aware of an existing authorisation solution in the same system to produce the logs correctly. Depending on how these two work together, the logs might contain nothing, or meaningless info, or different types of info about successful executions of method calls, or failed authentication/authorisation checks for the method calls, or sometimes also successful authentication/authorisation checks. A sound approach for systematically addressing multiple security concerns in secure systems development is needed, but has not existed yet at least in the MDS research area.

6.3 Our Model-Driven Security Approach based on SoSPa

6.3.1 Overview of our approach

Our MDS approach is based on a *System of Security design Patterns* (SoSPa). Fig. 6.5 displays our meta-model of SoSPa (SoSPa-MM) that is an extension of RAM meta-model [122]. The core elements of SoSPa-MM are depicted in white. The rest are core elements of RAM meta-model. SoSPa aims at systematically addressing the globally accepted security concerns such as confidentiality, integrity, availability, accountability. Thus, SoSPa is composed of an extensible set of security solution blocks, e.g. authentication, authorisation, cryptography. Each security solution block consists of interrelated security design patterns. To support the selection of security design patterns, we use feature modelling as in software product line engineering to capture the variability and interrelations of security patterns. Specified by a feature model-like diagram, each security solution block can be used to form a specific, customised security solution. Each security design pattern in SoSPa contains all well-structured elements such as *context*, *problem*, *consequences* as can be seen in well-documented security patterns of existing catalogs. More than that, inter-pattern relations are captured and explicitly specified at the conceptual level as well as model level by RAM models. Fig. 6.5 shows that each

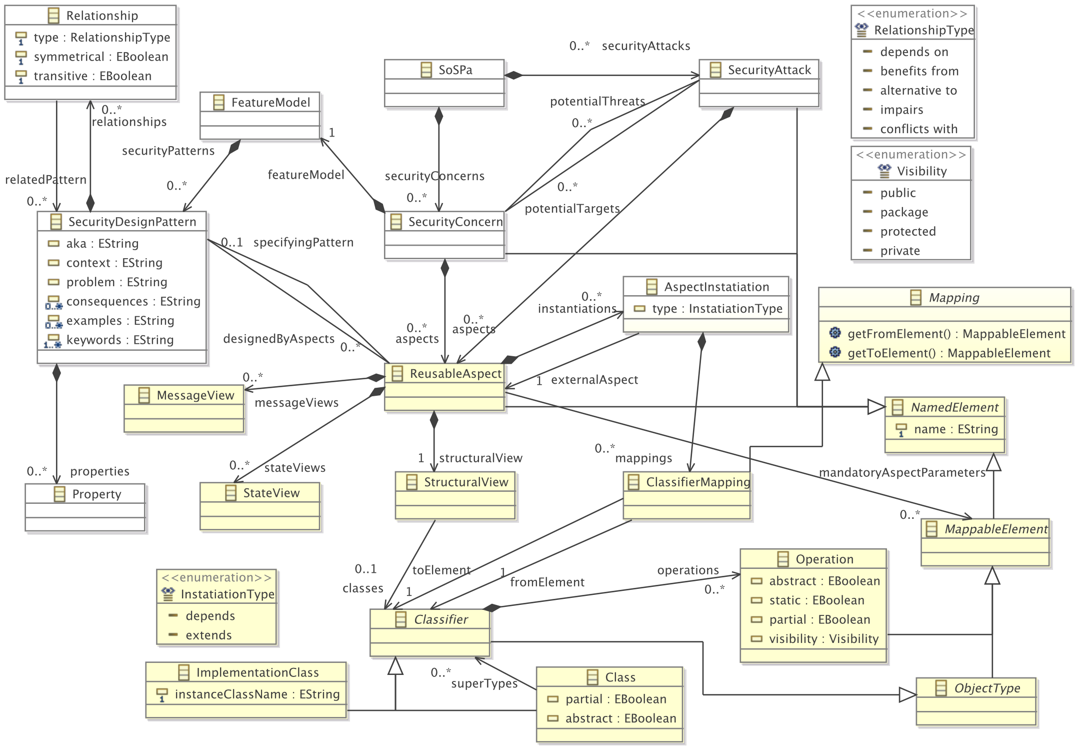


FIGURE 6.5: Meta-model of *System of Security design Patterns* (SoSPa-MM)

SecurityDesignPattern is associated with *ReusableAspect(s)* that realised the pattern. In other words, security design patterns are specified as reusable aspect models (RAM) to form a coherent system of them, i.e. SoSPa. The interrelations among security patterns are categorised into five types as specified in *RelationshipType*. We capture the core relation types among security patterns, i.e. *depends on*, *benefits from*, *alternative to*, *impairs*, *conflicts with*. These relations can be transitive and/or symmetrical.

Five main security solution blocks of SoSPa are *Authentication*, *Authorisation*, *Cryptography*, *Auditing*, and *Monitoring* as can be seen in Fig. 6.6. Derived from security requirements, a customised security solution can be built up from a combination (OR relation) of these security solution blocks. Each feature (node) of the feature model can be associated directly with a RAM model. For example, *Authentication* feature is directly specified by a RAM model named *Authentication*. The features with underlined names are security patterns which can be refined by composing the hierarchical RAM models realising them. For example, *Security Session* pattern is realised by the RAM model *SecuritySession*, and also the other relevant RAM models like *SessionManager*, *Session*. Some low-level features are not really security patterns but generic RAM models which help building security patterns, e.g. *Map*, *ZeroToManyAssociation*. Because in SoSPa, security patterns are built on hierarchical RAM models (see Section 6.4), the interrelations among security patterns are actually specified at the model/design level.

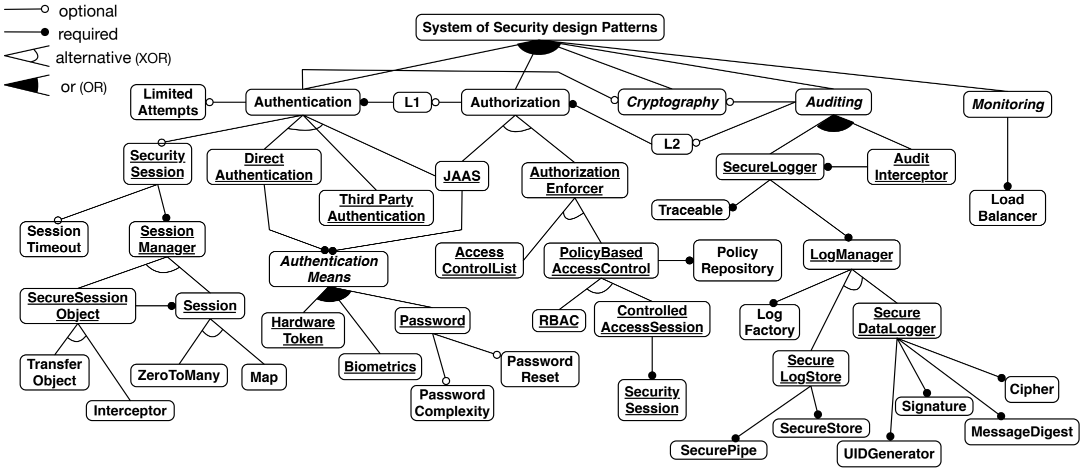


FIGURE 6.6: A partial feature model of SoSPa

In other words, the interrelations are specified based on the relations of RAM models that the security patterns are built on. We elaborate more on this in Section 6.3.3.

6.3.2 Pattern-Driven Secure Systems Development Process

This section presents the development process in three main stages, especially emphasising the selection and composition of security design patterns into a target system design.

[Security threats identification& analysis]: This is not the focus of this work. We assume that misuse cases are created in this stage. Attack models might be created from risk analyses, e.g. using the CORAS framework as discussed in [85].

[Security design patterns selection and application]
Step 1 - Constructing security solutions from the security patterns in SoSPa: For each security concern, the interrelations specified in the feature model (Fig. 6.6) are used to select the most appropriate security design patterns, i.e., the pattern that best matches with the context and the security problem, most satisfies the interrelations with the other already selected security design patterns, and maximises the positive impact on relevant quality properties like usability, performance. All the RAM models of the selected security design patterns and other required RAM models are woven into the RAM model of the top most feature in the hierarchy corresponding to the security concern of the feature model. This step derives a detailed RAM design of a customised security solution for the concern, including its *customisation interface* and *usage interface*. For example, to construct a customised authentication solution, all the selected features under *Authentication* feature are woven into it. The output of this step is a complete RAM model, i.e. the woven RAM model of the authentication solution.

This woven RAM model of the authentication solution later can be integrated into a base system model via its customisation interface. More details can be find in the case study described in Section 6.5.

Step 2 - Defining mappings to integrate the newly built security solutions to a base system model: For each selected security pattern, use the customisation interface of the generated design to map the generic design elements to the application-specific context. This step generates the mappings of the parameterised elements in the security design pattern with the target elements in the target system design. Any constraints/conflicts between mappings of all the selected security design patterns need to be resolved. Most constraints are predefined by SoSPa for the obvious interrelations (e.g. L1 and L2 discussed in the case study). Some ad-hoc constraints might need to be provided by the designer in some rare cases when the RAM weaver gives warning about potential conflicts while weaving RAM models [122].

Step 3 - Weaving the security solutions into the base system model: All the security solutions are automatically woven into the target system design. The mappings from previous step are the input for this weaving process.

[Verification&validation of security patterns application]: Analyse the woven secure system against the attack models obtained before. The attack models can be used for formal verification of security properties in the woven model, or can be used for test cases generation like in a security testing approach. This is part of future work.

6.3.3 Interrelations of Security Patterns in SoSPa

This section shows how five interrelations among security patterns at conceptual level can be realised at detailed design (model) level in SoSPa.

6.3.3.1 Depend-on relation

Security pattern X *depends on* security pattern Y means that X will not function correctly without Y. This relation is not symmetrical, but transitive. In SoSPa, this relation is specified as the mandatory “required” relation among RAM models that realise the security patterns. Let security pattern X be realised by RAM model A. Security pattern Y is realised by RAM model B. X depends on Y means that model A (directly or indirectly) depends on (requires) model B. Thus, model A realising security pattern X can only be applicable to any base system if all the RAM models that A depends on, such as model B, have been woven into A. In Fig. 6.6, the patterns of *Authorisation* and *Auditing* depends on the patterns of *Authentication*. That means security solution of *Authorisation* must be completed by a security solution of *Authentication*.

6.3.3.2 Benefit-from relation

Security pattern X *benefits from* security pattern Y means that implementing Y will add to the value already provided by implementing X. At design level, X benefits from Y if RAM model A realising X *optionally* depends on RAM model B realising Y. For example, *Authentication* patterns can benefit from *SecuritySession* pattern. But it is up to designers to decide if the chosen *Authentication* pattern needs to use *SecuritySession*.

6.3.3.3 Alternative-to relation

Security pattern X is *alternative* to security pattern Y means that X provides a similar security solution like Y's. The designer can choose either X or Y for addressing the same security problem. For instance, *DirectAuthentication* pattern is alternative to *ThirdPartyAuthentication*.

6.3.3.4 Impair-with relation

Security pattern X *impairs* with security pattern Y means that X and Y are not recommended together. In case X and Y are both selected for working together, they may result in inconsistencies. A conflict resolution could be provided to make them working consistently together. For example, *Load Balancer* pattern impairs with *Security Session* pattern. If no conflict resolution can be found, we say that X conflicts with Y.

6.3.3.5 Conflict-with relation

Security pattern X *conflicts with* security pattern Y means that implementing Y in a system that contains X will result in inconsistencies. An example is that the *Audit Interceptor* pattern could conflict with the *SecureChannel* pattern.

6.4 Security Design Patterns in SoSPa

This section presents some key security design patterns of SoSPa. More details can be found in Appendix D. We show how hierarchical RAM models are used to specify security patterns from abstract level till detailed design level. Besides, each security pattern is presented with its interrelations to the others.

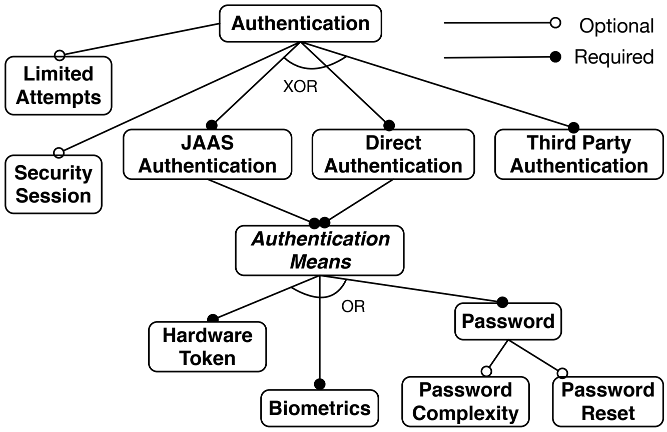


FIGURE 6.7: A feature model of *Authentication*

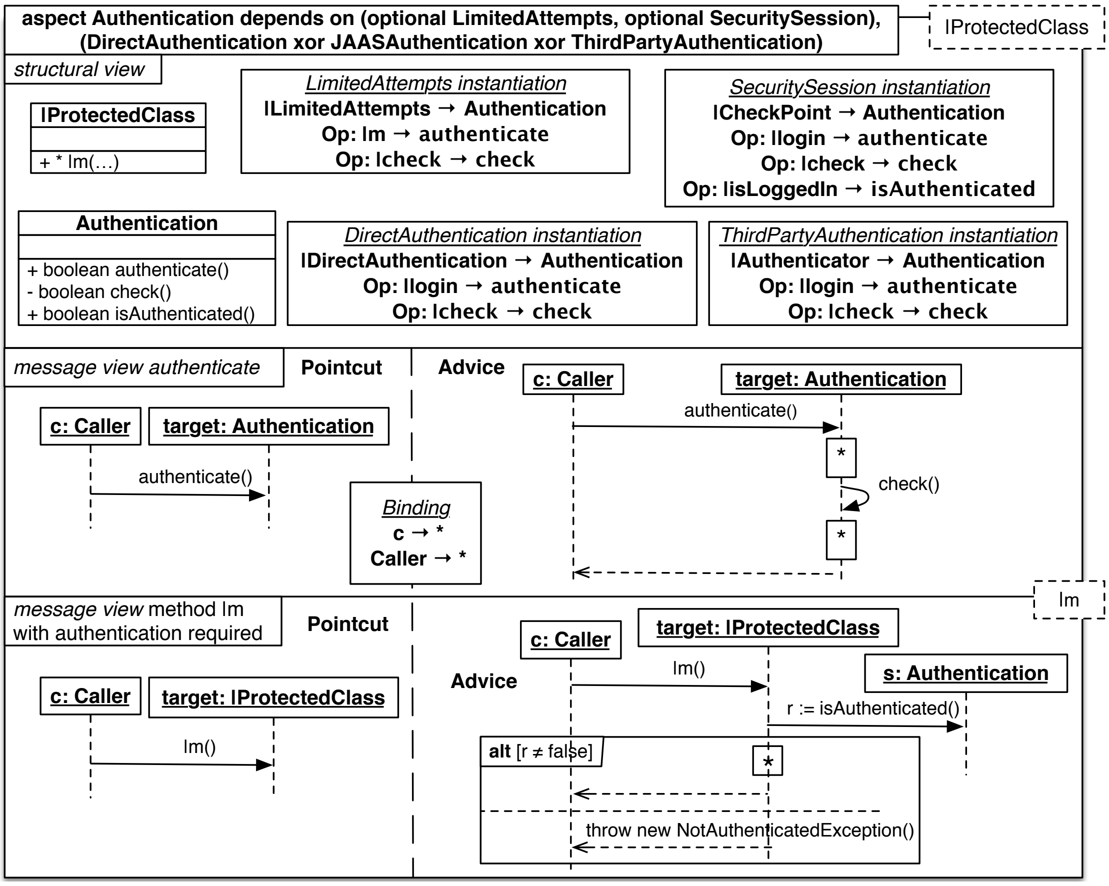

aspect Authentication depends on (optional LimitedAttempts, optional SecuritySession), (DirectAuthentication xor JAASAuthentication xor ThirdPartyAuthentication)
IProtectedClass
structural view
IProtectedClass
+ * Im(...)
LimitedAttempts instantiation
ILimitedAttempts → Authentication
Op: Im → authenticate
Op: Icheck → check
SecuritySession instantiation
ICheckPoint → Authentication
Op: Ilogin → authenticate
Op: Icheck → check
Op: IisLoggedIn → isAuthenticated
Authentication
+ boolean authenticate()
- boolean check()
+ boolean isAuthenticated()
DirectAuthentication instantiation
IDirectAuthentication → Authentication
Op: Ilogin → authenticate
Op: Icheck → check
ThirdPartyAuthentication instantiation
IAuthenticator → Authentication
Op: Ilogin → authenticate
Op: Icheck → check
message view authenticate
Pointcut
Advice
c: Caller
target: Authentication
authenticate()
c: Caller
target: Authentication
authenticate()
*
check()
*
Binding
c → *
Caller → *
message view method Im with authentication required
Pointcut
Im
c: Caller
target: IProtectedClass
Im()
Advice
c: Caller
target: IProtectedClass
Im()
s: Authentication
r := isAuthenticated()
alt [r ≠ false]
*
throw new NotAuthenticatedException()

FIGURE 6.8: Aspect *Authentication*

6.4.1 Authentication Patterns

Deciding which authentication mechanism to employ is very important because it might influence other security mechanisms like authentication, encryption. Selecting an authentication mechanism may affect the selection of other security mechanisms.

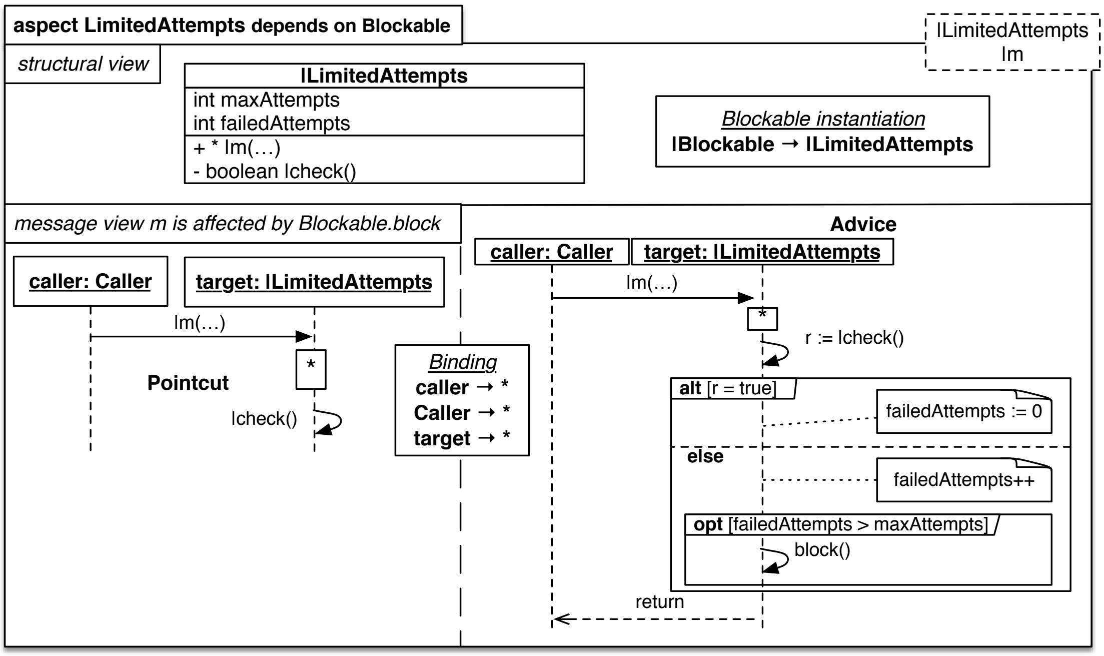


FIGURE 6.9: Aspect *LimitedAttempts* adopted from [122]

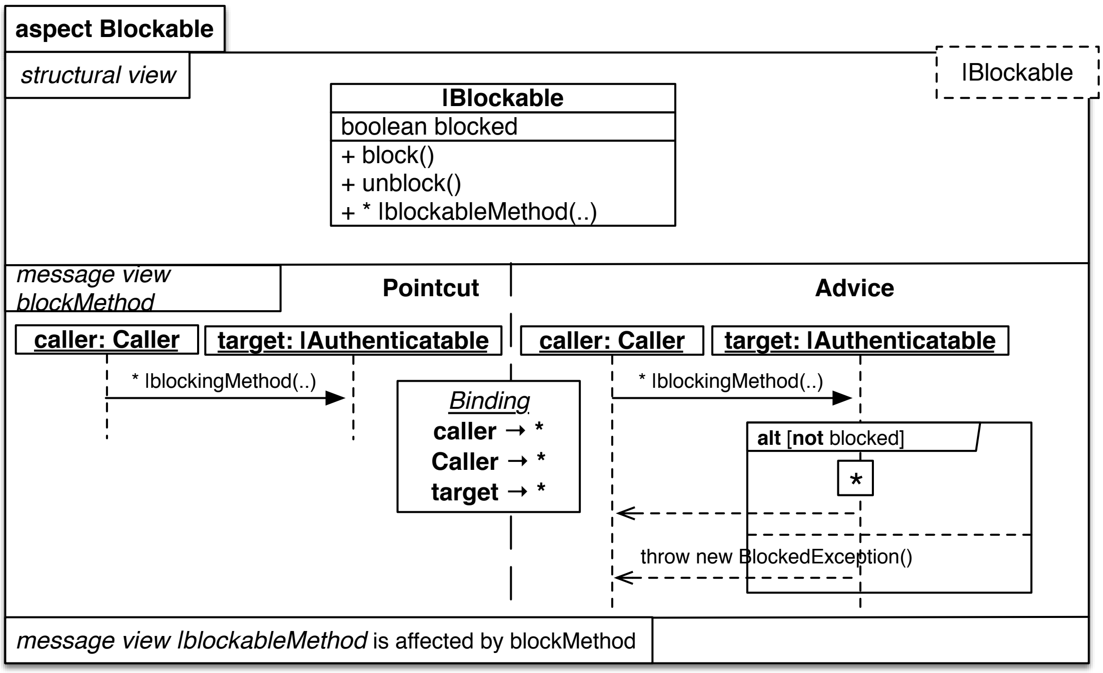


FIGURE 6.10: Aspect *Blockable* [122]

The main security patterns for authentication can be seen in Fig. 6.6. The *Authentication* feature is specified by a RAM model with the most basic authentication logic (Fig. 6.8). For every call to any protected method `|m` of any protected class `|ProtectedClass`, the caller must be already authenticated before method `|m` is executed. There are two

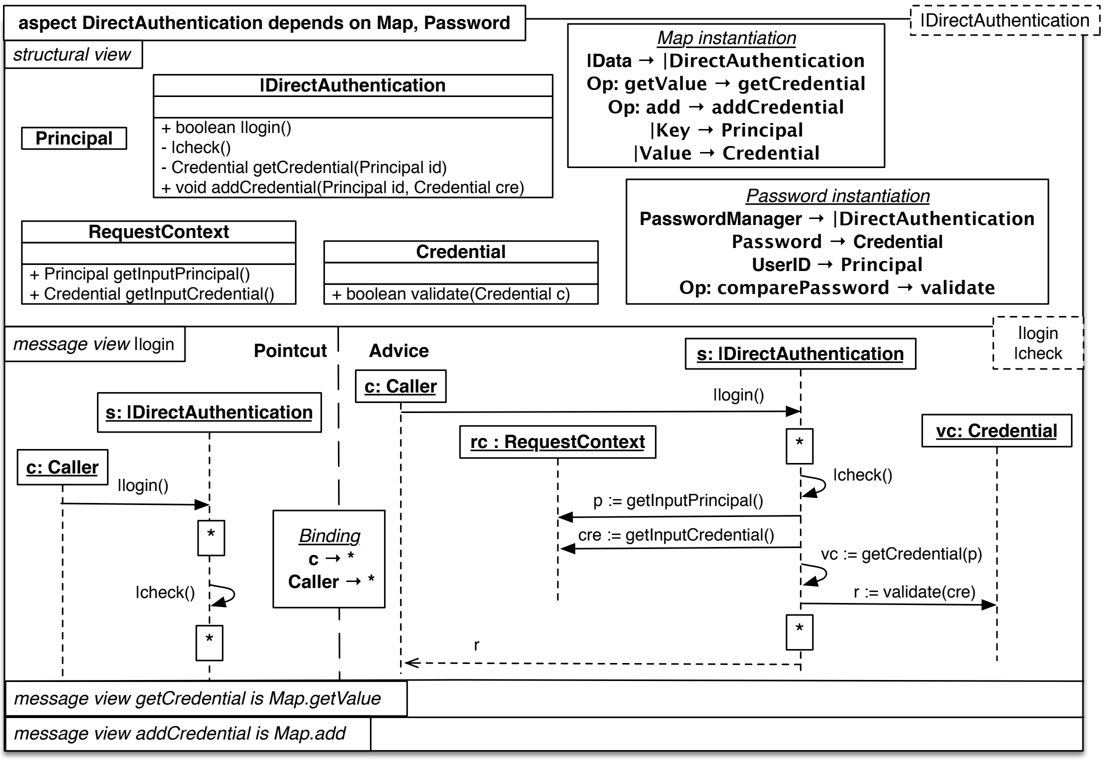


FIGURE 6.11: Aspect *DirectAuthentication*

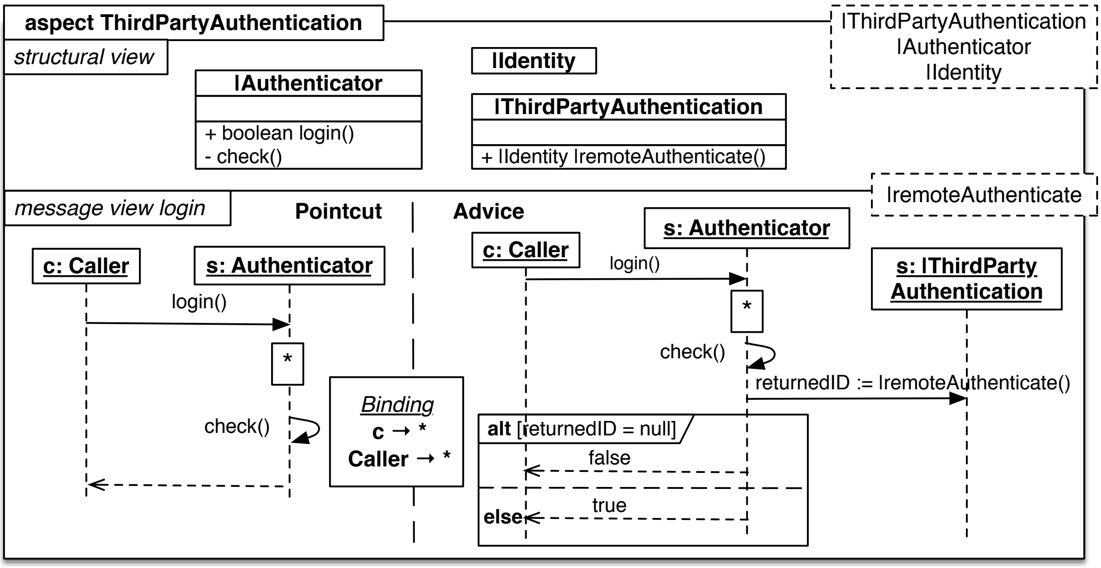


FIGURE 6.12: Aspect *ThirdPartyAuthentication*

optional features that *Authentication* can reuse: *LimitedAttempts* (Fig. 6.9) and *SecuritySession* (Fig. 6.14). *LimitedAttempts* adopted from [122] specifies that an authentication request is blocked after some consecutive unsuccessful authentication attempts. It reuses the *Blockable* aspect [122] (see Fig. 6.10).

The underlying authentication mechanisms are abstract and to be refined by composing the model elements `|Authentication`, `|authenticate`, and `check` with the parameterised elements of the selected RAM models, e.g. *SecuritySession*, *DirectAuthentication* that *Authentication* depends on. The feature model in Fig. 6.7 shows that designer could choose different alternatives below the *Authentication* feature for designing a customised authentication solution, e.g. *DirectAuthentication*, *ThirdPartyAuthentication*, or *JAASAuthentication* [220].

Fig. 6.11 shows the RAM model of *DirectAuthentication* pattern which is based on the *Authentication Enforcer* pattern in [220] and the *Authenticator* pattern in [72]. *RequestContext* contains the user's principal and credential extracted from the protocol-specific request mechanism. An instance of *RequestContext* is often provided by a specific implementation framework. We do not discuss this *RequestContext* class in details. By using the input *Principal*, the *DirectAuthentication* retrieves the corresponding *Credential* from an identity store that it manages using a *Map* aspect. The input *Credential* is checked against the retrieved Credential. Using *DirectAuthentication* requires designer to decide on what kinds of shared secrets to be used for authentication. The abstract aspect *AuthenticationMeans* shows that share secrets could be user password, biometrics, or hardware token (one time password, OTP). These features could be used together, e.g. user password with OTP. If using user password for authentication, the *Password* pattern (see Fig. D.1 in Appendix D) will be woven into the *DirectAuthentication* aspect.

Third-party authentication provider (*ThirdPartyAuthentication*, Fig. 6.12) can also be used to validate client's credentials. The main idea of this pattern is to map the authentication process to a proxy to call the authentication method provided by a third-party authentication provider. Shared secrets among the third-party provider and their clients are invisible to the authentication solution being constructed.

On the other hand, session can bring more benefits to an authentication solution, e.g. for maintaining an authenticated status. We describe security session patterns in the next section. The *SecuritySession* pattern is optional to *Authentication*.

Note that the order of dependencies specified on the top of each RAM model is important to make the patterns work consistently together. For example, the order of weaving dependencies into *Authentication* must be from left-to-right, and then top-down, i.e. *LimitedAttempts*, *SecuritySession*, *DirectAuthentication*. The orders of weaving are also part of SoSPa to provide for designers. The RAM weaver can execute the orders of weaving dependencies automatically. We elaborate more on this in the case study given in Section 6.5.

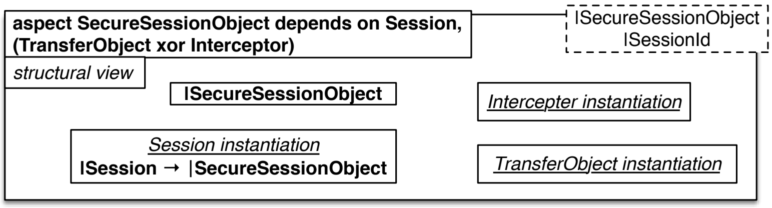

aspect SecureSessionObject depends on Session,
(TransferObject xor Interceptor)
|SecureSessionObject
|SessionId
structural view
|SecureSessionObject
Intercepter instantiation
Session instantiation
|Session → |SecureSessionObject
TransferObject instantiation

FIGURE 6.13: Aspect *SecureSessionObject*

6.4.2 Security Session Patterns

The feature model in Fig. 6.6 shows how the aspects related to security session are organised. The patterns for security session are based on the *Security Session* pattern in [211], the *Controlled Access Session* pattern in [72], and the *Authentication Enforcer*, *Secure Session Object* patterns in [220]. In SoSPa, the *SecuritySession* pattern depends on the *SessionManager* aspect for managing sessions. The designer can choose between a generic *Session* pattern showed in Fig. 6.1 or the *SecureSessionObject* pattern [220]. The *SecureSessionObject* pattern is based on the *Subject Description* pattern in [201] and the *Secure Session Object* pattern in [220]. It reuses the *Session* aspect by specifically defining generic attributes of a security session. Logically, if the validation process in authentication returned by the `|check` method is successful, a new session object is created. Then, any security-related information can be stored in this object, e.g. the validated *Principal*. The authenticated status is associated with the session as long as the session is active. The *SessionTimeout* pattern provides a timing mechanism that requests authenticating again if the corresponding session is expired.

The *SecuritySession* pattern in Fig. 6.14 shows how a *SecuritySession* is created and used for maintaining the authenticated status of an object. After a `|login` request, the *CheckPoint* first checks if the caller has already been authenticated by looking up in the *SessionManager* for any *SecuritySession* associated to the *caller*. If none already established session found, the validation process is performed by the `check` function. This `login` function will be actually instantiated later, e.g. by the `login` method in the *DirectAuthentication* aspect when both *SecuritySession* and *DirectAuthentication* are woven into the *Authentication* aspect. If the validation process is successful, the *CheckPoint* calls the *SessionManager* to create a new *SecuritySession* object and add the validated *Principal* to this object.

The *SessionManager* in Fig. 6.15 reuses a generic *Map* aspect to control sessions. The *Key* element of *Map* is instantiated with *SessionID* object. The *Value* element of *Map* is instantiated with *Session* object. Via the *SessionManager*, sessions can be created, retrieved, and disposed.

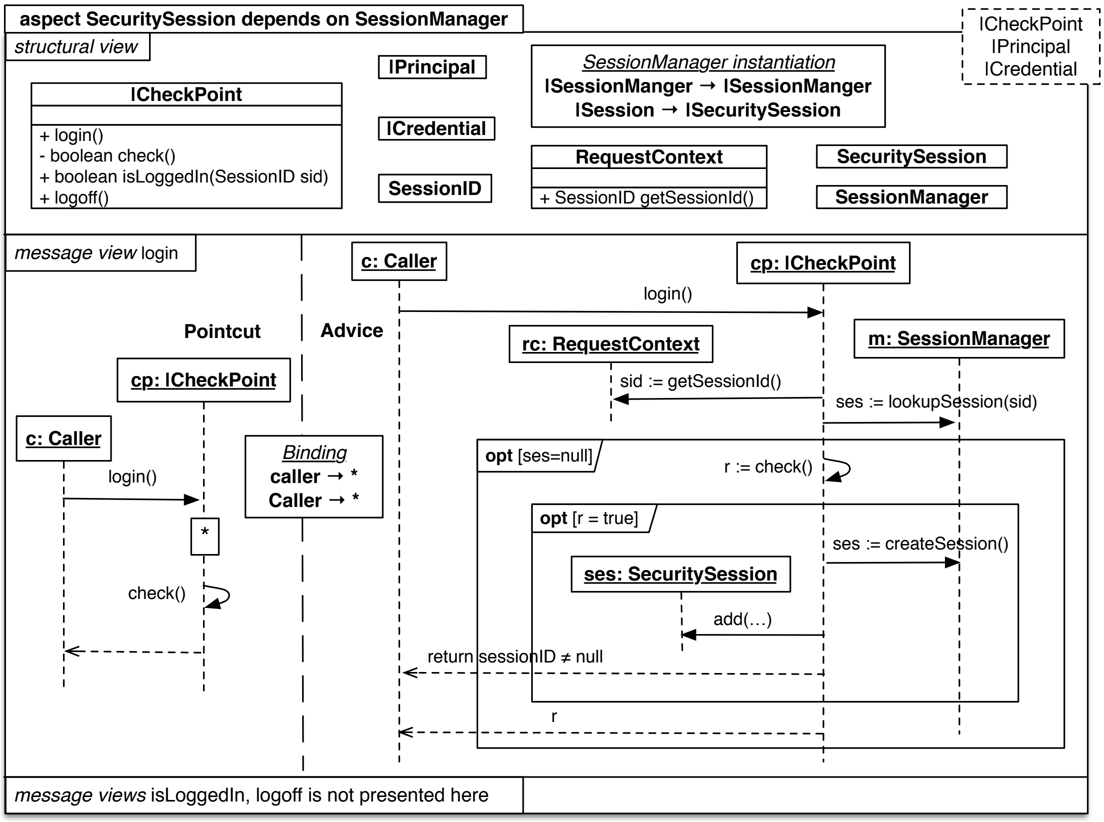


Figure 6.14: Aspect *SecuritySession*

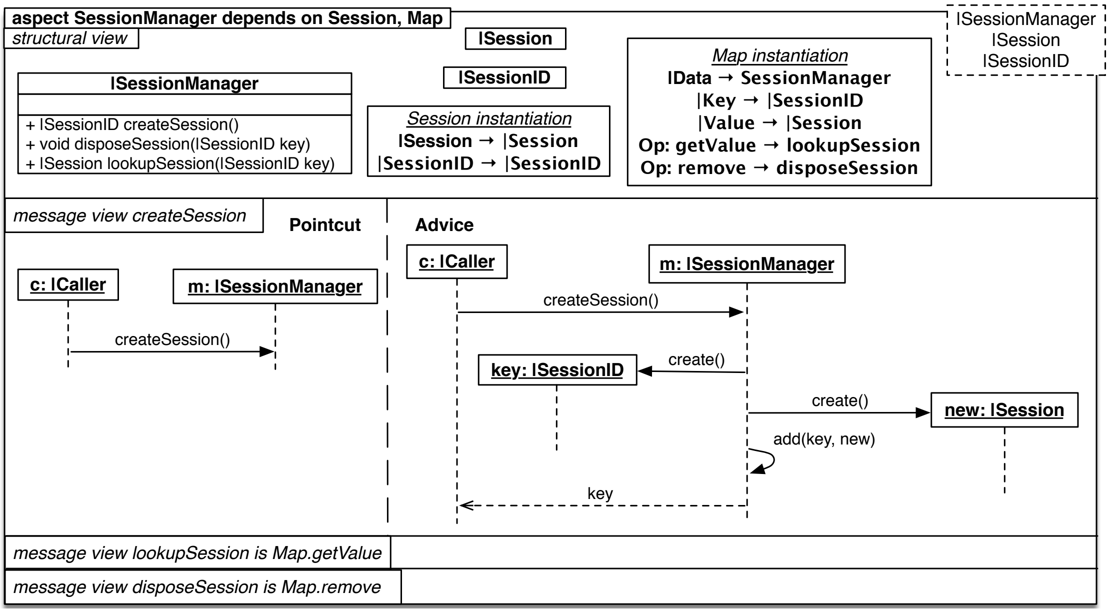


Figure 6.15: Aspect *SessionManager*

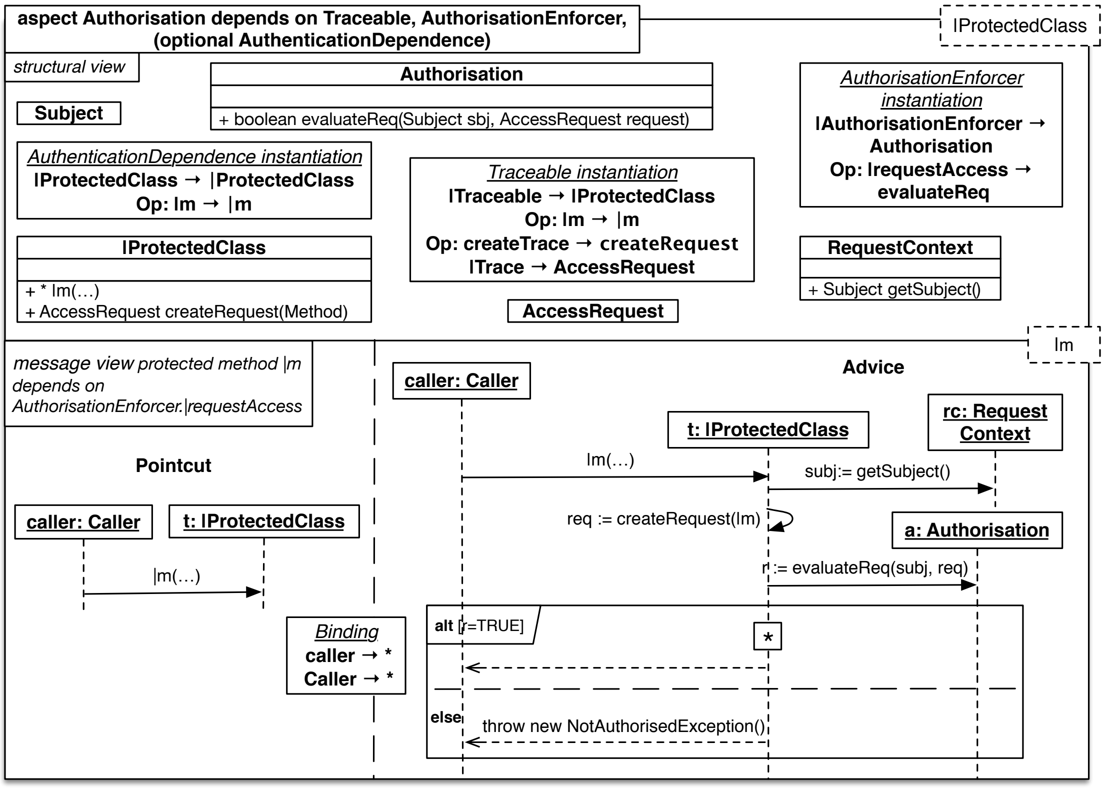

aspect Authorisation depends on Traceable, AuthorisationEnforcer, (optional AuthenticationDependence)
IProtectedClass
structural view
Subject
Authorisation
+ boolean evaluateReq(Subject sbj, AccessRequest request)
AuthorisationEnforcer instantiation
IAuthorisationEnforcer → Authorisation
Op: IrequestAccess → evaluateReq
AuthenticationDependence instantiation
IProtectedClass → |ProtectedClass
Op: Im → |m
Traceable instantiation
ITraceable → IProtectedClass
Op: Im → |m
Op: createTrace → createRequest
ITrace → AccessRequest
IProtectedClass
+ * Im(…)
+ AccessRequest createRequest(Method)
RequestContext
+ Subject getSubject()
AccessRequest
Im
message view protected method |m depends on AuthorisationEnforcer.|requestAccess
Pointcut
caller: Caller
t: IProtectedClass
|m(…)
Binding
caller → *
Caller → *
Advice
caller: Caller
t: IProtectedClass
rc: Request Context
Im(…)
subj:= getSubject()
req := createRequest(Im)
a: Authorisation
r := evaluateReq(subj, req)
alt [r=TRUE]
*
else
throw new NotAuthorisedException()

FIGURE 6.16: Aspect *Authorisation*

6.4.3 Authorisation Patterns

As can be seen in Fig. 6.6, the *Authorisation* feature (pattern) optionally depends on the *Authentication* feature (pattern) presented before. Fig. 6.16 shows the corresponding *Authorisation* RAM model. *Authorisation* can be employed together with *Authentication* if the optional *AuthenticationDependence* aspect is selected. *AuthenticationDependence* presented in Fig. 6.18 shows that authentication must be done before the method |m called. The *Authorisation* RAM model itself contains a parameterised *Authorisation* class with |evaluateReq function to evaluate any request AccessRequest to method |m of a protected resource |ProtectedClass. The AccessRequest is created with all necessary elements of a method call such as Method, AccessKind. Fig. 6.16 and Fig. 6.17 show that the generic *Traceable* aspect of RAM [122] is reused in *Authorisation* to create an AccessRequest. The |Subject requesting access is also obtained from the RequestContext. The |Subject and the AccessRequest objects are used for the access decision process managed by the *Authorisation*. If the request is granted, the protected method |m will be executed. Otherwise, an authorisation exception will be returned. The requestAccess method is refined further by the *AuthorisationEnforcer* pattern.

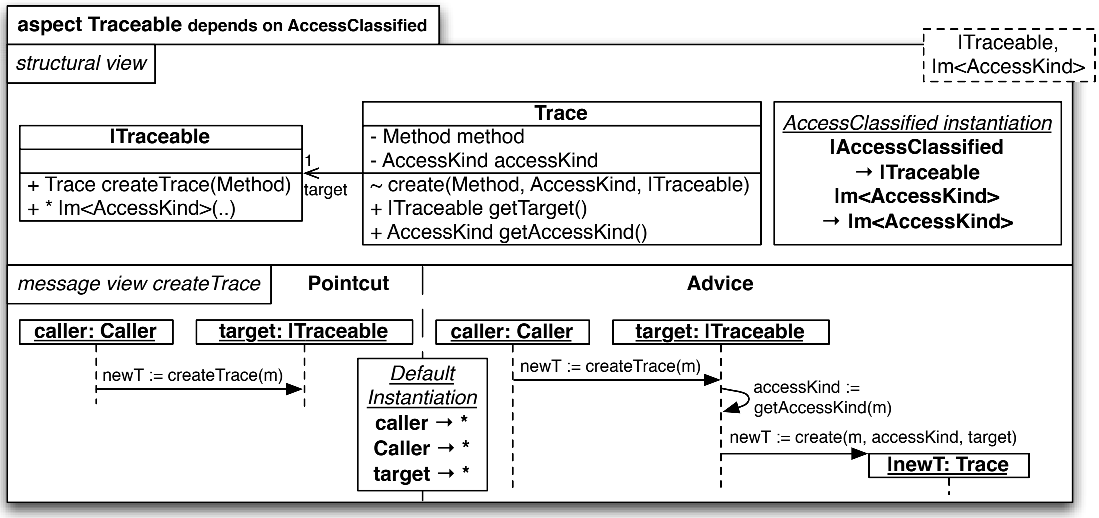


FIGURE 6.17: Aspect *Traceable* [122]

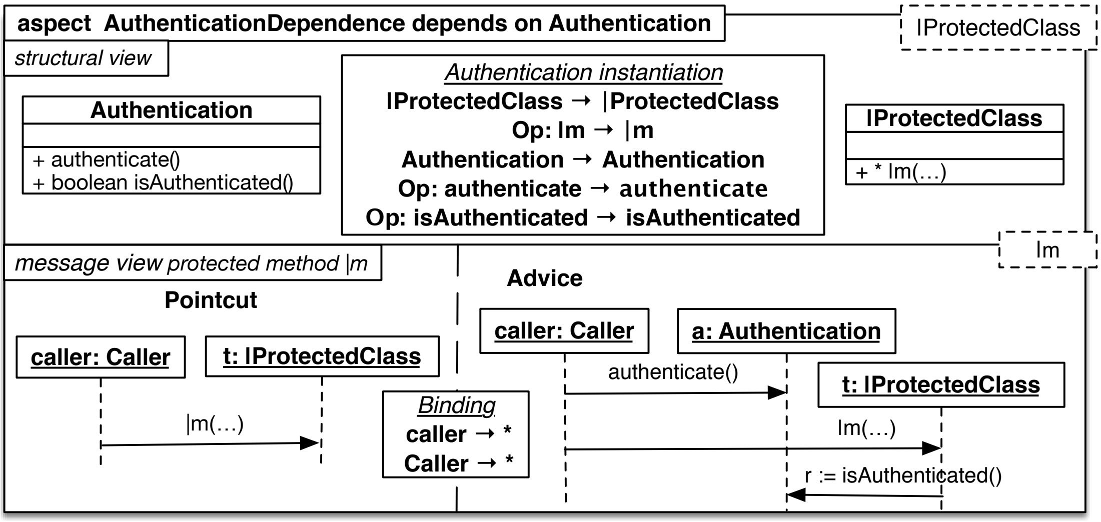

aspect AuthenticationDependence depends on Authentication
IProtectedClass
structural view
Authentication
+ authenticate()
+ boolean isAuthenticated()
Authentication instantiation
IProtectedClass → |ProtectedClass
Op: Im → |m
Authentication → Authentication
Op: authenticate → authenticate
Op: isAuthenticated → isAuthenticated
IProtectedClass
+ * Im(...)
Im
message view protected method |m
Pointcut
caller: Caller
t: IProtectedClass
|m(...)
Binding
caller → *
Caller → *
Advice
caller: Caller
a: Authentication
authenticate()
t: IProtectedClass
Im(...)
r := isAuthenticated()

FIGURE 6.18: Aspect *AuthenticationDependence* (renamed to L1 in Fig. 6.6)

The *AuthorisationEnforcer* pattern in Fig. 6.19 contains the *AuthorizationEnforcer* class that processes any request access together with other RAM models such as *PolicyBasedAC*, *ControlledAccessSession*. The *AuthorizationProvider* gets all the permissions of the request subject and comes up with a decision on the request to be returned by the *AuthorizationEnforcer*. The permissions of the request subject can be retrieved from a policy manager like *PolicyBasedAC* pattern or from the cached data of the subject in a secure session object (*ControlledAccessSession* pattern). These patterns are based on the patterns in [211, 72, 220].

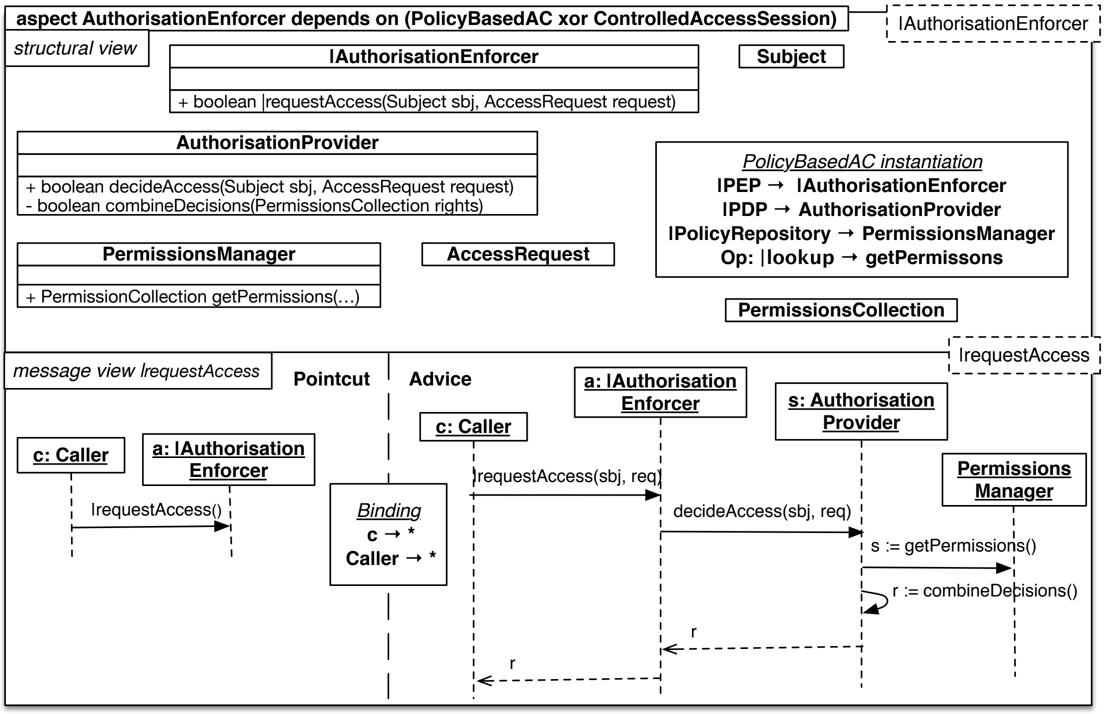

aspect AuthorisationEnforcer depends on (PolicyBasedAC xor ControlledAccessSession)
|AuthorisationEnforcer
structural view
|AuthorisationEnforcer
+ boolean |requestAccess(Subject sbj, AccessRequest request)
Subject
AuthorisationProvider
+ boolean decideAccess(Subject sbj, AccessRequest request)
- boolean combineDecisions(PermissionsCollection rights)
PolicyBasedAC instantiation
|PEP → |AuthorisationEnforcer
|PDP → AuthorisationProvider
|PolicyRepository → PermissionsManager
Op: |lookup → getPermissons
PermissionsManager
+ PermissionCollection getPermissions(...)
AccessRequest
PermissionsCollection
message view |requestAccess
|requestAccess
Pointcut
Advice
c: Caller
a: |Authorisation Enforcer
|requestAccess()
Binding
c → *
Caller → *
c: Caller
a: |Authorisation Enforcer
s: Authorisation Provider
Permissions Manager
|requestAccess(sbj, req)
decideAccess(sbj, req)
s := getPermissions()
r := combineDecisions()
r
r

FIGURE 6.19: Aspect *AuthorisationEnforcer*

6.4.4 Auditing (Accountability)

In critical systems, the ability to keeping tracks of who did what and when is very important. Security patterns for auditing can solve this accountability concern. The RAM models for auditing and their interrelations specified in Fig. 6.6 are based on the *Secure Logger*, *Audit Interceptor* patterns [220], and the *Security Logger and Auditor* pattern [72]. Two common patterns for auditing are *SecureLogger* and *AuditInterceptor* in which the latter depends on the former. Fig. 6.20 shows the structural view and message view of the *SecureLogger* pattern. The classes and methods being traced are mapped to the parameterised `Traced` class and method `|m`. Once the *Trace* object and the identity of *caller* are created, they are sent to the *SecureLogger* for being logged. How a *Trace* object can be created is specified by the generic *Traceable* aspect mentioned before. In Fig. 6.17, method `createTrace` returns a *Trace* object containing the *target* (*Traceable*) being traced, the *method* called, and the *accessKind*. Three kinds of access are specified in the *AccessClassified* aspect [122]: read, write, update. The *SecureLogger* creates the corresponding *LogEntry* object with the *Trace*, the identity of *caller*, and a time stamp. The *LogEntry* object should be processed into a *LogMessage* depending on the format required by the *LogManager* aspect. The *LogManager* is responsible for the actual serialisation of the log (using *LogFactory* aspect) to a secure storage (either using *SecureLogStore* pattern or *SecureDataLogger* pattern [220]).

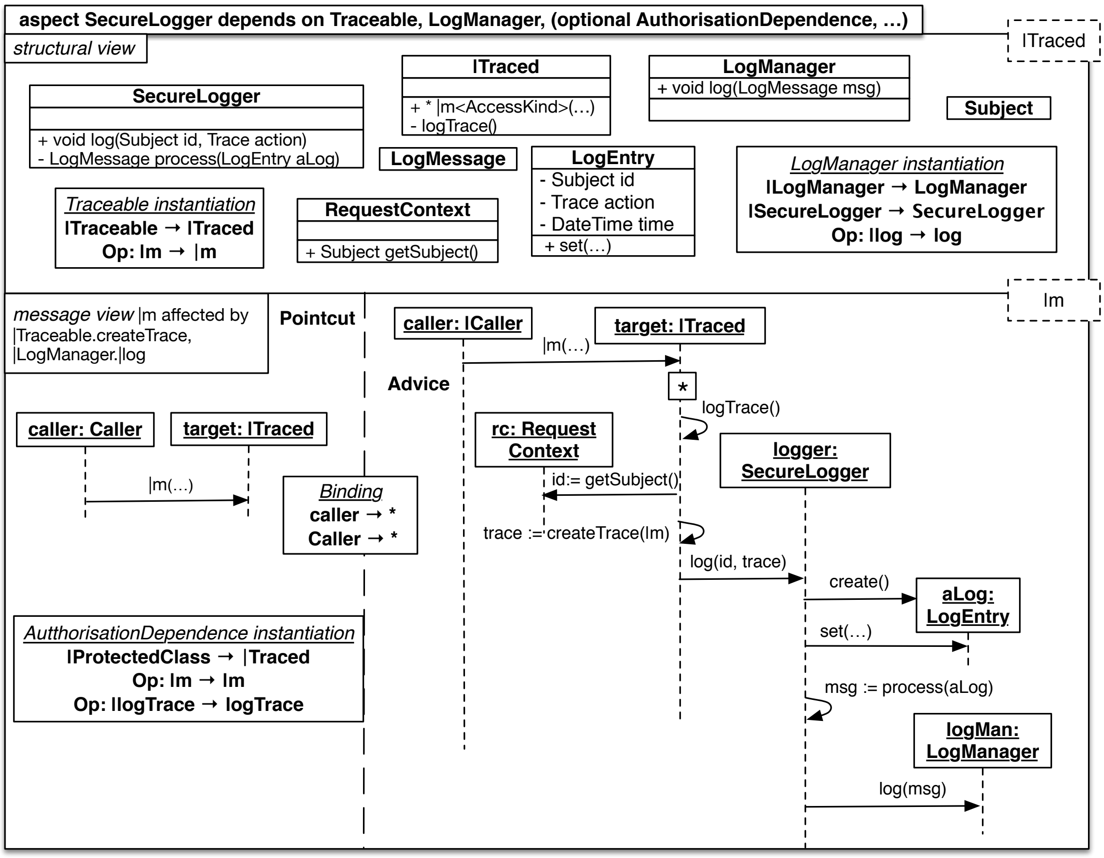


Figure 6.20: Aspect *SecureLogger*

SecureLogger in Fig. 6.20 is a generic logger that just simply logs a trace whenever the method `|m` is called. There is no specification on whether `|m` has been successfully authenticated and/or authorised, or actually executed. Different variations of when and how a trace is logged are provided in different logging strategy aspects. For example, aspect *AuthorisationDependence* in Fig. 6.21 specifies that a trace is logged only if the `caller` to `|m` has been authorised successfully and `|m` has been executed.

To summarise, we have presented some key security patterns of SoSPa and their dependencies to each others, and to other RAM models. Note that the order of dependencies on top of each RAM model matters to the order of weaving. How dependencies are woven into a RAM model are specified by instantiation mappings.

6.5 Evaluation and Discussion

6.5.1 Case Study and Results

By using SoSPa with the patterns selection and application process described in Section 6.3.2, multiple security concerns/misuse cases of CMS can be addressed/mitigated

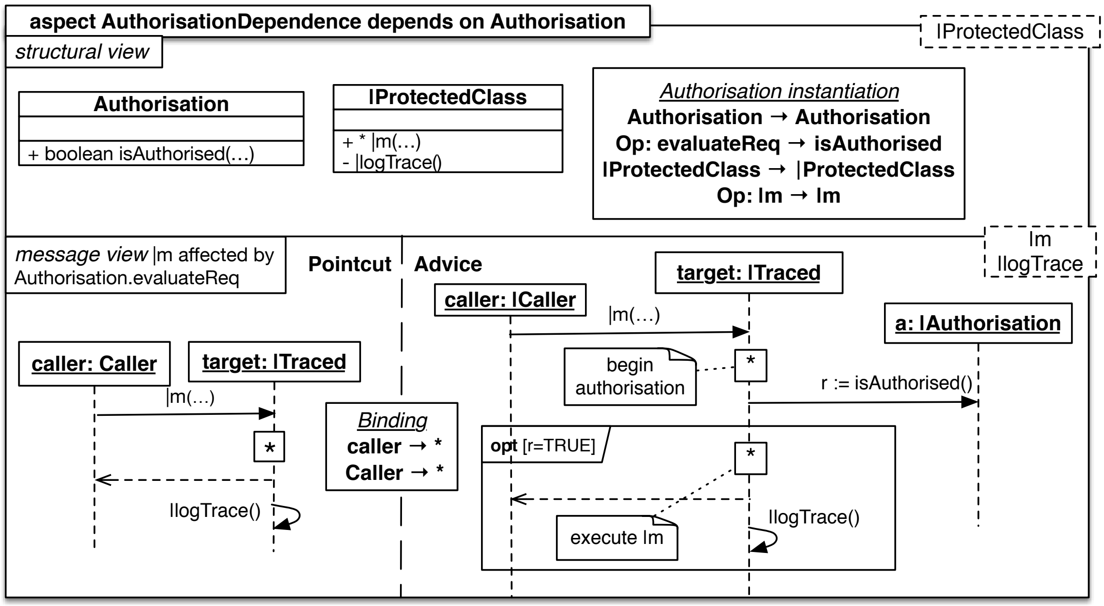


FIGURE 6.21: Aspect *AuthorisationDependence* (renamed to L2 in Fig. 6.6)

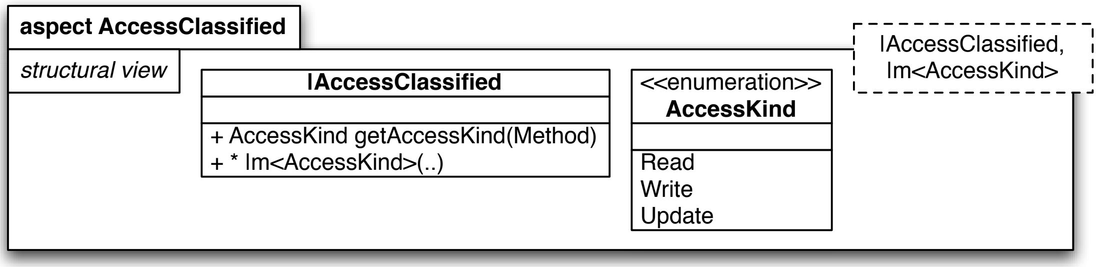


FIGURE 6.22: Aspect *AccessClassified* [122]

properly. We demonstrate the three main steps of the patterns selection and application process as follows.

Step 1 - Constructing security solutions for CMS from the security patterns in SoSP*a*: First, authenticating CMS users is the most fundamental security requirement of CMS as described in [125]. Fig. 6.6 shows that after selecting the *Authentication* feature, the designer can choose to employ *DirectAuthentication* or *JAASAuthentication*. *ThirdPartyAuthentication* is not a solution because CMS must use its own identity store for authenticating CMS users. Besides, *LimitedAttempts*, which specifies that an authentication request is blocked after some consecutive unsuccessful authentication attempts, could be already selected to partially mitigate [MUC-A1]. Assuming the designer selected *DirectAuthentication*, now he selects *Password* because using user password is a concrete requirement of CMS. Moreover, a strong user password solution must be employed to better mitigate [MUC-A1]. *PasswordComplexity* [211], *PasswordReset* [192] can be used together with *Password* pattern. One of the best solutions to mitigate [MUC-A1] is to use *Password* in combination with *HardwareToken* (OTP) [211]. To mitigate [MUC-A2], *SecuritySession* and also *SessionTimeout* need to be employed in the authentication solution. The *SessionTimeout* aspect makes sure that a session is invalidated (e.g. user is automatically logged off) if its inactive time exceeds a predefined threshold. When *SecuritySession* is selected, it means all the aspects that it depends on, e.g. mandatory *SessionManager*, are also selected and composed into it. This hierarchical aspects composition is applied to every feature in the feature model. Thus, all the selected features below *Authentication* are automatically woven into it, according to the order of dependencies specified on top of *Authentication*, and the model elements mappings defined in the corresponding RAM models. For example, once *LimitedAttempts* is selected, it is woven into *Authentication* first. In this way, a concrete authentication solution, namely *wovenAuthentication* RAM model has been built and ready to be integrated into CMS base design to fulfil its user authentication requirements and mitigate its potential misuse cases.

Similarly, a concrete authorisation solution can be built to mitigate the misuse cases [MUC-A3] and [MUC-A4]. Assuming *AuthorisatonEnforcer* with *PolicyBasedAccessControl* and *Role-Based Access Control* (*RBAC*) have been selected. All the selected RAM models for the authorisation solution such as *AuthorisatonEnforcer*, *PolicyBasedAccessControl* are woven into the *Authorisation* RAM model to create a *wovenAuthorisation* RAM model.

And so on, misuse cases [MUC-B1] can be mitigated by constructing a suitable *Auditing* solution, e.g. using *SecureLogger*. [MUC-B2] is also mitigated by using *SecureLogger* because either *SecureLogStore* or *SecureDataLogger* is employed to protect the logged data from being tampered. All the selected RAM models for *SecureLogger* are woven into the *SecureLogger* RAM model, resulting in a *wovenAuditing* RAM model.

To mitigate misuse cases [MUC-C1] and [MUC-C2], *SecureChannel* pattern [220], or *TLS* pattern as presented by [85], can be employed to secure transmitted data.

Step 2 - Mapping the security solutions to the CMS base design: For each security solution built in the previous step, its parameterised model elements can be mapped to the target elements in the CMS design to integrate the security solution into CMS. Note that the customisation interface of the top-most RAM model (e.g. *Authentication*) in the hierarchy of a security solution is also the customisation interface of that security solution (*wovenAuthentication*). Let us make the `createMission` function of CMS secure and its execution logged. We would come up with the following mappings:

```
wovenAuthentication.|ProtectedClass→CrisisManager
wovenAuthentication.|m→createMission
wovenAuthorisation.|ProtectedClass→CrisisManager
wovenAuthorisation.|m→createMission
```

```
wovenAuditing.|Traced→CrisisManager
wovenAuditing.|m→createMission
```

As we see, the constraints among the mappings have to be resolved. From the security requirements of CMS, the authorisation solution has to work with the existing (already built) authentication solution. Thus, the *AuthenticationDependence* feature (or L1 in Fig. 6.6) must also be selected for the authorisation solution `wovenAuthorisation`. That means `wovenAuthentication` is woven into *AuthenticationDependence*, and their woven RAM model is then woven into `wovenAuthorisation` to create a `wovenAuthenticationAuthorisation`. By doing so, the constraints among `wovenAuthentication` and `wovenAuthorisation` have been resolved. Similarly, to log a wrongly created rescue mission action of a *SuperObserver* in CMS as described in [MUC-B2], the *AuthorisationDependence* feature (or L2 in Fig. 6.6) is needed for `wovenAuditing`. By weaving `wovenAuthenticationAuthorisation` with *AuthorisationDependence* and then into `wovenAuditing` to create a `wovenAllSolutions` model, all the constraints have been resolved. After that, the instantiation directives for integrating all the security solutions into the CMS base design are straight forward:

```
wovenAllSolutions.|Traced → CrisisManager
wovenAllSolutions.|m → createMission
```

Step 3 - Weaving the security solutions into the CMS base design: This step can be automatically done by the RAM weaver once all the mappings and constraints have been specified in the previous step. Fig. 6.23 shows final woven model. For readability, we only display the key parts of the customised authentication, authorisation, and auditing solutions woven into the base model. The woven model shows that the `createMission` action now can only be executed by an authenticated user that is authorised to execute this task, and a trace of this action is securely logged. Of course, a formal analysis of the woven model against attack models could show a formal proof that the woven model is resilient to different attacks. Constructing attack models in a similar way as security solution models and then employing formal analysis techniques could be a good direction for future work.

6.5.2 Discussion

Our work raises an important question related to the abstraction level proposed by SoSPa. More specifically, the question is how can we guarantee that the level of details is sufficient? Or how to guarantee that there is no need to develop new RAM models or security patterns? Our answer is that the required level of details strongly depends on the expected use of SoSPa. Obviously, if SoSPa is used to generate code that can

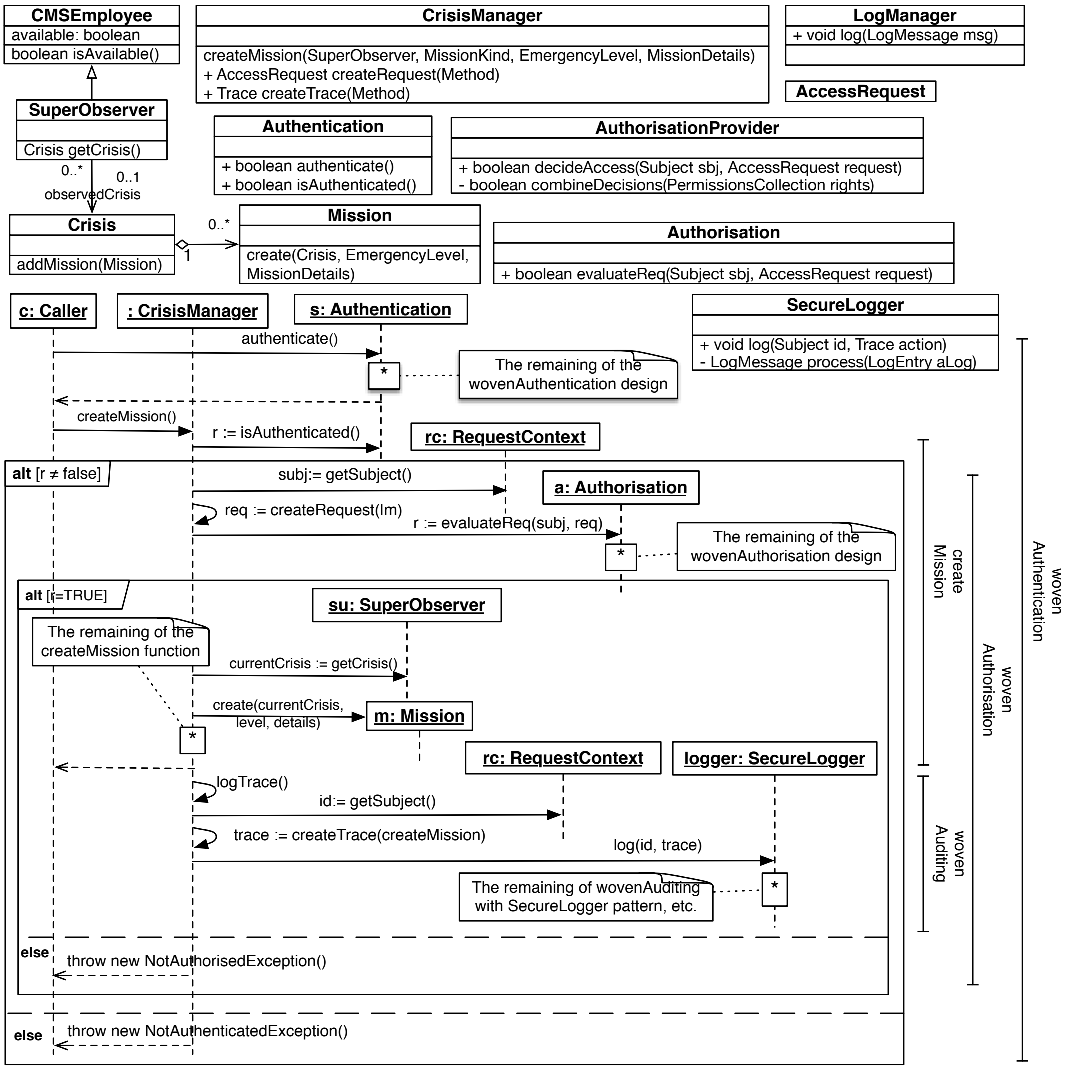

CMSEmployee
available: boolean
boolean isAvailable()
CrisisManager
createMission(SuperObserver, MissionKind, EmergencyLevel, MissionDetails)
+ AccessRequest createRequest(Method)
+ Trace createTrace(Method)
LogManager
+ void log(LogMessage msg)
AccessRequest
SuperObserver
Crisis getCrisis()
0..*
0..1
observedCrisis
Authentication
+ boolean authenticate()
+ boolean isAuthenticated()
AuthorisationProvider
+ boolean decideAccess(Subject sbj, AccessRequest request)
- boolean combineDecisions(PermissionsCollection rights)
Crisis
addMission(Mission)
0..*
1
Mission
create(Crisis, EmergencyLevel, MissionDetails)
Authorisation
+ boolean evaluateReq(Subject sbj, AccessRequest request)
c: Caller
: CrisisManager
s: Authentication
SecureLogger
+ void log(Subject id, Trace action)
- LogMessage process(LogEntry aLog)
authenticate()
*
The remaining of the wovenAuthentication design
createMission()
r := isAuthenticated()
rc: RequestContext
alt [r ≠ false]
subj:= getSubject()
a: Authorisation
req := createRequest(lm)
r := evaluateReq(subj, req)
*
The remaining of the wovenAuthorisation design
alt [r=TRUE]
su: SuperObserver
The remaining of the createMission function
currentCrisis := getCrisis()
create(currentCrisis, level, details)
m: Mission
*
rc: RequestContext
logger: SecureLogger
logTrace()
id:= getSubject()
trace := createTrace(createMission)
log(id, trace)
The remaining of wovenAuditing with SecureLogger pattern, etc.
*
else
throw new NotAuthorisedException()
else
throw new NotAuthenticatedException()
create Mission
woven Auditing
woven Authorisation
woven Authentication

FIGURE 6.23: Automatically Woven Secure CMS Model

run, the level of details should be very high. If the goal is to generate a code skeleton of a secure system and test cases, the level could be lower. The level of details of RAM models in SoSPa is not high enough for full code generation, but high enough for specifying all the important semantics of security patterns and their interrelations. That means a code skeleton of a secure system can be generated that preserves the important semantics of the employed security patterns and their interrelations.

Each security design pattern can also be associated with the side effects of its adoption on other quality properties, e.g. performance, usability. An impact model of the security design patterns for each quality property can be built as discussed in [13]. Impact models are useful for analysis of the trade-off among alternatives which leads to a thoughtful decision on systematically selecting the right security design patterns for the job. On the other hand, attack models can also be specified using SoSPa-MM. These attack models are associated to the security concerns, and can be woven into the system model to generate misuse models for formal analysis of security, or generate test cases for testing. In this chapter, we only focus on interrelations among security patterns, not yet considering the relations to other quality properties and attack models. The types of interrelations in this chapter are aligned with the five types of inter-pattern relations presented in [239].

A second question is related to the completeness and extensibility of SoSPa. As previously discussed, the set of security patterns mentioned in this chapter is not yet exhaustive. We have nonetheless considered the most illustrative for the purpose of our work. However, an interesting feature of SoSPa is that it is fully extensible. As a result, if a user realises that some details or some security patterns are missing, he can extend it. This is eased by the use of RAM and the explicitness of the relationships among the security patterns. More about SoSPa can be found in Appendix D.

In a final note, we recall that to the best of our knowledge, SoSPa is the first attempt to provide an extensive set of concrete design models of security patterns that can be integrated systematically. We provide RAM models that users can explore according to several dimensions: not only from high to low level of details, but also in the variability perspective traversing a large number of possible security solutions.

6.6 Related Work

There are other aspect-oriented security design methodologies for modelling security aspects using patterns, and then weaving them into functional models. Georg et al. [85] propose a methodology that allows not only security mechanisms but also attacks to be modelled as aspect models. The attacks models can be composed with the primary model of the application to obtain the misuse model. The authors then use the Alloy Analyser[1] to reason about the misuse model and the security-treated model. Mouheb et al. [171] develop a UML profile that allows specifying security mechanisms as aspect models. The aspect models often go together with their integration specification to be woven automatically into UML design models. Their approach allows security aspects to be woven automatically into UML design models (class diagrams, state machine diagrams, sequence diagrams, and activity diagrams) [171]. In [172], the authors present a full security hardening approach, from design to implementation. Abramov et al. [1] propose an MDS framework for integrating access control policies into database development. At the pre-development stage, organisational policies are specified as security patterns. Then, the specified security patterns guide the definition and implementation of the security requirements which are defined as part of the data model. The database code

[1] http://alloy.mit.edu

can be generated automatically after the correct implementation of the security patterns has been verified at the design stage. Bouaziz et al. [46] introduce a security pattern integration process for component-based models. With this process, security patterns can be integrated in the whole development process, from UML component modelling until aspect code generation. Recently, Horcas et al. [98] propose a hybrid AOSD and MDE approach for automatically weaving a customised security model into the base application model. By using the Common Variability Language (CVL) and ATL, different security concerns can be woven into the base application in an aspect-oriented way, according to weaving patterns. However, dependencies between the security aspects and their application orders have not been taken into account. In general, all these approaches have not considered security aspects as a coherent system in which their interrelations and constraints are taken into account.

As using cloud services provided by cloud providers is getting more popular, [163] recently propose an enterprise security pattern for securing Software as a Service. The security solution provided by the pattern can be driven by making design decisions whilst performing the transformation between the solution models. Specifically, from a Computation Independent Model (CIM), different PIMs can be derived based on different design decisions with security patterns. Those PIMs are transformed into PSMs which are then transformed into Product Dependent Models PDMs.

Shiroma et al. [213] propose an approach to leverage model transformations for specifying and implementing application procedures of security patterns, including inter-pattern dependencies. The inter-pattern dependencies here mean the order of consecutive applications of security patterns by performing consecutive transformations. The order is specified by defining the output of the previous model transformation as a precondition in the subsequent model transformation. In fact, the approach only presents the dependencies in terms of the order of application of security patterns. There are many other important interrelations that one must consider such as conflicts, benefits, and alternatives among security patterns. Their approach is only able to deal with 8 per 27 security patterns in [211] because the other 19 patterns do not have structures described. With SoSPa, we can address an extensive, extensible system of security patterns. All key interrelations among security patterns are considered, not only the order of patterns application.

Our MDS approach based on SoSPa initially explored in the position paper [181] is inspired by [13]. Alam et al. [13] propose an approach based on RAM for designing software with *concern* as the main unit of reuse. They show how their ideas can be realised by using an example of low-level design concern, i.e. the *Association* concern. The adoption of their approach to high-level concerns like security has not been dealt with yet. In this chapter, we realised the ideas in [181] by developing a system of RAM models specifying multiple security patterns and their interrelations to form SoSPa. We extend the concept of using variation interface in [13] for specifying the interrelations among security patterns. Our work results in a unified *System of Security design Patterns*. With SoSPa, for the first time, abstract security patterns can be specified, refined as detailed designs with concrete semantics. Also for the first time, interrelations among patterns can be concretely specified in the detailed designs, thanks to RAM. Furthermore, we plan to extend the idea of using impact model in [13] for specifying the constraints of security patterns with other quality properties like performance, usability.

6.7 Conclusions

This chapter has presented an MDS approach based on a *System of Security design Patterns* to systematically guide and automate the application of multiple security patterns in secure systems development. Our *System of Security design Patterns* is specified by using an extended meta-model of RAM, namely SoSPa. Based on SoSPa, security patterns are collected, specified as reusable aspect models to form a coherent system of them. SoSPa is part of a full *Model-Driven Software Development* framework. Our framework allows systematically selecting security design patterns, constructing security solutions, and automatically composing them with a target system design. We evaluated our approach by leveraging the *System of Security design Patterns* to design the security of crisis management systems. The result shows that multiple security concerns of the case study have been addressed. More importantly, the different security solutions are thoughtfully selected and systematically integrated.

Chapter 7

Conclusions and Future Work

Contents

This chapter concludes the thesis by summarising its main content in Section 7.1.1 and revisiting its main objectives and contributions in Section 7.1.2. Finally, in Section 7.2 some directions for future work based on this thesis are suggested.

7.1 Conclusions

7.1.1 Summary

Software systems must be secure especially when we are living in a world that becoming more and more digitalised. The digital world cannot be made secure by only enhancing network security and other perimeter solutions, but essentially also by building better, secure software systems [149]. Security software engineering is playing very important role but in fact facing many tough challenges. These modern challenges could be solved by introducing innovative, sound methodologies for engineering secure software systems. MDS is the specialised MDE approach for engineering secure software systems. MDS aims at proposing sound model-driven methodologies that could solve the challenges of security software engineering for developing secure software systems. However, MDS is not mature yet with some existing significant limitations and open issues. In this thesis, three main current open issues in the state of the art of MDS research have been pointed out from an extensive systematic review of MDS. The following open issues have been addressed in this thesis: no systematic MDS engineering of multiple security concerns, the underuse of AOM in MDS, and the lack of tool chains for supporting a full MDS development cycle. As a major outcome of this PhD work, our full MDS framework, called MDS-MoRe, could enable MDS practitioners: 1) to address multiple security concerns together systematically; 2) to leverage integrated MDS techniques in most of the main stages of a development process: from modelling till testing; and 3) to enhance the modularity and reusability with AOM and SOC in their MDS development process.

7.1.2 Revisited Contributions

This thesis proposes solutions to three main open issues in the current state of the art of MDS research. We revisit the original contributions described in Chapter 1. The contributions are explicitly used to answer all the research questions, i.e. *RQ0, RQ1.1, RQ1.2, RQ1.3, RQ2.1, RQ2.2, and RQ2.3.*

In Chapter 3, the current state of the art of MDS research has been revealed by our extensive systematic review. The results show that most MDS approaches focus on *authorisation* and *confidentiality* while only few publications address further security concerns like *integrity*, *availability*, and *authentication*. Moreover, security concerns are often dealt with separately. Very few MDS approaches tackle multiple security concerns systematically. Besides, most of the approaches try to separate security concerns from core business logic, but only few weave security aspects into primary models. In our review, we have identified five principal MDS studies which seem more mature than the others. On the other hand, there are also other emerging/less common MDS approaches that we have found out. Another important finding comes from the trend analysis on the key artefacts of MDS over more than a decade. All these results have answered the ***RQ0****: What are the current limitations and open issues in the state of the art of* MDS *research?*, and given directions for our research work presented in the following chapters, especially to tackle three of the main open issues of MDS research.

In Chapter 4, we show an overview of our two systematic MDS approaches for tackling the three open issues above. The former focuses on tackling a specific, complex security concern, i.e. access control management with advanced delegation features, from domain-specific modelling, composing, implementing, and testing as presented in Chapter 5. This could provide the basis for developing a tool chain for supporting modelling, composing, implementing, and testing in an MDS development process. The latter consists of a unified *System of Security design Patterns* for dealing with multiple security concerns systematically. This work also focuses on promoting AOM in MDS as presented in Chapter 6.

In Chapter 5, a complete model-driven framework has been proposed to enable dynamic enforcement of delegation and access control policies that allows the automatic configuration of the system according to the changes in delegation/access control rules. Moreover, we have presented the use of mutation analysis to test the delegation policy enforcement. Thus, our MDS framework has been proposed from modelling till testing which could be the basis for developing a tool chain. The results of Chapter 5 can be used to answer the ***RQ1.3****: How to build a tool chain which is based on highly integrated* MDE *techniques covering all the main stages of an* MDS *development cycle?* and ***RQ2.3****: How can model-based security testing techniques be applied to facilitate the validation of the resulting secure systems?*

In Chapter 6, we have presented an MDS approach based on a *System of Security design Patterns* (SoSPa) to systematically guide and automate the application of multiple security patterns in secure systems development. SoSPa is part of a full *Model-Driven Software Development* framework. Our framework allows systematically selecting security design patterns, constructing security solutions, and automatically composing them with a target system design. Thus, Chapter 6 has addressed the ***RQ1.1****: How to address multiple security concerns more systematically?*, ***RQ1.2****: How to leverage* AOM *techniques to better enhance separation-of-concern in the* MDS *development process?*, and ***RQ2.1****: How can modelling techniques be used for specifying multiple security concerns?*

In general, for answering the ***RQ2.2****: How can model composition techniques be employed for composing the security models with the target system model?*, we have shown that ad-hoc model transformations can be used for composing models as presented in Chapter 5. Moreover, in Chapter 6 we have shown that an existing model weaver like RAM weaver can be used for composing models in the SoSPa approach.

7.2 Future Work

There are at least three main directions for future work based on three main contributions of this thesis. First, an up-to-date extensive systematic review of MDS can be conducted by inheriting the results of our review presented in Chapter 3. Second, tool chains for supporting all the main stages of the MDS development methodologies presented in Chapters 5 and 6 can be developed. Moreover, the two main methodologies presented in parallel in Chapter 4 can be combined in a hybrid approach in which the link between them is specified. Third, there are potential future work for extending the work on SoSPa such as developing the impact models, the security attack patterns, and the verification and validation methodologies.

7.2.1 For An Up-to-date Systematic Review of MDS

Our SLR protocol and the list of finally selected MDS papers could be used as the base for a follow-up SLR of MDS in the future. A reviewer would need to check again the citation criterion for those primary MDS papers using up-to-date citation numbers on Google Scholar. After obtaining a subset of MDS papers from the original set, a forward snowballing on that subset should be conducted. Only forward snowballing is required in this step because new, extra MDS papers will only be found in this way. After reviewing and selecting a new set of MDS papers from the forward snowballing step, the full snowballing process can be operated on the new set. The final newly found MDS papers after snowballing will be included into the base subset to form a new final set for data extraction, synthesis, and analysis.

7.2.2 For Developing MDS Tool Chains

An expected future work is developing complete MDS tool chain(s) based on the MDS-MoRe framework. A tool chain would realise all the potential improvement for the productivity and quality in engineering modern secure systems.

In Chapter 5, we have presented an MDS approach from modelling till testing. This approach can provide the basis for developing a tool chain for supporting the model-driven development of secure software systems. In our prototype, we have developed a DSL for specifying complex delegation features in access control management. This is the first but fundamental step to build a tool for supporting security officers to define their delegation policies. In the next step, model transformations to transform and integrate security policies to the target system model have been implemented in Kemerta, where as code generation has been implemented using XPand. We have not built any prototype yet for supporting the mutation testing approach but the main ideas are 1) a tool for generating mutants based on mutation operators we proposed, and 2) another tool for generating test cases from the security model.

One more point to add is that our work presented in Chapter 5 only focuses on the delegation of rights. Further work could be dedicated to the delegation of obligations and the support for usage control [206]. Usage control is called the next generation of access control with more flexible access management mechanisms that we would adopt our current approach for. We have not dealt with this idea yet but suggest it for future work. Moreover, revocation mechanism in our current approach has not been completely taken into account, i.e. without options of strong/weak revocation.

In Chapter 6, we have mentioned our MDS process with three main stages: Security threats identification & analysis; Security design patterns selection & application; and Verification & validation of security patterns application. Another tool chain for supporting all three main stages could be developed as well. The main part of Security design patterns selection & application can be supported by RAM tool. We could extend the tool to support the selection of security design patterns based on the feature model of SoSPa and the library of security design patterns in SoSPa. For the first stage and second stage, tools support could be built based on the ideas for extending our work in the next section.

7.2.3 For Extending SoSPa

Some suggestions for future work in this direction include developing the impact models, making use of the security attack patterns, and integrating the verification and validation methodologies for SoSPa.

The first foreseen future work can be extending the impact models in [13] for specifying the side-effects of security patterns regarding other quality properties like performance, usability. Another point is incorporating the attack models to generate misuse models for formal analysis of security, or generating test cases for security testing. We could leverage the formal analysis approach proposed by Georg et al. [85] where for some security concerns, e.g. authentication, could be formally verified. For the security concerns that leveraging formal analysis for verification is not yet a feasible solution, we could adopt testing techniques. One direction could be a test-driven process for validating security design patterns application. We may have some inspiration from the approach proposed by Kobashi et al. [133] in which test templates could be introduced into the security design patterns for later validation of their application.

7.2.4 Towards Realising The Full MDS-MoRe Approach

The two main methodologies presented in parallel in Chapter 4 can be combined in a hybrid approach where the link between them is specified. The main idea is to use the pattern-driven MDS approach in Chapter 6 for systematically tackling multiple security concerns. Based on that, for some complex security concerns, the corresponding security patterns could be extended/integrated with some DSLs to better specify the complexity of security concerns, e.g. advanced delegation features. The most advanced form of SoSPa could be a system of security patterns based on a system of DSLs in which each security solution can be built on specific DSL (s). Of course, there is still much work to do to realise this idea.

Appendix A

An Informal Introduction to MDS

To promote research and present research activities being taken place in Luxembourg to broad audience in Luxembourg's society, many events such as LuxDoc[1] Science Slam, FNR[2] Researchers' Day, Pop-Up Science, Science Cafe have been organised annually. In the context of LuxDoc Science Slam, Luxembourg Researchers' Day, Pop-Up Science, MDS and our work on the research project funded by FNR, namely MITER[3], have been presented to a broad audience in Luxembourg. The main idea of such events is to present research activities in a creative, artistic way to attract a broad audience with different backgrounds. It is challenging to introduce MDS in an interesting way to a broad audience with different backgrounds. To address the challenge, MDS and our MITER project have been presented in the form of a multilingual, scientific poem, namely "A Multilingual, Scientific Poem on Model-Driven Security". The poem is structured as a classical scientific paper with four main sections: introduction, approach, evaluation, and conclusion. In the poem, English is mainly used, but some words in Luxembourgish, French, and German are also used to give some flavour of Luxembourg, a multilingual country. Moreover, the poem should not be simply read, but sung in a Vietnamese karaoke singing style! There are two main versions of the poem which are slightly different in the first section and how it should be sung. The first version of this poem won the second place at the LuxDoc Science Slam 2013[4]. The second version of this poem won the first place at the LuxDoc Science Slam 2014[5]. The remainder of this appendix consists of four main sections of the poem.

[1] www.luxdoc.org

[2] www.fnr.lu

[3] https://sites.google.com/site/jacqueskleinwebpage/grants/miter

[4] The full video of this event can be found here: https://www.youtube.com/watch?v=rBipinbHsyo

[5] A short interview in German: http://www.journal.lu/article/mit-karaoke-zum-sieg/

A.1 Introduction

The more digital world[6] in which we live, the more crucial IT security is becoming. However, traditional methods for the development of secure systems are getting more inefficient to ensure security [182]. Not many days pass without new headlines on the newspapers about IT security vulnerabilities, security attacks, digital data leaks, and so on [144]. In this context, Luxembourg with its significant banking industry (e.g. CETREL[7], SPUERKEESS[8]), e-government, and data centres has made IT security an important task. On the other hand, the increasing complexity of systems, quickly changing security threats, and time pressure all require an innovative sound methodology for secure systems development (see[9] Listing A.1).

```
 1 Securite? Why to bother?
   Internet banking ”tout le monde”!
 3 Smart phone is now a ”must”.
   Face–Goog know you that’s for sure ;)
 5
   Moien Letzebuerg, why Secherhect needed?
 7 What if CETREL lost your info?
   Credit cards would have gone!
 9 Millions of Euros to be lost
   SPUERKEESS knows all the cost
11 Better Security Systems must be enforced!

13 Why not yet that secure?
   Securite threats change so quick
15 Business requirements change so quick
   Time pressure on the development process
17 Time to call for innovative sound methods
   For Secure Systems to be developed
```

Listing A.1: Section “Introduction” of the poem

In the second version of this poem, the introduction section is slightly different and should be sung differently (see[10] Listing A.2).

```
   Security? Why to bother?
 2 Internet banking ”tout le monde”!
   Smart phone is now, la la la, a ”must”.
 4 Face–Goog know you that’s for sure ;)
   Amazon, eBay there you shop
```

[6] tout le monde=all over the world

[7] CETREL is a credit card company in Luxembourg

[8] SPUERKEESS, a.k.a. BCEE is the national bank of Luxembourg

[9] Lines 1-11 (except line 6) of Listing A.1 should be sung as the National Anthem (song) of Vietnam “Tien Quan Ca”.

[10] Lines 1-14 of Listing A.2 should be sung as the Vietnamese traditional folk song “Beo dat may troi”.

```
Cloud shouldn't let you down...

Moien Letzebuerg, what to bother?
Why Secherhect needed?
What if CETREL lost your information?
 Credit cards would have gone!
 Millions of Euros to be lost
SPUERKEESS knows all the cost
 Security must be enforced!

Why not yet that secure?
 Securite threats change so quick
 Business requirements change so quick
 Time pressure on the development process
 Time to call for innovative sound methods
 For Secure Systems to be developed
```

Listing A.2: Section “Introduction” v2.0 of the poem

A.2 Approach

Model-Driven Engineering (MDE) has been considered by some researcher [38] as a solution to the handling of complex and evolving software systems. As a specialisation of MDE, *Model-Driven Security* (MDS) provides means to tackle the complexity, and increase the productivity in modern secure systems development [190]. Roughly speaking, security concerns of a system can be modelled by security experts using tailored Domain Specific Languages. Separately is the business logic (base system) modelled by the system designers. The security models and the system models can be composed together in order to produce the secure system model [190]. The secure system model can then be used for (partial) code generation, including (configured) security infrastructures [184]. Along the way of this process, model checking, simulation, and model-based security testing techniques can be employed to verify and validate the resulting secure systems [191]. Listing A.3 shows the core aspects of MDS approach[11].

```
1 How about the MDS method?
   Modeling the security concerns for the good
3 By the hands of the security experts
   In a way that they must have understood
5 On the other hand, the business logic
   Taken care by the business modelers
```

[11] Lines 1-18 of Listing A.3 should be sung as the song “Mot coi di ve”. Lines 5-15 of Listing A.4 should be sung as “Noi vong tay lon”. For more information: http://en.wikipedia.org/w/index.php?title=Trinh_Cong_Son

```
Those to be auto−composed together
Secure systems just one−click further!

Model checking and testing to be leveraged
All the security concerns can be simulated
To make sure the secure systems accomplished!
For both security concerns and business logic :)

What so good is also for the code
The language that computers like the most!
Secure code is generated at no cost ;)
From the secure models that we got!
```

Listing A.3: Section "Approach" of the poem

A.3 Evaluation

In general, MDS has many values to become the most promising methodology for modern secure systems development. By leveraging model transformations and code generation, most of the development process can be automated which is very productive and less error prone.

Moreover, model-based verification and validation techniques can be employed for ensuring the correctness and security of the resulting systems. Some specific problem areas such as smart-cards or cryptographic protocols are applicable for formal verification methods. If formal verification is still unfeasible for larger systems due to increased complexity and dependencies, we can apply (model-based) security testing. Listing A.4 addresses these points[12].

```
   Why MDS can be proved?
 2 such as a sound method
   for the secure systems to be developed.
 4
   First, at no cost to generate secure code.
 6 Time pressure does not need a thought.
   Productivity surely that we got.
 8 New threats can be dealt till the source
   Just some clicks away from models to code
10
   Second, quality of the product
12 would it be secure and good?
   Yes, of course we can prove
14 That secure system models are good
   By model checking and testing that we did
```

[12] Lines 18-21 of Listing A.4, and Listing A.5 should be sung as the kid song "Chiec den ong sao"

```

And last but not least,
The MDS systems are adaptive
All the new threats to be dealt with
At run time they can be treated!
And now we reach to the conclusion.
```

Listing A.4: Section “Evaluation” of the poem

A.4 Conclusion

```
In the more digital world that we live
The more secure-systems we all need
MDS shows us what are great
For the secure systems to be fit :)
```

Listing A.5: Section ”Conclusion” of the poem

This chapter has presented a multilingual, scientific poem on *Model-Driven Security* (MDS). The poem is structured as a scientific paper with four main sections: introduction, approach, evaluation, and conclusion. The poem has shown us 1) why IT security is getting more crucial (especially in Luxembourg); 2) how MDS could help building secure IT systems; 3) what aspects show that MDS is indeed a sound method; and finally 4) summary of those key points. Listing A.5 concludes that MDS can help to build secure IT systems in the nowadays digital world. The main language used in the poem is English, but some words in Luxembourgish, French, and German are also used to give some flavour of Luxembourg, a multilingual country in where our research on MDS has been being conducted. Moreover, the poem is not read but should be sung in a Vietnamese karaoke singing style, including the rhythms of the National Anthem (song) of Vietnam and three other Vietnamese songs!

Acknowledgments

Our work on MDS is supported by the Fonds National de la Recherche (FNR) Luxembourg, under the project C10/IS783852/ MITER. Special thanks also go to the organisers of the yearly LuxDoc Science Slam, the Researchers’ Days and Pop Up Science in Luxembourg.
Thanks!, *Dankeschon!*, *Merci!*, *Villmols merci!*, *Cam on!*

Appendix B

Advances in MDS: A Summary

This appendix summarises a book chapter where we describe and evaluate a set of representative and influential *Model-Driven Security* approaches in the literature [144]. This book chapter [144] and the systematic review presented in Chapter 3 can complement well for each other. The former provides an in-depth analysis on some specific, representative MDS approaches. The latter systematically presents a detailed overview on the key artefacts of every MDS approach and the whole MDS research so far. In this appendix, a short overview of the book chapter and its main conclusions are given. Readers can find more details in [144].

B.1 Overview

The book chapter is organised with the following main sections. First, a set of characteristics of MDS is identified and described in order to form a taxonomy for further evaluation of MDS approaches. Then, we evaluate a few selected MDS approaches against our taxonomy. The discussion section provides a table comparing the evaluated approaches, and discusses some open issues and validity threats for MDS. Finally, the related work is addressed and potential challenges for MDS are raised to conclude the chapter.

In order to propose a clear vision of what is MDS, we first introduced the main concepts on which MDS is based on. In particular, we focused on the notions of MDE, such as metamodels, model transformations, etc, but also on the notion of separation of concerns or separation of views. We have then proposed a detailed taxonomy consisting of the main concepts and elements of MDS. Based on our taxonomy we have described, summarised, evaluated and discussed five well-known MDS approaches: UMLsec, SecureUML, Sectet, ModelSec and SecureMDD. We pointed out some current limitations of MDS approaches, and sketched some relevant open issues. Overall, this chapter provides a broad view of the field and is intended as an introduction to MDS for students, researchers or practitioners.

B.2 Main Conclusions

This work allows us to provide insights about future directions for MDS research and industrial practice. A primary challenge is to reach a better level of maturity: this of course requires building tools, but also conducting more systematic industrial experimentation. The latter is obviously difficult in such a critical software application domain. However, recent progress in MDE in theories and tooling, as well as the continuous interest from industry in modelling, may directly be of benefit to the development of MDS.

We noticed that most of the existing MDS approaches implement separation of security concerns from the business logic (SOC), even if for the approaches we surveyed this principle cannot be considered as following the AOM paradigm. By leveraging AOM, security concerns can be specified as *aspect* models that can be *woven* into the primary (business logic) models. The AOM paradigm could thus be used to enhance the modularity of the security-critical systems and the reusability of the security models.

MDS has to deal with the business complexity, but also with the additional complexity of security concerns which are multiple in nature. An intuitive solution to this variety is for the software development methodology to reflect the heterogeneous nature of such systems with the goal of making their design simpler. From our evaluation we can deduce that it is difficult to develop a general-purpose DSL intended to model all security concerns simultaneously. This is because different security properties may require different interactions with the business models that are too complex to be modelled by a human using a single DSL. For easing the modelling task and allowing better verification possibilities, an interesting possibility is that each security concern is modelled using a specifically tailored DSL.

However, spreading security concerns over several models raises several crucial challenges in our opinion: *First*, it hinders the understanding of the overall system's security by the experts since they need to deal with several models simultaneously. This drawback can however be balanced by the narrowed focus of those models. *Second*, it requires powerful composition operators for creating models that amalgamate all security aspects. This is crucial for later phases: whereas separate security models make it possible to analyse security properties independently, the enforcement of those policies in the business part of the systems and code generation requires merging all security aspects before reaching platform code. *Third*, it complicates keeping all security models synchronised over common information and poses additional challenges when tracking integrated verification results back to security information that is distributed over several models.

Towards addressing some of the challenges above, in Chapter 6 we propose an MDS approach based on a unified *System of Security design Patterns* that can support systematically dealing with multiple security concerns simultaneously.

Appendix C

OSGi and Kevoree

C.1 OSGi (Equinox) as the target Adaptive Execution Platform

As shown in Figure 5.10, once we obtain the security-enforced architecture model from the previous steps, we have to reflect this security enforcement in the running system. In case we use Equinox[1] as the target execution platform, all components (business logic components and proxy components) are implemented as OSGi bundles (Spring Dynamic Modules) [202]. In OSGi service platforms, there are two ways to declare and bind services via interfaces (ports): declaring/binding exported services in Spring osgi-context.xml files, or in the source code by overriding the method *start* of *BundleActivator* class of OSGi bundle. Here we show the code for the sake of simplicity but in practice, the declaration of services and bindings can be configured in XML files which means no need to recompile code to change the bindings. Once the services are made available, they can be called from other services. For example, the code snippet in Listing C.1 shows how the *deleteBorrowerAccountService* of a proxy component of Role *Director* is bound to the exported service reference of the *deleteBorrowerAccountService* of the *BorrowerAccountResource* proxy component (lines 1-8). The lines 12-20 show that this Role *Director* can also access to *consultPersonnelAccount* of *PersonnelAccountResource*.

```
  ServiceReference[] refIdeleteBorrowerAccount_DIRECTOR = bundleContext.
      getServiceReferences(
2                 lms.proxy.interfaces.IdeleteBorrowerAccount.class
                      .getName(), "(host=BorrowerAccountResource)");
4 lms.proxy.interfaces.IdeleteBorrowerAccount
      serverIdeleteBorrowerAccount_DIRECTOR = (lms.proxy.interfaces.
      IdeleteBorrowerAccount) bundleContext
```

[1] http://www.eclipse.org/equinox/

```
             .getService(refIdeleteBorrowerAccount_DIRECTOR[0]);
 6
   myDIRECTORService
 8           .setdeleteBorrowerAccountService(
       serverIdeleteBorrowerAccount_DIRECTOR);

10 ...

12 ServiceReference[] refIconsultPersonnelAccount_DIRECTOR = bundleContext.
       getServiceReferences(
                 lms.proxy.interfaces.IconsultPersonnelAccount.class
14                     .getName(), ”(host=PersonnelAccountResource)”);
   lms.proxy.interfaces.IconsultPersonnelAccount
       serverIconsultPersonnelAccount_DIRECTOR = (lms.proxy.interfaces.
       IconsultPersonnelAccount) bundleContext
16           .getService(refIconsultPersonnelAccount_DIRECTOR[0]);

18 myDIRECTORService
             .setconsultPersonnelAccountService(
       serverIconsultPersonnelAccount_DIRECTOR);
```

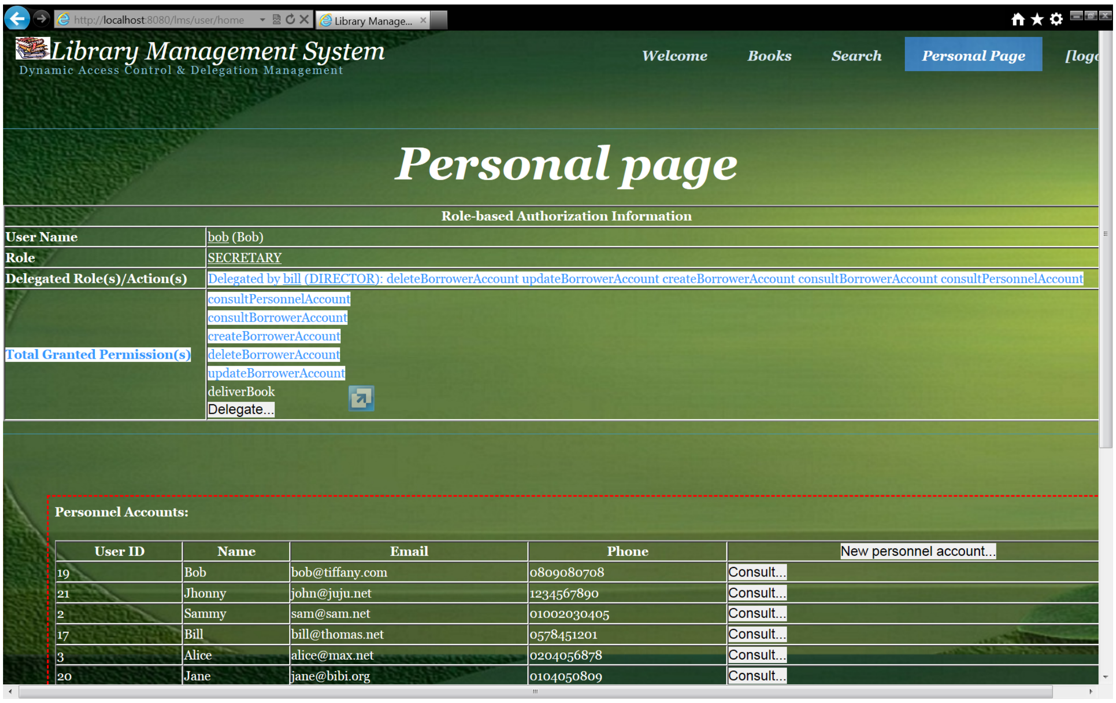
Listing C.1: Services and Bindings in the Director proxy component

As we mentioned before, all the proxy components are very light-weight components. Every method of proxy components only contains the redirecting call to another service that (directly/indirectly) calls to the real method in the business logic. The code snippet in Listing C.2 shows that a call to the *deleteBorrowerAccount* method (line 1) of a proxy component of Role *Director* actually is redirected to call the *deleteBorrowerAccount* method (line 3) of the *BorrowerAccountResource* proxy component that already was made available previously (lines 1-8, Listing C.1). Similarly, the *consultPersonnelAccount* method (line 7) contains a call to the *consultPersonnelAccount* method (line 10) of the *PersonnelAccountResource* proxy component that already was made available previously (lines 12-20, Listing C.1).

```
 1 public void deleteBorrowerAccount(
           lms.bo.user.BorrowerAccount borrowerAccount) throws BSException {
 3     deleteBorrowerAccountService.deleteBorrowerAccount(borrowerAccount);
     }
 5 ...

 7 public lms.bo.user.PersonnelAccount consultPersonnelAccount(
           lms.bo.user.User personnel) throws BSException {
 9     return consultPersonnelAccountService
           .consultPersonnelAccount(personnel);
11 }
```

Listing C.2: Redirecting the method calls in the Director proxy component


http://localhost:8080/lms/user/home
Library Manage...
Library Management System
Dynamic Access Control & Delegation Management
Welcome
Books
Search
Personal Page
[logo
Personal page

Role-based Authorization Information	
User Name	bob (Bob)
Role	SECRETARY
Delegated Role(s)/Action(s)	Delegated by bill (DIRECTOR): deleteBorrowerAccount updateBorrowerAccount createBorrowerAccount consultBorrowerAccount consultPersonnelAccount
Total Granted Permission(s)	consultPersonnelAccount consultBorrowerAccount createBorrowerAccount deleteBorrowerAccount updateBorrowerAccount deliverBook Delegate...

Personnel Accounts:

User ID	Name	Email	Phone	New personnel account...
19	Bob	bob@tiffany.com	0809080708	Consult...
21	Jhonny	john@juju.net	1234567890	Consult...
2	Sammy	sam@sam.net	01002030405	Consult...
17	Bill	bill@thomas.net	0578451201	Consult...
3	Alice	alice@max.net	0204056878	Consult...
20	Jane	jane@bibi.org	0104050809	Consult...

FIGURE C.1: *Bob* is delegated by *Bill* 5 permissions, e.g. *consult personnel account*

The adaptation process is directed by a generated reconfiguration script that is specific for Equinox adaptive execution platform. The reconfiguration script is executed in order to reflect the change of the policy from the model level to the running system, e.g. a new delegation rule is active. Figure C.1 shows a new delegation rule has been enforced in the running system so that *Bob* (*Secretary*) is delegated the permission to *consult personnel account* by *Bill* (*Director*). This means after enforcing this delegation rule, there exists a connection from the port *consultPersonnelAccount* of the User proxy component *Bob*, via corresponding Role and Resource proxy components, to the real method *consultPersonnelAccount* in the business logic.

C.2 Kevoree as the target Adaptive Execution Platform

In case we use Kevoree as the target execution platform, all components (business logic components and proxy components) are implemented as Kevoree component instances [80]. The adaptation process is driven by a generated reconfiguration Kevoree script. The Kevoree script orchestrates the adaptation process of the running system by adding, removing component instances, binding the service ports between proxy components. An example of the configuration of proxy components (the 3-layer architecture) is shown in Figure 5.8. In order to explain how the proxy components are implemented, let's take a look at the *Director* Role proxy component. This Role proxy component is representative as it is between the User layer and the Resource layer (Figure 5.8). It can be seen that this *Director* Role proxy component provides the ports (services) to the User proxy components and also requires the ports (services) of the Resource proxy components. The code snippet in Listing C.3 shows the ports required and provided by the *Director* Role proxy component. The required ports are bound to the corresponding ports provided by the *BorrowerAccountResource* proxy component and the *PersonnelAccountResource* proxy component. The provided ports are to be bound by the ports required by the corresponding User proxy components.

Once the corresponding User proxy component calls the service provided by the port “deleteBorrowerAccountIn” (line 8, Listing C.3), the method *deleteBorrowerAccount* (line 3, Listing C.4) in the *Director* Role proxy component is executed that in turn calls the service provided by the port “deleteBorrowerAccountOut” (line 5, Listing C.4. In fact this port is provided by the *BorrowerAccountResource* proxy component that finally calls to the corresponding method of *deleteBorrowerAccount* in the business logic code.

```
 1 @Requires({
           @RequiredPort(name="deleteBorrowerAccountOut", type = PortType.
       SERVICE, className = IDeleteBorrowerAccount.class, optional = true),
 3         @RequiredPort(name = "consultPersonnelAccountOut", type = PortType.
       SERVICE, className = IconsultPersonnelAccount.class, optional = true),
           ...
 5 })

 7 @Provides({
           @ProvidedPort(name="deleteBorrowerAccountIn", type = PortType.
       SERVICE, className =
 9 IDeleteBorrowerAccount.class),
           @ProvidedPort(name = "consultPersonnelAccountIn", type = PortType.
       SERVICE, className = IconsultPersonnelAccount.class),
11         ...
   })
```

Listing C.3: The ports required and provided by the *Director* Role proxy component

```
   @Override
 2 @Port(name = "deleteBorrowerAccountIn", method = "deleteBorrowerAccount")
   public void deleteBorrowerAccount(BorrowerAccount borrowerAccount) throws
       BSException {
 4
        IDeleteBorrowerAccount deleteBorrowerAccountPort = getPortByName("
       deleteBorrowerAccountOut",IDeleteBorrowerAccount.class);
 6
        deleteBorrowerAccountPort.deleteBorrowerAccount(borrowerAccount);
 8 }
```

Listing C.4: Redirecting the method call in the *Director* Role proxy component in Kevoree

There are three main advantages of using Kevoree over OSGi as the execution platform. Firstly, all we need to provide for the platform is the Kevoree reconfiguration script saying how to adapt the system. The Kevoree execution platform takes care of the necessary adaptation order for the running system according to changes. In case of using OSGi, we have to take care of the adaptation order manually. Secondly, the *model@runtime* environment of Kevoree makes it easier for implementing our model driven framework. In Kevoree, we can use the Kevoree framework itself to manage the security policy models. Thirdly, the way of declaring ports and bindings in Kevoree are very close to the concepts of ports and bindings described in our 3-layer architecture (Figure 5.8). This makes it very convenient to implement the running systems in Kevoree.

Appendix D

More of SoSPa

In this part, we present the RAM models of some more aspects that are mostly based on the patterns in [211, 72, 220].

D.1 More Patterns for Authentication, Authorisation

Fig. D.1 shows a simple *Password* aspect. The *Password* aspect defines how user password is created and managed. This aspect can be reused, e.g. in the *DirectAuthentication* pattern, to check if the password used for login is matched with the correct password managed by the system (using `compare` method).

As shown in Fig. D.2, before looking up for active permissions of the request subject, the policy decision point (*PDP*) gets information of the subject (e.g. the roles of the subjects) and other information about the resource (e.g. conflict-of-interests) and environment (e.g. time-bound context, or location-based context). The `lookup` method will access the *PolicyRepository* to retrieve all the active permissions according to the subject, the resource, and the environment. The *PDP* then evaluates all the data and comes up with a final decision for the request.

Fig. D.3 shows how the policy(s) are managed by the PolicyRepository.

D.2 More Patterns for Auditing

Fig. D.4 shows how a `SecureLogger` can use a `LogManager` to take care of the actual logging. In fact, the `LogManager` gets an instance of `Logger` to delegate the logging

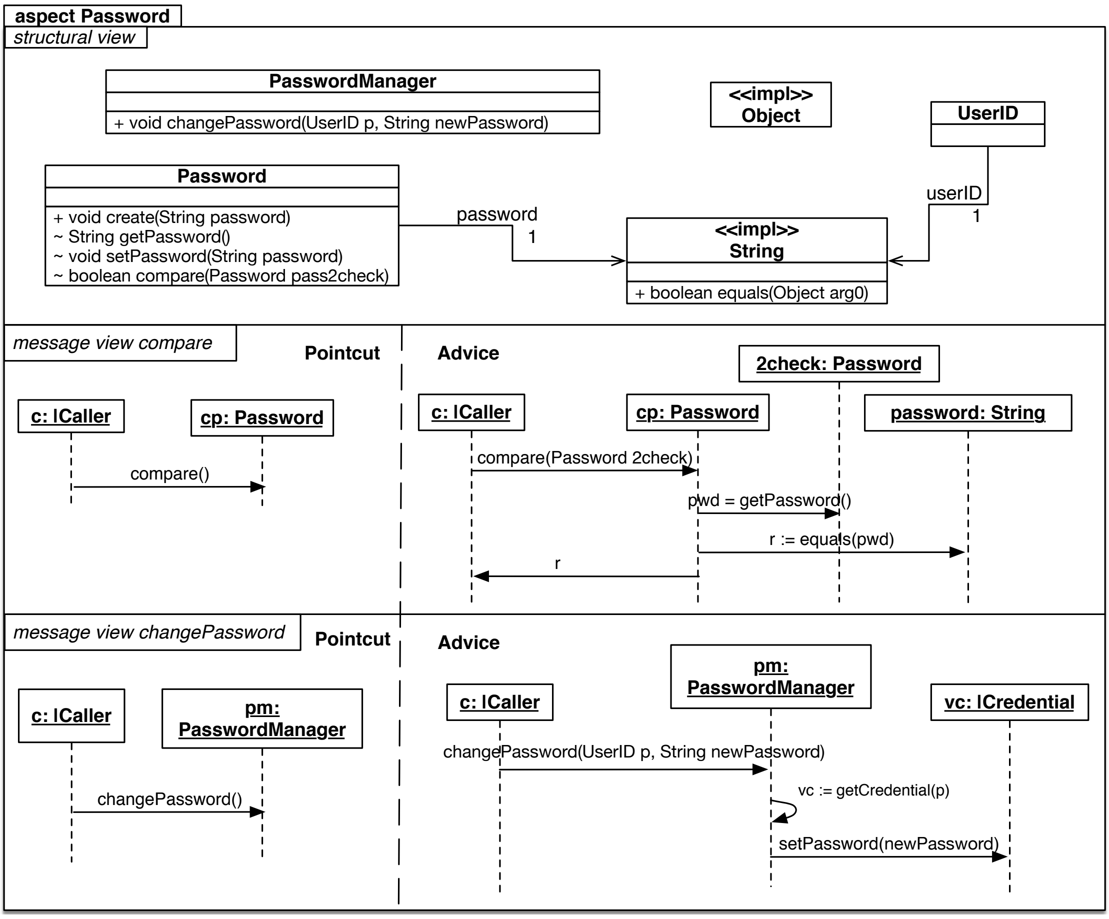


FIGURE D.1: Aspect *Password*

action. These are based on the *Secure Logger*, *Audit Interceptor* patterns [220], and the *Security Logger and Auditor* pattern [72].

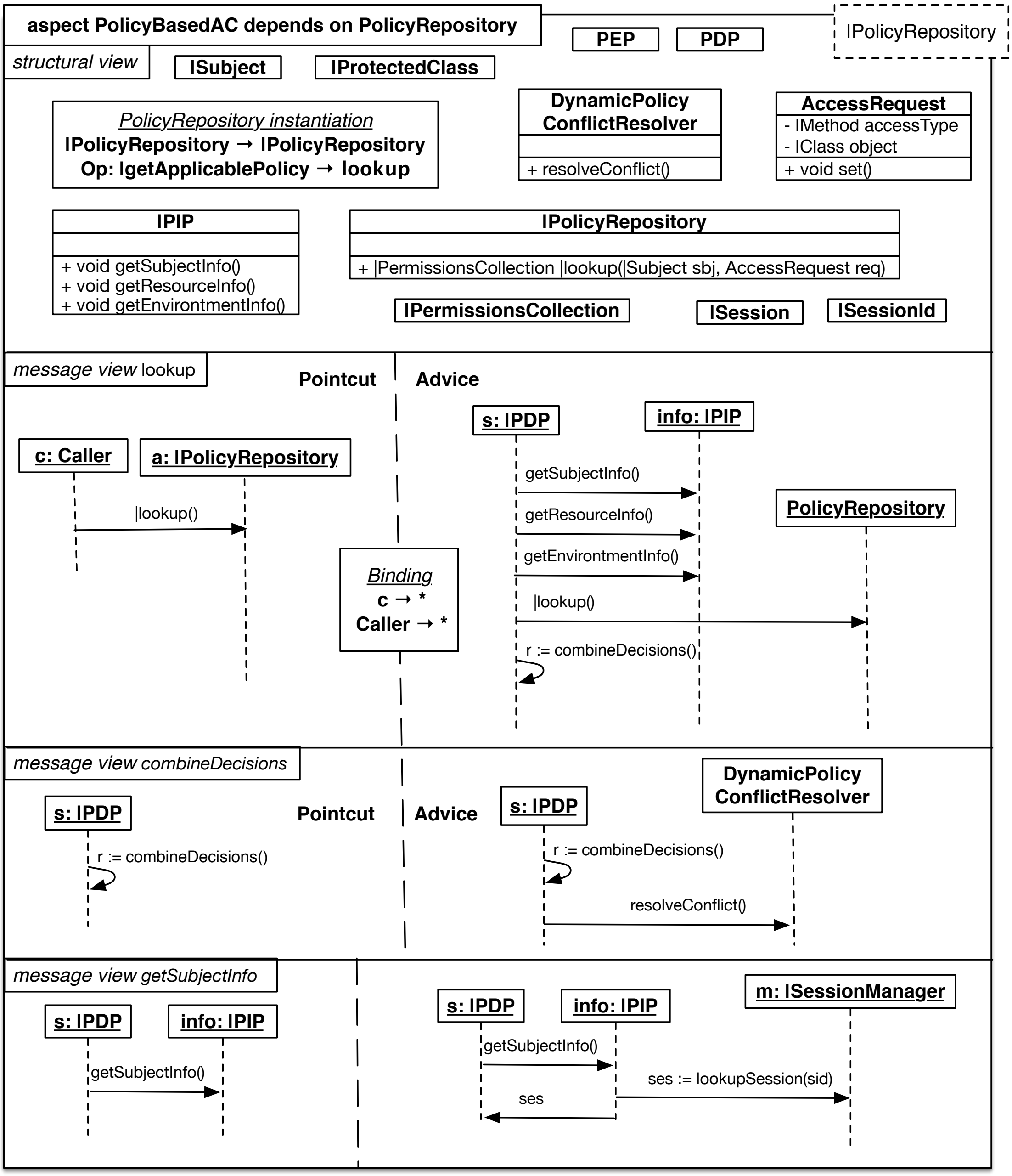


Figure D.2: Aspect *PolicyBasedAC*

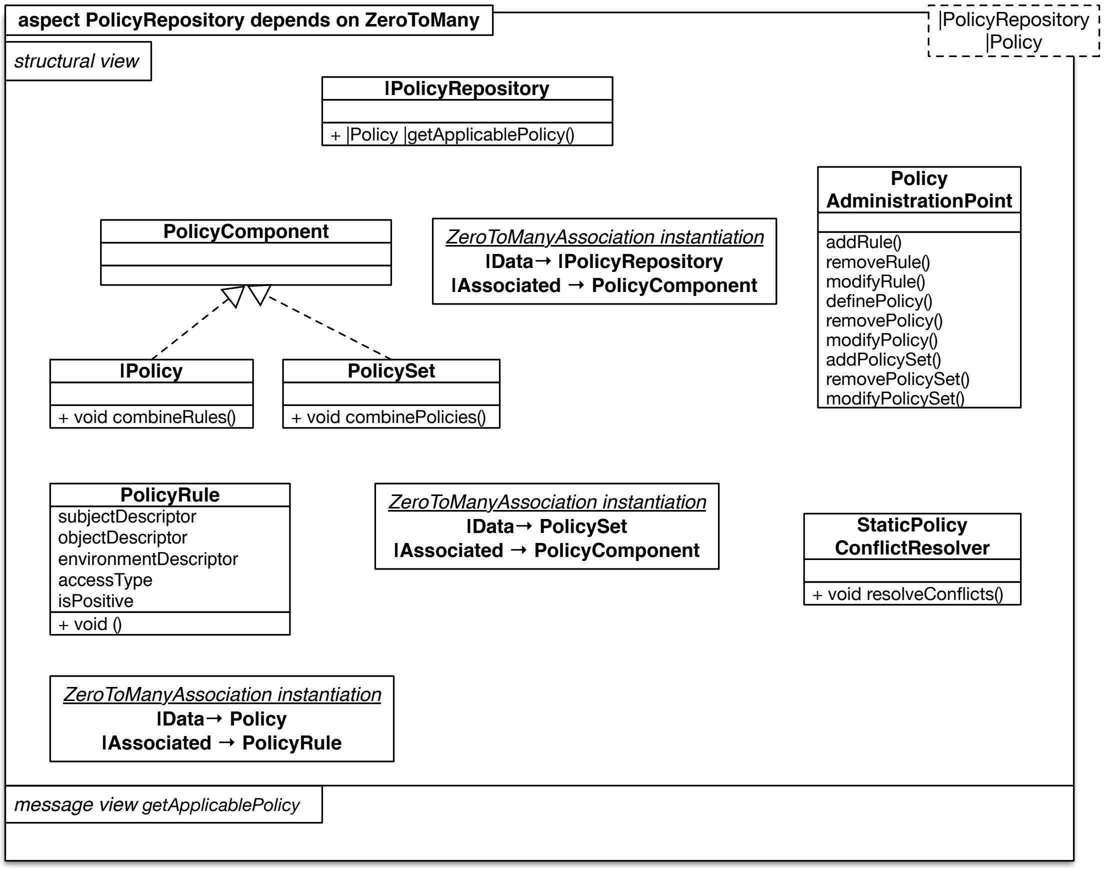


Figure D.3: Aspect *PolicyRepository*

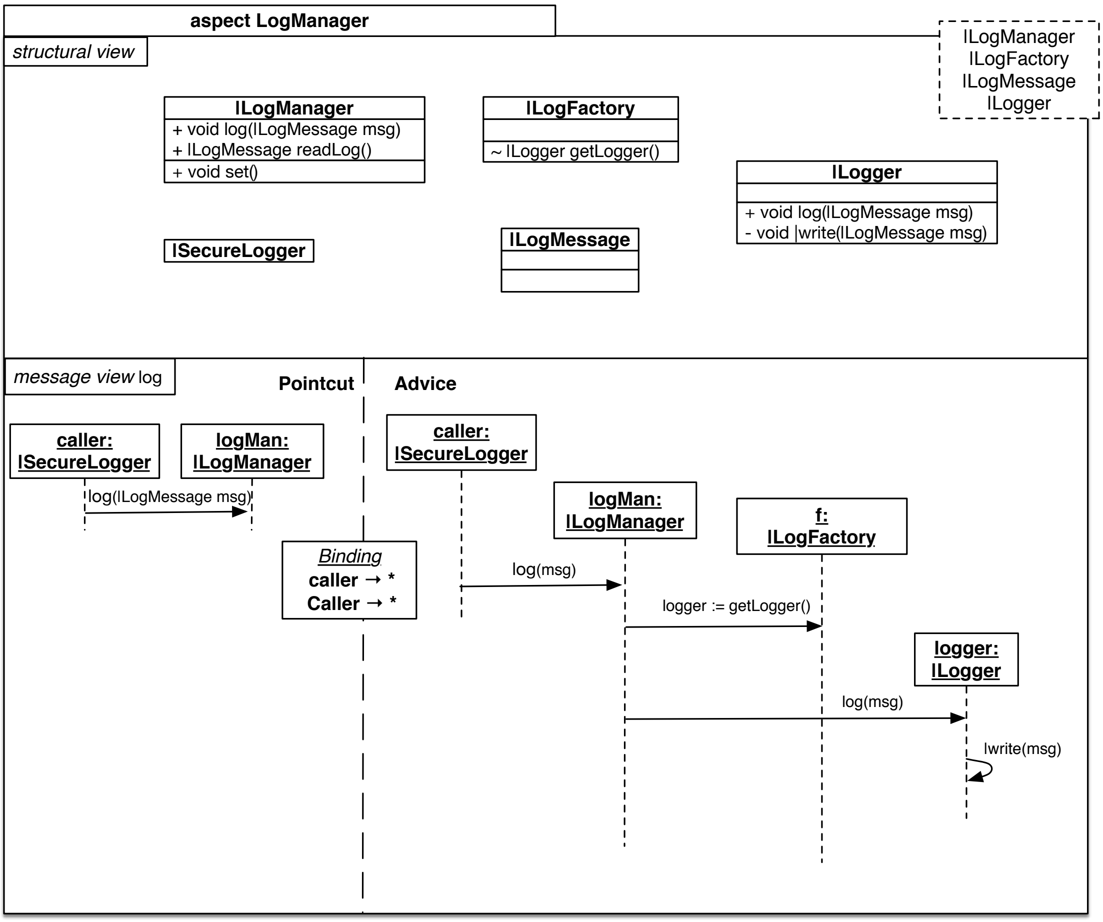


Figure D.4: Aspect *LogManager*

Appendix E

List of Publications, Services, Grants, and Awards

E.1 Publications included in the thesis

- **Phu Hong Nguyen**, Jacques Klein, Yves Le Traon, and Max E. Kramer."A Systematic Review of Model-Driven Security." In Software Engineering Conference (APSEC, 2013 20th Asia-Pacific, vol. 1, pp. 432-441. IEEE, 2013. [182]

- **Phu Hong Nguyen**, Max E. Kramer, Jacques Klein, and Yves Le Traon. "An Extensive Systematic Review on the Model-Driven Development of Secure Systems." In Information and Software Technology, 2015. [183]

- Levi Lucio, Qin Zhang, **Phu Hong Nguyen**, Moussa Amrani, Jacques Klein, Hans Vangheluwe, and Yves Le Traon. "Advances in Model-Driven Security." Advances in Computers 93 (2014): 103-152. [144]

- **Phu Hong Nguyen**, Gregory Nain, Jacques Klein, Tejeddine Mouelhi, and Yves Le Traon. "Model-driven adaptive delegation." In Proceedings of the 12th annual international conference on Aspect-oriented software development, pp. 61-72. ACM, 2013. (one of the best papers) [190]

- **Phu Hong Nguyen**, Gregory Nain, Jacques Klein, Tejeddine Mouelhi, and Yves Le Traon. "Modularity and Dynamic Adaptation of Flexibly Secure Systems: Model-Driven Adaptive Delegation in Access Control Management." In Transactions on Aspect-Oriented Software Development XI, pp. 109-144. Springer Berlin Heidelberg, 2014. [184]

- **Phu Hong Nguyen**, Mike Papadakis, and Iram Rubab. "Testing Delegation Policy Enforcement via Mutation Analysis." In Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on, pp. 34-42. IEEE, 2013. [191]
- **Phu Hong Nguyen**, Jacques Klein, and Yves Le Traon. "Model-Driven Security with A System of Aspect-Oriented Security Design Patterns." In 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling. 2014. [181]
- **Phu Hong Nguyen**, Koen Yskout, Thomas Heyman, Jacques Klein, Riccardo Scandariato, and Yves Le Traon. "SoSPa: A System of Security Design Patterns for Systematically Engineering Secure Systems." In ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems. 2015. [185]

E.2 Other publications/presentations

- **Phu Hong Nguyen**, Jacques Klein, and Yves Le Traon. "Model-Driven Security with Modularity and Reusability for Secure Systems Development." In STAF-DS. 2015. [187]
- **Phu Hong Nguyen**, Koen Yskout, Thomas Heyman, Jacques Klein, Riccardo Scandariato, and Yves Le Traon. "Model-Driven Security based on A Unified System of Security Design Patterns.", Technical Report, University of Luxembourg, 2015. [186].
- **Phu Hong Nguyen**, Jacques Klein, and Yves Le Traon. "Modeling, composing, and testing of security concerns in a Model-Driven Security approach." International Symposium on Engineering Secure Software and Systems-Doctoral Symposium. 2014. [189]
- **P. H. Nguyen**, J. Klein, G. Nain, and Y. Le Traon, "Migrating Legacy Secure Systems to Model-Driven Adaptive Secure Systems", a position paper (2 pages) presented at the 11th edition of the BElgian-NEtherlands software eVOLution symposium (BENEVOL 2012) held at the Delft University of Technology, The Netherlands, 2012.
- van Amstel, Marcel F., Mark GJ van den Brand, and **Phu H. Nguyen**. "Metrics for model transformations". In Proceedings of the Ninth Belgian-Netherlands Software Evolution Workshop (BENEVOL 2010), Lille, France (December 2010). 2010. [17]

- **P.H. Nguyen**, Quantitative Analysis of Model Transformations. Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2010) [188].

E.3 Awards/Grants

- Huygens Talent Scholarship: Full scholarship provided by the Minister for Education, Culture and Science of the Netherlands to the most talented international students, for studying the 2-year Master programme at the Eindhoven University of Technology, 2008-2010 (total value > 40kEUR).
- A full grant by the NATO Academic of Science for participating in the summer school Marktoberdorf 2014 on Dependable Software Systems Engineering.
- A full grant (EUR 2000) by the National Research Fund (FNR), Luxembourg for participating in the summer school SFM12:MDE, 2012.
- Registration and accommodation grant for participating in the Summer School on Software Synthesis (ExCAPE) held at UC Berkeley, in June 2013.
- Full grants for participating in the International Summer Schools on Training And Research On Testing (TAROT) 2012, 2013. Travel grants AOSD2013, ESSoS2014.
- Certificate for exceptional contributions, support, and commitment in the organisation of the Sixth IEEE International Conference on Software Testing, Verification and Validation (ICST 2013).
- Japan-ASEAN University Network Scholarship, 2007.
- Award for displaying exceptional personal dedication, teamwork and contribution to the ibm.com project, 2007.
- Won the first prize at the LuxDoc Science Slam 2014 in Luxembourg for presenting my scientific, multilingual poem on *Model-Driven Security*.

E.4 Professional, Teaching, and Social Activities

- An external reviewer for the following journal(s)/conference(s)/workshop(s): INFSOF, ISEB, QRS 2015, COMPSAC 2015, QSIC 2014, SAM 2014, VAO 2014, ICWS 2014, ICSE-NIER 2014, MODELS 2013, ISARCS 2013, AOSD 2013, SAC-SVT 2013, QSIC 2013, Modevva 2012, toolseurope 2012, ICST 2012.

- A webmaster, a student volunteer, and the main photographer for the ICST 2013 in Luxembourg, `http://www.icst.lu`, 2013.
- A member of IEEE, ACM, 2013-Present.
- A representative of students (at the Faculte des Sciences, de la Technologie et de la Communication) in the University Council, University of Luxembourg, 2013-2015.
- An active (committee) member of the organisation of young researchers in Luxembourg, LuxDoc (`http://www.LuxDoc.org`), 2013-2015.
- The vice chair representative of the SnT PhD Student Association, 2013-2015.
- A member of Huygens Talent Circle, 2008-Present.
- A member of Toastmasters International, 2013-Present.
- A lecturer at the FPT University, Vietnam, 2011.
- A teaching assistant at the University of Luxembourg, 2011-2015.

E.5 Certificates for Personal Development Skills

So far I have taken the trainings in the following personal development skills, mainly provided by the University of Luxembourg.

- Presentation Skills for Scientific Conferences.
- Leading and Planning.
- Good Scientific Practice.
- Introduction to Entrepreneurship and Entrepreneurial Behaviour.
- Lecturing and Teaching.
- Time and Self-Management.
- Proposal Writing for Young Researchers.
- Project Management.
- Media Interview Training.
- Poster Presentation.
- Developing Intercultural Skills and Communication in Multilingual Context.
- Scientific Writing.

Bibliography

[1] J. Abramov, O. Anson, M. Dahan, P. Shoval, and A. Sturm. “A methodology for integrating access control policies within database development”. In: *Computers & Security* 31.3 (2012), pp. 299–314.

[2] J. Abramov, O. Anson, A. Sturm, and P. Shoval. “Tool support for enforcing security policies on databases”. In: *Conference on Advanced Information Systems Engineering* (2012), pp. 126–141.

[3] J. Abramov, A. Sturm, and P. Shoval. “Evaluation of the Pattern-based method for Secure Development (PbSD): A controlled experiment”. In: *Information and Software Technology* 54.9 (2012), pp. 1029–1043.

[4] T. Ahmed and A. R. Tripathi. “Static verification of security requirements in role based CSCW systems”. In: *SACMAT 03: Proceedings of the eighth ACM symposium on Access control models and technologies* (2003), pp. 196–203.

[5] G.-J. Ahn, B. Mohan, and S.-P. Hong. “Towards Secure Information Sharing using Role-Based Delegation”. In: *J. Netw. Comput. Appl.* 30.1 (2007), pp. 42–59.

[6] G. Ahn and H. Hu. “Towards realizing a formal RBAC model in real systems”. In: *Proceedings of the 12th ACM symposium on Access control models and technologies* (2007), p. 215.

[7] W. Al Abed, V. Bonnet, M. Schöttle, E. Yildirim, O. Alam, and J. Kienzle. “TouchRAM: A multitouch-enabled tool for aspect-oriented software design”. In: *Software Language Engineering*. Springer, 2013, pp. 275–285.

[8] M. Alam, R. Breu, and M. Hafner. “Modeling permissions in a (U/X) ML world”. In: (2006), pages.

[9] M. Alam, R. Breu, and M. Breu. “Model driven security for Web services (MDS4WS)”. In: *Multitopic Conference, 2004. Proceedings of INMIC 2004. 8th International.* 2004, pp. 498–505.

[10] M. Alam. "Model driven security engineering for the realization of dynamic security requirements in collaborative systems". In: *Models in Software Engineering* 4364 (2007), pp. 278–287.

[11] M. Alam, M. Hafner, and R. Breu. "Constraint based role based access control in the SECTET-framework: A model-driven approach". In: *Journal of Computer Security* (2008), pp. 1–13.

[12] M. Alam, M. Hafner, R. Breu, and S. Unterthiner. "A framework for modeling restricted delegation in service oriented architecture". In: *Trust and Privacy in Digital Business* (2006), pp. 142–151.

[13] O. Alam, J. Kienzle, and G. Mussbacher. "Concern-Oriented Software Design". In: *MODELS*. 2013.

[14] M. Almorsy and J. Grundy. "SecDSVL: A Domain-Specific Visual Language To Support Enterprise Security Modelling". In: *Software Engineering Conference (ASWEC), 2014 23rd Australian*. IEEE. 2014, pp. 152–161.

[15] M. Almorsy, J. Grundy, and A. S. Ibrahim. "Adaptable, model-driven security engineering for SaaS cloud-based applications". In: *Automated Software Engineering* 21.2 (2013), pp. 187–224.

[16] M. Almorsy, J. Grundy, and A. S. Ibrahim. "Mdse@ r: model-driven security engineering at runtime". In: *Cyberspace Safety and Security*. Springer, 2012, pp. 279–295.

[17] M. F. van Amstel, M. G. van den Brand, and P. H. Nguyen. "Metrics for model transformations". In: *Proceedings of the Ninth Belgian-Netherlands Software Evolution Workshop (BENEVOL 2010)*. 2010.

[18] R. Anderson. *Security engineering: A guide to building dependable distributed systems. 2001.*

[19] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin. "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria". In: *IEEE Trans. Software Eng.* 32.8 (2006), pp. 608–624.

[20] ATLAS. *ATLAS Transformation Language*. http://www.eclipse.org/m2m/atl/. 2008.

[21] C. Y. Baldwin and K. B. Clark. *Design rules: The power of modularity*. Vol. 1. MIT press, 2000.

[22] E. Barka and R. Sandhu. "Role-Based Delegation Model/Hierarchical Roles (RBDM1)". In: *Proceedings of the 20th Annual Computer Security Applications Conference*. ACSAC 2004. IEEE Computer Society, 2004, pp. 396–404.

[23] S. Barker and M. Fernández. “Term rewriting for access control”. In: *Data and applications security XX*. Springer, 2006, pp. 179–193.

[24] D. Basin, M. Clavel, J. Doser, and M. Egea. “A Metamodel-Based Approach for Analyzing Security-Design Models”. In: *Model Driven Engineering Languages and Systems*. Ed. by G. Engels, B. Opdyke, D. Schmidt, and F. Weil. Vol. 4735. Lecture Notes in Computer Science. Springer, 2007, pp. 420–435.

[25] D. Basin, M. Clavel, J. Doser, and M. Egea. “Automated analysis of security-design models”. In: *Information & Software Technology* 51.5 (2009), pp. 815–831.

[26] D. Basin, M. Clavel, and M. Egea. “A Decade of Model-Driven Security”. In: *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, SACMAT 2011*. Innsbruck, Austria: ACM, 2011, pp. 1–10.

[27] D. Basin, M. Clavel, and M. Egea. “A decade of model-driven security”. In: *Proceedings of the 16th ACM symposium on Access control models and technologies*. SACMAT ’11. Innsbruck, Austria: ACM, 2011, pp. 1–10.

[28] D. Basin, M. Clavel, and M. Egea. “Model-driven development of security-aware GUIs for data-centric applications”. In: *Springer* (2011), pp. 101–124.

[29] D. Basin, M. Clavel, M. Egea, M. A. G. de Dios, and C. Dania. “A model-driven methodology for developing secure data-management applications”. In: *Software Engineering, IEEE Transactions on* 40.4 (2014), pp. 324–337.

[30] D. Basin, J. Doser, and T. Lodderstedt. “Model Driven Security for Process-Oriented Systems”. In: *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies, SACMAT 2003*. Como, Italy: ACM, 2003, pp. 100–109.

[31] D. Basin, J. Doser, and T. Lodderstedt. “Model driven security for process-oriented systems”. In: *Proceedings of the eighth ACM symposium on Access control models and technologies*. SACMAT ’03. ACM, 2003, pp. 100–109.

[32] D. Basin, J. Doser, and T. Lodderstedt. “Model Driven Security: from UML Models to Access Control Infrastructures”. In: *Transactions on Software Engineering and Methodology*. Vol. 15. ACM, 2006, pp. 39–91.

[33] M. Ben-Ghorbel-Talbi, F. Cuppens, N. Cuppens-Boulahia, and A. Bouhoula. “A delegation model for extended RBAC”. In: *International Journal of Information Security* 9.3 (2010), pp. 209–236.

[34] E. Bertino, S. Jajodia, and P. Samarati. “A flexible authorization mechanism for relational data management systems”. In: *ACM Trans. Inf. Syst.* 17.2 (1999), pp. 101–140.

[35] A. Bertolino, M. Busch, S. Daoudagh, F. Lonetti, and E. Marchetti. “A Toolchain for Designing and Testing Access Control Policies”. In: *Engineering Secure Future Internet Services and Systems*. Springer, 2014, pp. 266–286.

[36] C. Bertolissi, M. Fernández, and S. Barker. “Dynamic event-based access control as term rewriting”. In: *Data and Applications Security XXI*. Springer, 2007, pp. 195–210.

[37] B. Best, J. Jürjens, and B. Nuseibeh. “Model-Based Security Engineering of Distributed Information Systems Using UMLsec”. In: *29th International Conference on Software Engineering, 2007. ICSE 2007.* 2007, pp. 581–590.

[38] J. Bezivin. “Model Driven Engineering: An Emerging Technical Space”. In: *Generative and Transformational Techniques in Software Engineering*. Vol. 4143. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 36–64.

[39] J. Biolchini, P. G. Mian, A. C. C. Natali, and G. H. Travassos. “Systematic review in software engineering”. In: *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES* 679.05 (2005), p. 45.

[40] M. Bishop. *Computer security: art and science*. Vol. 200. Addison-Wesley, 2012.

[41] C. Blanco, E. Fernandez-Medina, and J. Trujillo. “Modernizing Secure OLAP Applications with a Model-Driven Approach”. In: *The Computer Journal* (2014).

[42] C. Blanco. “Applying QVT in order to implement secure data warehouses in SQL Server Analysis Services”. In: *Journal of Research and Practice in Information Technology* 41.2 (2009).

[43] C. Blanco and I. de Guzmán. “Showing the Benefits of Applying a Model Driven Architecture for Developing Secure OLAP Applications.” In: *J. UCS* 20.2 (2014), pp. 79–106.

[44] C. Blanco and R. Pérez-Castillo. “Towards a modernization process for Secure Data Warehouses”. In: *Data Warehousing and Knowledge Discovery* Mda 2003 (2009), pp. 24–35.

[45] M. Borek, N. Moebius, K. Stenzel, and W. Reif. “Model-Driven Development of Secure Service Applications”. In: *2012 35th Annual IEEE Software Engineering Workshop* (2012), pp. 62–71.

[46] R. Bouaziz, S. Kallel, and B. Coulette. “An Engineering Process for Security Patterns Application in Component Based Models”. In: *Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE, 2013, pp. 231–236.

[47] C. Braga. “A transformation contract to generate aspects from access control policies”. In: *Software and Systems Modeling* 10.3 (2011), pp. 395–409.

[48] M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. 1st ed. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.

[49] R. Breu, M. Hafner, F. Innerhofer-Oberperfler, and F. Wozak. “Model-Driven Security Engineering of Service Oriented Systems”. In: *Information Systems and e-Business Technologies*. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2008, pp. 59–71.

[50] R. Breu, M. Hafner, B. Weber, and A. Novak. “Model Driven Security for Inter-Organizational Workflows in e-Government”. In: *E-Government: Towards Electronic Democracy*. Vol. 3416. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 122–133.

[51] R. Breu, G. Popp, and M. Alam. “Model based development of access policies”. In: *International Journal on Software Tools for Technology Transfer* 9.5-6 (2007), pp. 457–470.

[52] A. Brucker, J. Doser, and B. Wolff. “A Model Transformation Semantics and Analysis Methodology for SecureUML”. In: Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 306–320.

[53] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J. Stefani. “The Fractal Component Model and its Support in Java”. In: *Software Practice and Experience, Special Issue on Experiences with Auto-adaptive and Reconfigurable Systems* 36.11-12 (2006), pp. 1257–1284.

[54] C. C. Burt, B. R. Bryant, R. R. Raje, A. Olson, and M. Auguston. “Model driven security: unification of authorization models for fine-grain access control”. In: *Enterprise Distributed Object Computing Conference, 2003. Proceedings. Seventh IEEE International* (2003), pp. 159–171.

[55] M. Busch, N. Koch, and S. Suppan. “Modeling Security Features of Web Applications”. In: *Engineering Secure Future Internet Services and Systems*. Springer, 2014, pp. 119–139.

[56] V. C. Hu, E. Martin, J. Hwang, and T. Xie. “Conformance Checking of Access Control Policies Specified in XACML”. In: *Computer Software and Applications Conference, COMPSAC 2007, 31st Annual International* 2 (2007), pp. 275–280.

[57] S. Clarke. “Composition of Object-Oriented Software Design Models”. PhD thesis. Dublin City University, Ireland, 2001.

[58] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley, 2005.

[59] M. Clavel, V. Silva, C. Braga, and M. Egea. “Model-Driven Security in Practice: An Industrial Experience”. In: *Model Driven Architecture – Foundations and Applications*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 326–337.

[60] S. Cook, G. Jones, S. Kent, and A. C. Wils. *Domain-Specific Development with Visual Studio DSL Tools*. Addison-Wesley, 2007.

[61] J. Crampton and H. Khambhammettu. “Delegation in role-based access control”. In: *International Journal of Information Security* 7.2 (2007), pp. 123–136.

[62] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. Chaudron. “A classification framework for software component models”. In: *Software Engineering, IEEE Transactions on* 37.5 (2011), pp. 593–615.

[63] F. Cuppens and N. Cuppens-Boulahia. “Modeling contextual security policies”. In: *International Journal of Information Security* 7.4 (2007), pp. 285–305.

[64] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. “Hints on Test Data Selection: Help for the Practicing Programmer”. In: *Computer* 11.4 (1978), pp. 34–41.

[65] M. A. G. de Dios, C. Dania, D. Basin, and M. Clavel. “Model-Driven Development of a Secure eHealth Application”. In: *Engineering Secure Future Internet Services and Systems* (2014), pp. 97–118.

[66] T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl. “MAC and UML for secure software design”. In: *FMSE ’04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*. Washington DC: ACM, 2004, pp. 75–85.

[67] G. Dupe, M. Belaunde, R. Perruchon, H. Besnard, F. Guillard, and V. Oliveres. *SmartQVT*. http://smartqvt.elibel.tm.fr/.

[68] M. Eby, J. Werner, G. Karsai, and A. Ledeczi. “Integrating security modeling into embedded system design”. In: *Engineering of Computer-Based Systems, 2007. ECBS’07. 14th Annual IEEE International Conference and Workshops on the.* IEEE. 2007, pp. 221–228.

[69] Y. Elrakaiby, M. Amrani, and Y. Le Traon. “Security@ runtime: A flexible mde approach to enforce fine-grained security policies”. In: *Engineering Secure Software and Systems*. Springer, 2014, pp. 19–34.

[70] Y. Elrakaiby, T. Mouelhi, and Y. Le Traon. “Testing Obligation Policy Enforcement Using Mutation Analysis”. In: *ICST ’12 Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation* (2012), pp. 673–680.

[71] M. Felderer, P. Zech, R. Breu, M. Büchler, and A. Pretschner. “Model-based security testing: a taxonomy and systematic classification”. In: *Software Testing, Verification and Reliability* (2015).

[72] E. Fernandez-Buglioni. *Security patterns in practice: designing secure architectures using software patterns.* John Wiley & Sons, 2013.

[73] E. Fernández-Medina and M. Piattini. “Extending OCL for secure database development”. In: *UML 2004—The Unified Modeling Language. Modeling Languages and Applications* (2004), pp. 380–394.

[74] E. Fernández-Medina and J. Trujillo. “Extending UML for designing secure data warehouses”. In: *Conceptual Modeling–ER 2004* (2004), pp. 217–230.

[75] E. Fernández-Medina and M. Piattini. “Designing Secure Databases”. In: *Information and Software Technology.* Vol. 47. Elsevier, 2005, pp. 463–477.

[76] E. Fernández-Medina, J. Trujillo, R. Villarroel, and M. Piattini. “Developing secure data warehouses with a UML extension”. In: *Information Systems* 32.6 (2006), pp. 826–856.

[77] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. “Proposed NIST standard for role-based access control”. In: *ACM Trans. Inf. Syst. Secur.* 4.3 (2001), pp. 224–274.

[78] T. Fink, M. Koch, and K. Pauls. “An MDA approach to access control specifications using MOF and UML profiles”. In: *Electronic Notes in Theoretical Computer Science* 142 (2006), pp. 161–179.

[79] F. Fleurey, B. Baudry, R. France, and S. Ghosh. “A Generic Approach For Automatic Model Composition”. In: *Proceedings of the 11th International Workshop on Aspect-Oriented Modeling, AOM at MoDELS 2007.* Nashville, TN, USA: Springer-Verlag, 2007, pp. 7–15.

[80] F. Fouquet, G. Nain, B. Morin, E. Daubert, O. Barais, N. Plouzeau, and J.-M. Jézéquel. “An Eclipse Modelling Framework Alternative to Meet the Models@Runtime Requirements”. In: *Model Driven Engineering Languages and Systems.* Ed. by R. France, J. Kazmeier, R. Breu, and C. Atkinson. Vol. 7590. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 87–101.

[81] J. Fox and J. Jurjens. “Introducing security aspects with model transformation”. In: *Engineering of Computer-Based Systems, 2005. ECBS’05. 12th IEEE International Conference and Workshops on the* (2005).

[82] R. France, I. Ray, G. Georg, and S. Ghosh. “Aspect-Oriented Approach to Early Design Modelling”. In: *IEE Proceedings - Software.* Vol. 151. IEEE Computer Society, 2004, pp. 173–185.

[83] J. P. S. Gallino, M. de Miguel, J. F. Briones, and A. Alonso. "Domain-Specific multi-modeling of security concerns in service-oriented architectures". In: *Web Services and Formal Methods* (2012), pp. 128–142.

[84] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[85] G. Georg, I. Ray, K. Anastasakis, B. Bordbar, M. Toahchoodee, and S. H. Houmb. "An aspect-oriented methodology for designing secure applications". In: *Information and Software Technology* 51.5 (2009), pp. 846–864.

[86] S. Gilmore, L. Gönczy, N. Koch, P. Mayer, M. Tribastone, and D. Varró. "Non-functional properties in the model-driven development of service-oriented systems". In: *Software & Systems Modeling* 10.3 (2010), pp. 287–311.

[87] M. Hafner, M. Alam, and R. Breu. "Towards a MOF/QVT-Based Domain Architecture for Model Driven Security". In: *Model Driven Engineering Languages and Systems*. Vol. 4199. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 275–290.

[88] M. Hafner and R. Breu. "Realizing model driven security for inter-organizational workflows with ws-cdl and uml 2.0". In: *Model Driven Engineering Languages and Systems* 3713 (2005), pp. 39–53.

[89] M. Hafner and R. Breu. "Extending Sectet : Advanced Security Policy". In: *Security Engineering for Service-Oriented Architectures*. Springer Berlin Heidelberg, 2009, pp. 159–188.

[90] M. Hafner and R. Breu. "Modeling Security Critical SOA Applications". In: *Security Engineering for Service-Oriented Architectures*. Springer Berlin Heidelberg, 2009, pp. 93–119.

[91] M. Hafner, R. Breu, B. Agreiter, and A. Nowak. "SECTET: An Extensible Framework for the Realization of Secure Inter-Organizational Workflows". In: *Internet Research*. Vol. 16. Emerald Group Publishing Limited, 2006, pp. 491–506.

[92] M. Hafner, M. Memon, and M. Alam. "Modeling and enforcing advanced access control policies in healthcare systems with sectet". In: *Models in Software Engineering* (2008), pp. 132–144.

[93] R. G. Hamlet. "Testing Programs with the Aid of a Compiler". In: *IEEE Trans. Softw. Eng.* 3.4 (1977), pp. 279–290.

[94] F. Hansen and V. Oleshchuk. "Conformance checking of rbac policy and its implementation". In: *Proceedings of Information Security Practice and Experience: First International Conference, ISPEC* 3439 / 2005 (2005).

[95] T. Heyman, K. Yskout, R. Scandariato, and W. Joosen. “An Analysis of the Security Patterns Landscape”. In: *Proceedings of SESS ’07.* 2007.

[96] B. Hoisl and S. Sobernig. “Integrity and confidentiality annotations for service interfaces in SoaML models”. In: *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on.* IEEE. 2011, pp. 673–679.

[97] B. Hoisl, S. Sobernig, and M. Strembeck. “Modeling and enforcing secure object flows in process-driven SOAs: an integrated model-driven approach”. In: *Software & Systems Modeling* 13.2 (2012), pp. 513–548.

[98] J.-M. Horcas, M. Pinto, and L. Fuentes. “An aspect-oriented model transformation to weave security using CVL”. In: *MODELSWARD.* 2014, pp. 138–150.

[99] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. “Empirical assessment of MDE in industry”. In: *Proceedings of the 33rd International Conference on Software Engineering.* ICSE ’11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 471–480.

[100] I. Jacobson and P. Ng. *Aspect-Oriented Software Development with Use Cases.* Addison-Wesley, 2004.

[101] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. “Flexible support for multiple access control policies”. In: *ACM Trans. Database Syst.* 26.2 (2001), pp. 214–260.

[102] S. Jalali and C. Wohlin. “Systematic literature studies: database searches vs. backward snowballing”. In: *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement.* 2012, pp. 29–38.

[103] J. Jensen and M. G. Jaatun. “Security in Model Driven Development: A Survey”. In: *Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security.* ARES ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 704–709.

[104] Y. Jia and M. Harman. “An Analysis and Survey of the Development of Mutation Testing”. In: *IEEE Trans. Software Eng.* 37.5 (2011), pp. 649–678.

[105] J. Jürjens. “Model-based security engineering for real”. In: *FM 2006: Formal Methods* (2006), pp. 600–606.

[106] J. Jürjens. “Model-based security engineering with UML”. In: *Foundations of Security Analysis and Design III* (2005), pp. 42–77.

[107] J. Jürjens. “UMLsec: Extending UML for Secure Systems Development”. In: *UML’02: 5th International Conference on The UML.* Dresden, Germany: Springer-Verlag, 2002, pp. 412–425.

[108] J. Jürjens and S. Houmb. “Dynamic secure aspect modeling with UML: From models to code”. In: *Model Driven Engineering Languages and Systems* (2005), pp. 142–155.

[109] J. Jürjens. “Formal Semantics for Interacting UML Subsystems”. In: *Proceedings of the IFIP TC6/WG6.1 Fifth International Conference on Formal Methods for Open Object-Based Distributed Systems, FMOODS 2002.* University of Twente, Netherlands: Wolters Kluwer, 2002, pp. 29–43.

[110] J. Jürjens. “Principles for Secure Systems Design”. PhD thesis. Oxford University, United Kingdom, 2002.

[111] J. Jürjens. “Towards Development of Secure Systems Using UMLsec”. In: *Fundamental Approaches to Software Engineering.* Vol. 2029. Lecture Notes in Computer Science. Springer-Verlag, 2001, pp. 187–200.

[112] J. Jürjens. “UMLsec: Extending UML for Secure Systems Development”. In: *UML 2002 - The Unified Modeling Language.* Vol. 2460. Lecture Notes in Computer Science. Springer-Verlag, 2002, pp. 412–425.

[113] J. Jürjens. “Using UMLsec and goal trees for secure systems development”. In: *Proceedings of the 2002 ACM symposium on Applied computing.* ACM. 2002, pp. 1026–1030.

[114] J. Jürjens, L. Marchal, M. Ochoa, and H. Schmidt. “Incremental security verification for evolving UMLsec models”. In: *Modelling Foundations and Applications.* Springer, 2011, pp. 52–68.

[115] J. Jürjens and P. Shabalin. “Tools for secure systems development with UML”. In: *International Journal on Software Tools for Technology Transfer* 9.5-6 (2007), pp. 527–544.

[116] A. Kaddani, A. Baina, and L. Echabbi. “Towards a Model Driven Security for critical infrastructures using OrBAC”. In: *Multimedia Computing and Systems (ICMCS), 2014 International Conference on.* IEEE. 2014, pp. 1235–1240.

[117] A. Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin. “Organization based access control”. In: *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on.* 2003, pp. 120–131.

[118] K. Kasal, J. Heurix, and T. Neubauer. “Model-Driven Development Meets Security: An Evaluation of Current Approaches”. In: *Proceedings of the 44th Hawaii International Conference on System Sciences, HICSS-44 2011.* Kauai, Hawaii, USA: IEEE Computer Society, 2011, pp. 1–9.

[119] B. Katt, M. Gander, R. Breu, and M. Felderer. “Enhancing Model Driven Security through Pattern Refinement Techniques”. In: *Formal Methods for Components and Objects* Ffg 822740 (2013), pp. 169–183.

[120] A. A. Khwaja and J. E. Urban. “A Synthesis of Evaluation Criteria for Software Specifications and Specification Techniques”. In: *International Journal of Software Engineering and Knowledge Engineering* 12.05 (2002), pp. 581–599. eprint: http://www.worldscientific.com/doi/pdf/10.1142/S0218194002001062.

[121] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. “Aspect-Oriented Programming”. In: *Proceedings of the European Conference on Object-Oriented Programming*. Vol. 1241. Springer-Verlag, 1997, pp. 220–242.

[122] J. Kienzle, W. Al Abed, F. Fleurey, J.-M. Jézéquel, and J. Klein. “Aspect-Oriented Design with Reusable Aspect Models”. In: *Transactions on Aspect-Oriented Software Development VII*. Ed. by S. Katz, M. Mezini, and J. Kienzle. Vol. 6210. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pp. 272–320.

[123] J. Kienzle, W. Al Abed, and J. Klein. “Aspect-oriented multi-view modeling”. In: *Proceedings of the 8th ACM international conference on Aspect-oriented software development*. ACM. 2009, pp. 87–98.

[124] J. Kienzle, E. Duala-Ekoko, and S. Gélineau. “AspectOptima: A case study on aspect dependencies and interactions”. In: *Transactions on Aspect-Oriented Software Development V*. Springer, 2009, pp. 187–234.

[125] J. Kienzle, N. Guelfi, and S. Mustafiz. “Crisis Management Systems: A Case Study for Aspect-Oriented Modeling”. In: *Transactions on Aspect-Oriented Software Development VII*. Ed. by S. Katz, M. Mezini, and J. Kienzle. Vol. 6210. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 1–22.

[126] D.-K. Kim and P. Gokhale. “A Pattern-Based Technique for Developing UML Models of Access Control Systems”. In: *Computer Software and Applications Conference, 2006. COMPSAC ’06. 30th Annual International*. 2006, pp. 317–324.

[127] D.-K. Kim, I. Ray, R. France, and N. Li. “Modeling Role-Based Access Control Using Parameterized UML Models”. In: *Fundamental Approaches to Software Engineering*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 180–193.

[128] S. Kim, D.-K. Kim, L. Lu, S. Kim, and S. Park. “A feature-based approach for modeling role-based access control systems”. In: *Journal of Systems and Software* 84.12 (2011), pp. 2035–2052.

[129] B. Kitchenham. “Guidelines for performing systematic literature reviews in software engineering”. In: *EBSE Technical Report* (2007).

[130] B. Klatt. “Xpand: A Closer Look at the model2text Transformation Language”. In: *Language* 10/16/2008 (2007).

[131] J. Klein, F. Fleurey, and J. Jézéquel. “Weaving Multiple Aspects in Sequence Diagrams”. In: *Transactions on Aspect-Oriented Software Development (TAOSD).* Vol. 4620. Lecture Notes in Computer Science. Springer-Verlag, 2007, pp. 167–199.

[132] A. G. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley, 2003.

[133] T. Kobashi, N. Yoshioka, T. Okubo, H. Kaiya, H. Washizaki, and Y. Fukazawa. “Validating Security Design Patterns Application Using Model Testing”. In: *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on.* IEEE. 2013, pp. 62–71.

[134] M. E. Kramer, J. Klein, J. R. Steel, B. Morin, J. Kienzle, O. Barais, and J. Jézéquel. “Achieving Practical Genericity in Model Weaving through Extensibility”. In: *Proceedings of the 6th International Conference on the Theory and Practice of Model Transformations, ICMT 2013.* Vol. 7909. Lecture Notes in Computer Science. Budapest, Hungary: Springer-Verlag, 2013, pp. 108–124.

[135] B. W. Lampson. “Protection”. In: *SIGOPS Oper. Syst. Rev.* 8.1 (1974), pp. 18–24.

[136] U. Lang and R. Schreiner. “Model Driven Security Management: Making Security Management Manageable in Complex Distributed Systems”. In: *Proceedings of the 1st Workshop on Modeling Security, MODSEC 2008.* Vol. 413. Toulouse, France: CEUR Workshop Proceedings (CEUR-WS.org), 2008.

[137] J. de Lara and H. Vangheluwe. “AToM^3: A Tool for Multi-Formalism and Meta-Modelling”. In: *Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering, FASE 2002.* Lecture Notes in Computer Science. Grenoble, France: Springer-Verlag, 2002, pp. 174–188.

[138] Y. Le Traon, T. Mouelhi, and B. Baudry. “Testing security policies: going beyond functional testing”. In: *in Proceedings of 18th IEEE international symposium on Software Reliability. ser ISSRE* (2007).

[139] Y. Le Traon, T. Mouelhi, A. Pretschner, and B. Baudry. “Test-Driven Assessment of Access Control in Legacy Applications”. In: *Software Testing, Verification, and Validation, 2008 1st International Conference on* (2008), pp. 238–247.

[140] D. LeBlanc and M. Howard. *Writing secure code.* Pearson Education, 2002.

[141] K. Li, L. Mounier, and R. Groz. “Test Generation from Security Policies Specified in Or-BAC”. In: *Computer Software and Applications Conference, 2007. COMPSAC 2007, 31st Annual International* 2 (2007), pp. 255–260.

[142] V. B. Livshits and M. S. Lam. “Finding security errors in Java programs with static analysis”. In: *In Proceedings of the 14th Usenix Security Symposium* (August,2005).

[143] T. Lodderstedt, D. Basin, and J. Doser. “SecureUML: A UML-Based Modeling Language for Model-Driven Security”. In: *UML 2002 – The Unified Modeling Language*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 426–441.

[144] L. Lucio, Q. Zhang, P. H. Nguyen, M. Amrani, J. Klein, H. Vangheluwe, and Y. Le Traon. *Advances in Model-Driven Security*. Ed. by A. Hurson and A. Memon. Elsevier, 2014.

[145] N. MacDonald. *Model-Driven Security: Enabling a Real-Time, Adaptive Security Infrastructure*. Tech. rep. Gartner Inc., 2007.

[146] E. Martin and T. Xie. “A Fault Model and Mutation Testing of Access Control Policies”. In: *International World Wide Web Conference Committee* (2007).

[147] E. Martin and T. Xie. “Automated Test Generation for Access Control Policies via Change-Impact Analysis”. In: *Third International Workshop on Software Engineering for Secure Systems , SESS’07: ICSE Workshops* (2007).

[148] A. Masood, R. Bhatti, A. Ghafoor, and A. Mathur. “Scalable and Effective Test Generation for Role-Based Access Control Systems”. In: *IEEE Transactions on Software Engineering* 35,5 (2009), pp. 654–668.

[149] G. McGraw. *Software security: building security in*. Vol. 1. Addison-Wesley Professional, 2006.

[150] M. Memon, G. D. Menghwar, M. H. Depar, A. A. Jalbani, and W. M. Mashwani. “Security modeling for service-oriented systems using security pattern refinement approach”. In: *Software & Systems Modeling* 13.2 (2014), pp. 549–572.

[151] T. Mens and P. Van Gorp. “A taxonomy of model transformation”. In: *Electronic Notes in Theoretical Computer Science* 152 (2006), pp. 125–142.

[152] M. Menzel and C. Meinel. “A Security Meta-model for Service-Oriented Architectures”. In: *2009 IEEE International Conference on Services Computing* (2009), pp. 251–259.

[153] M. Menzel and C. Meinel. “SecureSOA Modelling Security Requirements for Service-Oriented Architectures”. In: *Services Computing (SCC), 2010 IEEE International Conference on* (2010), pp. 146–153.

[154] M. Menzel, R. Warschofsky, and C. Meinel. “A pattern-driven generation of security policies for service-oriented architectures”. In: *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE. 2010, pp. 243–250.

[155] Metacase. *Domain-Specific Modeling with MetaEdit+: 10 times faster than UML.* White Paper. 2009.

[156] N. Moebius, K. Stenzel, H. Grandy, and W. Reif. “SecureMDD: A Model-Driven Development Method for Secure Smart Card Applications”. In: *Availability, Reliability and Security, 2009. ARES ’09. International Conference on*. 2009, pp. 841–846.

[157] N. Moebius, K. Stenzel, and W. Reif. “Generating Formal Specifications for Security-Critical Applications - A Model-Driven Approach”. In: *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems, IWSESS 2009*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 68–74.

[158] N. Moebius, W. Reif, and K. Stenzel. “Modeling Security-Critical Applications with UML in the SecureMDD Approach”. In: *International Journal On Advances in Software* 1.1 (2009), pp. 59–79.

[159] N. Moebius and K. Stenzel. “Model-Driven Code Generation for Secure Smart Card Applications”. In: *Software Engineering Conference, 2009. ASWEC’09. Australian.* (2009), pp. 44–53.

[160] N. Moebius, K. Stenzel, M. Borek, and W. Reif. “Incremental Development of Large, Secure Smart Card Applications”. In: *Proceedings of the 1st International Workshop on Model-Driven Security*. Innsbruck, Austria: ACM, 2012, pp. 1–6.

[161] N. Moebius, K. Stenzel, and W. Reif. “Formal Verification of Application-Specific Security Properties in a Model-Driven Approach”. In: *Engineering Secure Software and Systems*. Vol. 5965. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 166–181.

[162] W. Moore, D. Dean, A. Gerber, G. Wagenknecht, and P. Vanderheyden. *Eclipse Development Using the Graphical Editing Framework and the Eclipse Modeling Framework*. IBM Redbooks, 2004.

[163] S. Moral-Garcia, S. Moral-Rubio, E. B. Fernandez, and E. Fernandez-Medina. “Enterprise security pattern: A model-driven architecture instance”. In: *Computer Standards & Interfaces* 36.4 (2014), pp. 748–758.

[164] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg. “Models@ Run.time to Support Dynamic Adaptation”. In: *Computer* 42.10 (2009), pp. 44–51.

[165] B. Morin, O. Barais, G. Nain, and J. Jézéquel. "Taming Dynamically Adaptive Systems with Models and Aspects". In: *Proceedings of the 31st International Conference on Software Engineering, ICSE 2009.* Vancouver, Canada: IEEE Computer Society, 2009, pp. 122–132.

[166] B. Morin, F. Fleurey, N. Bencomo, J.-M. Jézéquel, A. Solberg, V. Dehlen, and G. Blair. "An Aspect-Oriented and Model-Driven Approach for Managing Dynamic Variability". In: *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems.* MoDELS '08. Toulouse, France: Springer-Verlag, 2008, pp. 782–796.

[167] B. Morin, T. Mouelhi, F. Fleurey, Y. Le Traon, O. Barais, and J.-M. Jézéquel. "Security-driven model-based dynamic adaptation". In: *Proceedings of the IEEE/ ACM international conference on Automated software engineering.* ASE '10. ACM, 2010, pp. 205–214.

[168] P. J. Mosterman and H. Vangheluwe. "Computer Automated Multi-Paradigm Modeling: An Introduction". In: *Simulation.* Vol. 80. SAGE Publications, 2004, pp. 433–450.

[169] T. Mouelhi, F. Fleurey, B. Baudry, and Y. L. Traon. "A model-based framework for security policy specification, deployment and testing". In: *Model Driven Engineering Languages and Systems* 1 (2008), pp. 537–552.

[170] T. Mouelhi, Y. Le Traon, and B. Baudry. "Transforming and Selecting Functional Test Cases for Security Policy Testing". In: *Proceedings of the 2009 International Conference on Software Testing Verification and Validation.* ICST '09. IEEE Computer Society, 2009, pp. 171–180.

[171] D. Mouheb, C. Talhi, M. Nouh, V. Lima, M. Debbabi, L. Wang, and M. Pourzandi. "Aspect-Oriented Modeling for Representing and Integrating Security Concerns in UML". In: *Software Engineering Research, Management and Applications 2010.* Springer Berlin Heidelberg, 2010, pp. 197–213.

[172] D. Mouheb, C. Talhi, A. Mourad, and V. Lima. "An Aspect-Oriented Approach for Software Security Hardening: from Design to Implementation." In: *SoMeT* (2009).

[173] H. Mouratidis and P. Giorgini. "Secure tropos: a security-oriented extension of the tropos methodology". In: *International Journal of Software Engineering and Knowledge Engineering* 17.02 (2007), pp. 285–309.

[174] H. Mouratidis, J. Jürjens, and J. Fox. "Towards a comprehensive framework for secure systems development". In: *Advanced information systems engineering* (2006), pp. 48–62.

[175] H. Mouratidis, A. Sunyaev, and J. Jürjens. “Secure information systems engineering: experiences and lessons learned from two health care projects”. In: *Advanced Information Systems Engineering* (2009), pp. 231–245.

[176] P. Muller, F. Fleurey, and J. Jézéquel. “Weaving Executability into Object-Oriented Meta-Languages”. In: *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems*. Vol. 3713. Lecture Notes in Computer Science. Half Moon Resort, Montego Bay, Jamaica: Springer-Verlag, 2005, pp. 264–278.

[177] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. “Weaving executability into object-oriented meta-languages”. In: *International Conference on Model Driven Engineering Languages and Systems (MoDELS), LNCS 3713*. Springer, 2005, pp. 264–278.

[178] S. Na and S. Cheon. “Role delegation in role-based access control”. In: *Proceedings of the fifth ACM workshop on Role-based access control*. RBAC ’00. Berlin, Germany: ACM, 2000, pp. 39–44.

[179] Y. Nakamura and M. Tatsubori. “Model-driven security based on a web services security architecture”. In: *Services Computing, 2005 IEEE International Conference on* (2005).

[180] R. Neisse and J. Doerr. “Model-based specification and refinement of usage control policies”. In: *2013 Eleventh Annual Conference on Privacy, Security and Trust* (2013), pp. 169–176.

[181] P. H. Nguyen, J. Klein, and Y. Le Traon. “Model-Driven Security with A System of Aspect-Oriented Security Design Patterns”. In: *Proceedings of the 2Nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*. VAO ’14. York, United Kingdom: ACM, 2014, 51:51–51:54.

[182] P. H. Nguyen, J. Klein, Y. Le Traon, and M. E. Kramer. “A Systematic Review of Model-Driven Security”. In: *Software Engineering Conference (APSEC, 2013) 20th Asia-Pacific*. Vol. 1. IEEE. 2013, pp. 432–441.

[183] P. H. Nguyen, M. E. Kramer, J. Klein, and Y. Le Traon. “An Extensive Systematic Review on the Model-Driven Development of Secure Systems”. In: *Information & Software Technology* 68 (2015), pp. 62–81.

[184] P. H. Nguyen, G. Nain, J. Klein, T. Mouelhi, and Y. Le Traon. “Modularity and Dynamic Adaptation of Flexibly Secure Systems: Model-Driven Adaptive Delegation in Access Control Management”. In: *Transactions on Aspect-Oriented Software Development XI*. Springer, 2014, pp. 109–144.

[185] P. H. Nguyen, K. Yskout, T. Heyman, J. Klein, R. Scandariato, and Y. Le Traon. "SoSPa: A System of Security Design Patterns for Systematically Engineering Secure Systems". In: *International Conference on Model Driven Engineering Languages and Systems.* MODELS '15. Ottawa, Canada, 2015.

[186] P. H. Nguyen. "Model-Driven Security based on A Unified System of Security Design Patterns". In: (2015).

[187] P. H. Nguyen. "Model-Driven Security with Modularity and Reusability for Secure Systems Development". In: *STAF-DS 2014.* 2014.

[188] P. H. Nguyen. "Quantitative Analysis of Model Transformations". MA thesis. Eindhoven University of Technology, 2010.

[189] P. H. Nguyen, J. Klein, and Y. Le Traon. "Modeling, composing, and testing of security concerns in a Model-Driven Security approach". In: *International Symposium on Engineering Secure Software and Systems-Doctoral Symposium.* 2014.

[190] P. H. Nguyen, G. Nain, J. Klein, T. Mouelhi, and Y. Le Traon. "Model-Driven Adaptive Delegation". In: *Proceedings of the 12th annual International Conference on Aspect-Oriented Software Development.* Modularity:AOSD '13. Fukuoka, Japan: ACM, 2013, pp. 61–72.

[191] P. H. Nguyen, M. Papadakis, and I. Rubab. "Testing Delegation Policy Enforcement via Mutation Analsysis". In: *Proceedings of the Workshop on Mutation Testing @ the Sixth IEEE International Conference on Software Testing.* ICST'13. Luxembourg, Luxembourg: IEEE, 2013, pp. 61–72.

[192] E. R. O'connell. *Automated password reset.* US Patent 5,991,882. 1999.

[193] J. Offutt. "A mutation carol: Past, present and future". In: *Information & Software Technology* 53.10 (2011), pp. 1098–1107.

[194] M. Papadakis and N. Malevris. "Automatically performing weak mutation with the aid of symbolic execution, concolic testing and search-based testing". In: *Software Quality Journal* 19.4 (2011), pp. 691–723.

[195] M. Papadakis and N. Malevris. "Mutation based test case generation via a path selection strategy". In: *Information & Software Technology* 54.9 (2012), pp. 915–932.

[196] M. Papadakis and Y. L. Traon. "Using Mutants to Locate "Unknown" Faults". In: *ICST.* 2012, pp. 691–700.

[197] T. J. Parr and R. W. Quong. "ANTLR: A predicated-LL (k) parser generator". In: *Software: Practice and Experience* 25.7 (1995), pp. 789–810.

[198] J. Pavlich-Mariscal, S. Demurjian, and L. Michel. “A framework of composable access control features: Preserving separation of access control concerns from models to code”. In: *Computers & Security* 29.3 (2010), pp. 350–379.

[199] R. Prieto-Diaz. “Status report: Software reusability”. In: *software, IEEE* 10.3 (1993), pp. 61–66.

[200] I. Ray, R. France, N. Li, and G. Georg. “An aspect-based approach to modeling access control concerns”. In: *Information and Software Technology* 46.9 (2004), pp. 575–587.

[201] E. Rodriguez. “Security Design Patterns”. In: *ACSAC’03*. 2003.

[202] D. Rubio. *Pro Spring dynamic modules for OSGi service platforms*. 2009.

[203] Ó. Sánchez and F. Molina. “ModelSec : A Generative Architecture for Model-Driven Security”. In: *Journal of Universal Computer Science* 15.15 (2009), pp. 2957–2980.

[204] P. Sánchez, A. Moreira, and L. Fuentes. “Model-driven development for early aspects”. In: *Information and Software Technology* 52.3 (2010), pp. 249–273.

[205] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. “Role-Based Access Control Models”. In: *Computer* 29.2 (1996), pp. 38–47.

[206] R. Sandhu and J. Park. “Usage Control: A Vision for Next Generation Access Control”. In: *V. Gorodetsky et al. (Eds.): MMM-ACNS 2003, LNCS 2776* 1 (2003), pp. 17–31.

[207] F. Satoh, Y. Nakamura, and K. Ono. “Adding Authentication to model driven security”. In: *Web Services, 2006. ICWS’06. International Conference on* (2006), pp. 585–594.

[208] F. Satoh and Y. Yamaguchi. “Generic Security Policy Transformation Framework for WS-Security”. In: *IEEE International Conference on Web Services (ICWS 2007)* Icws (2007), pp. 513–520.

[209] S. Schefer-Wenzl and M. Strembeck. “Model-driven specification and enforcement of RBAC break-glass policies for process-aware information systems”. In: *Information and Software Technology* 56.10 (2014), pp. 1289–1308.

[210] M. Schnjakin, M. Menzel, and C. Meinel. “A pattern-driven security advisor for service-oriented architectures”. In: *Proceedings of the 2009 ACM workshop on Secure web services - SWS ’09* (2009), p. 13.

[211] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, 2013.

[212] S. Sendall and W. Kozaczynski. “Model Transformation: The Heart and Soul of Model-Driven Software Development”. In: *IEEE Software*. Vol. 20. IEEE Computer Society, 2003, pp. 42–45.

[213] Y. Shiroma, H. Washizaki, Y. Fukazawa, A. Kubo, and N. Yoshioka. “Model-Driven Security Patterns Application Based on Dependences among Patterns”. In: *Availability, Reliability, and Security, 2010. ARES ’10 International Conference on.* 2010, pp. 555–559.

[214] D. Simmonds, A. Solberg, R. Reddy, R. France, and S. Ghosh. “An aspect oriented model driven framework”. In: *EDOC Enterprise Computing Conference, 2005 Ninth IEEE International.* IEEE. 2005, pp. 119–130.

[215] K. Sohr, T. Mustafa, X. Bao, and G.-J. Ahn. “Enforcing Role-Based Access Control Policies in Web Services with UML and OCL”. In: *2008 Annual Computer Security Applications Conference (ACSAC)* (2008), pp. 257–266.

[216] E. Soler, J. Trujillo, E. Fernandez-Medina, and M. Piattini. “A Framework for the Development of Secure Data Warehouses based on MDA and QVT”. In: *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.* 2007, pp. 294–300.

[217] E. Soler, J. Trujillo, E. Fernandez-Medina, and M. Piattini. “A set of QVT relations to transform PIM to PSM in the Design of Secure Data Warehouses”. In: *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.* 2007, pp. 644–654.

[218] E. Soler, J. Trujillo, E. Fernandez-Medina, and M. Piattini. “Application of QVT for the Development of Secure Data Warehouses: A case study”. In: *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.* Vol. 3. 1. 2007, pp. 829–836.

[219] E. Soler, J. Trujillo, C. Blanco, and E. Fernández-Medina. “Designing Secure Data Warehouses by Using MDA and QVT”. In: *J. UCS* 15.8 (2009), pp. 1607–1641.

[220] C. Steel and R. Nagappan. *Core Security Patterns: Best Practices and Strategies for J2EE”, Web Services, and Identity Management.* Pearson Education India, 2006.

[221] E. Syriani. “A Multi-Paradigm Foundation for Model Transformation Language Engineering”. PhD thesis. McGill University, Canada, 2011.

[222] O. The OSGi Alliance. “OSGi Service Platform Core Specification, Release 4.1”. In: (2007).

[223] X. Thirioux, B. Combemale, X. Crégut, and P.-L. Garoche. “A Framework to Formalise the MDE Foundations”. In: *Proceedings of the 1st International Workshop on Towers of Models, TOWERS 2007*. Zürich, Switzerland: Springer-Verlag, 2007, pp. 14–30.

[224] J. Trujillo, E. Soler, E. Fernández-Medina, and M. Piattini. “A UML 2.0 profile to define security requirements for Data Warehouses”. In: *Computer Standards & Interfaces* 31.5 (2009), pp. 969–983.

[225] J. Trujillo, E. Soler, E. Fernández-Medina, and M. Piattini. “An engineering process for developing Secure Data Warehouses”. In: *Information and Software Technology* 51.6 (2009), pp. 1033–1051.

[226] M. Utting, A. Pretschner, and B. Legeard. “A taxonomy of model-based testing approaches”. In: *Software Testing, Verification and Reliability* 22.5 (2012), pp. 297–312.

[227] A. V. Uzunov, E. B. Fernandez, and K. Falkner. “Engineering Security into Distributed Systems: A Survey of Methodologies”. In: *Journal of Universal Computer Science* 18.20 (1, 2012), pp. 2920–3006.

[228] B. Vela, J. N. Mazón, C. Blanco, E. Fernández-medina, J. Trujillo, and E. Marcos. “Development of Secure XML Data Warehouses with QVT”. In: *Information and Software Technology* 55.9 (2013), pp. 1651–1677.

[229] B. Vela, C. Blanco, E. Fernández-Medina, and E. Marcos. “A practical application of our MDD approach for modeling secure XML data warehouses”. In: *Decision Support Systems* 52.4 (2012), pp. 899–925.

[230] B. Vela, C. Blanco, E. Fernández-Medina, and E. Marcos. “Model driven development of secure XML data warehouses: a case study”. In: *Proceedings of the 2010 EDBT/ICDT Workshops*. ACM. 2010, p. 10.

[231] B. Vela, E. Fernández-Medina, E. Marcos, and M. Piattini. “Model driven development of secure XML databases”. In: *ACM SIGMOD Record* 35.3 (2006), pp. 22–27.

[232] R. Villarroel. “A UML 2.0/OCL extension for designing secure data warehouses”. In: *Journal of Research and Practice in Information Technology* 38.1 (2006).

[233] H. Wada, J. Suzuki, and K. Oba. “A model-driven development framework for non-functional aspects in service oriented architecture”. In: *Web Services Research for Emerging Applications: Discoveries and Trends: Discoveries and Trends* X (2008), pp. 1–31.

[234] J. Whittle, P. K. Jayaraman, A. M. Elkhodary, A. Moreira, and J. Araújo. "MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation". In: *Transactions on Aspect-Oriented Software Development VI*. Vol. 5560. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 191–237.

[235] C. Wohlin and R. Prikladnicki. "Systematic literature reviews in software engineering". In: *Information and Software Technology* 55.6 (2013), pp. 919–920.

[236] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel. "Model-driven business process security requirement specification". In: *Journal of Systems Architecture* 55.4 (2009), pp. 211–223.

[237] L. Xiao. "An adaptive security model using agent-oriented MDA". In: *Information and Software Technology* 51.5 (2009), pp. 933–955.

[238] D. Xu, L. Thomas, M. Kent, T. Mouelhi, and Y. Le Traon. "A Model-Based Approach to Automated Testing of Access Control Policies". In: *SACMAT* (2012).

[239] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen. *A system of security patterns*. Technical report, KU Leuven. 2006.

[240] K. Yskout, R. Scandariato, and W. Joosen. "Do Security Patterns Really help Designers?" In: *Software Engineering (ICSE), 2015 37th International Conference on*. IEEE. 2015.

[241] K. Yskout, R. Scandariato, and W. Joosen. "Does organizing security patterns focus architectural choices?" In: *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE. 2012, pp. 617–627.

[242] X. Zhang, S. Oh, and R. Sandhu. "PBDM: a flexible delegation model in RBAC". In: *Proceedings of the eighth ACM symposium on Access control models and technologies*. SACMAT '03. Como, Italy: ACM, 2003, pp. 149–157.

[243] Z. J. Zhu and M. Zulkernine. "A model-based aspect-oriented framework for building intrusion-aware software systems". In: *Information and Software Technology* 51.5 (2009), pp. 865–875.