2010

Quantifiable non-functional requirements modeling and static verification for web service compositions

Hongyu Sun
Iowa State University

2010

Quantifiable non-functional requirements modeling and static verification for web service compositions

Hongyu Sun
Iowa State University

Quantifiable non-functional requirements modeling and static verification for web service compositions

by

Hongyu Sun

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Robyn R. Lutz, Major Professor
Samik Basu
Carl K. Chang
Ratnesh Kumar
Hridesh Rajan

Iowa State University

Ames, Iowa

2010

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

My five-and-a-half years of Ph.D. experience have been an unforgettable journey on the road to learning and self-discovery. It has been a great deal more than exploring research papers and academic journals and experimenting with new ideas. What I have gained is a deep understanding of fundamental concepts of computer science and a broad view of the state-of-the-art computer techniques. Beyond computer science, the experience has sharpened my ability to observe and think about the inner logics of everyday phenomena. I can say that a Ph.D. goes far beyond the boundaries of individual disciplines. It is a lifetime irreplaceable treasure. I owe gratitude to my professors, classmates, friends and family members who helped me through this journey.

First of all, I owe my deepest sense of gratitude to my advisor Dr. Robyn R. Lutz. I was fortunate to be Robyn's student after my college graduation and to receive her direction through my Ph.D. studies. Over the years, Robyn gave me continuous support for my research efforts while allowing the freedom to explore my own ideas. She taught me ways of conducting research in computer science and helped me identify problems and flaws in my work. Her insightful reviews and advice helped me overcome many difficulties, including language barriers, fundamental computer science concepts, the progress to a rigorous research altitude, choosing the right research directions, choosing

the relevant background reading materials and much more that cannot be listed here. I am truly thankful to her for having invested so much time and energy in me as a graduate student. I have learned from her patience, rigorousness and kindness and hope to pass these qualities down to my students or employees in the future.

. I am deeply grateful for all my POS members and research group members: Dr. Samik Basu, Dr. Carl K. Chang, Dr. Ratnesh Kumar, Dr. Hridesh Rajan and Dr. Vasant Honavar who reviewed my work frequently, and gave advice and guidance for dissertation writing. Their insightful questions significantly benefited the rigorousness of this dissertation.

In particular, Dr. Samik Basu's recommendations and suggestions have been invaluable for design of simulations and for improving the dissertation in general.

I also would like to thank my past and current classmates in the Laboratory for Software Safety: Sandeep Krishnan, Jingwei Yang, Wei Zhang, Jonathan Schroeder, Jing (Janet) Liu, Josh Dehlinger, Dinanath Nadkarni and Qian Feng. I enjoyed and still cherish the discussions and the harmonious working environment in the research group. Discussing with them various research topics enlightened me on many research problems and ideas.

In particular, I would like to thank the senior members of the laboratory: Josh Dehlinger and Jing (Janet) Liu. I shall never forget the guidance and help they provided when I first came to the United States and started my graduate studies.

I owe a great deal of gratitude to my parents, without whom I couldn't be who I am. Thanks to my father, Yuangang Sun, who urged me to pursue a Ph. D degree when I graduated from the college. He always gave me wise advice at each turn of my life and has helped me set life and career goals. His support has given me enormous mental strength, which allows me to challenge myself to keep flying higher and higher, without worrying about falling down. Thanks to my mother, Guiru Xian, who has cared for me every day in every detail. My mother, in order to talk with me online, learned to use computer in her 50s. When I was busy and didn't have time to talk with her on the IM, she said, "Silence is also fine. Every time I log in, if you are online, I assume you are safe and well. Then I feel safe too." During my graduate studies, her encouragement gave me strength to conquer the hardships I encountered. Her smile is like a static picture in my heart, which protects me from darkness and loneliness whenever I work late hours.

Finally, this research was supported through National Science Foundation grants 0541163 "Safety Analysis of Evolving Product Lines", 0702758 "Interactive and Verifiable Composition of Web Services To Satisfy End-User Goals" and 0916275 "Evidence-based Reliability Assessment of Software Product Lines", the latter with funds from the American Recovery and Reinvestment Act of 2009.

ABSTRACT

As service oriented architectures have become more widespread in industry, many complex web services are assembled using independently developed modular services from different vendors. Although the functionalities of the composite web services are ensured during the composition process, the non-functional requirements (NFRs) are often ignored in this process. Since quality of services plays a more and more important role in modern service-based systems, there is a growing need for effective approaches to verifying that a composite web service not only offers the required functionality but also satisfies the desired NFRs.

Current approaches to verifying NFRs of composite services (as opposed to individual services) remain largely ad-hoc and informal in nature. This is especially problematic for high-assurance composite web services. High-assurance composite web services are those composite web services with special concern on critical NFRs such as security, safety and reliability. Examples of such applications include traffic control, medical decision support and the coordinated response systems for civil emergencies. The latter serves to motivate and illustrate the work described here.

In this dissertation we develop techniques for ensuring that a composite service meets the user-specified NFRs expressible as hard constraints, e.g., “the messages of

particular operations must be authenticated." We introduce an automata-based framework for verifying that a composite service satisfies the desired NFRs based on the known guarantees regarding the non-functional properties of the component services. This automata-based model is able to represent NFRs that are hard, quantitative constraints on the composite web services. This model addresses two issues previously not handled in the modeling and verification of NFRs for composite web services: (1) the scope of the NFRs and (2) consistency checking of multiple NFRs.

A scope of a NFR on a web service composition is the effective range of the NFR on the sub-workflows and modular services of the web service composition. It allows more precise description of a NFR constraint and more efficient verification. When multiple NFRs exist and overlap in their scopes, consistency checking is necessary to avoid wasted verification efforts on conflicting constraints. The approach presented here captures scope information in the model and uses it to check the consistency of multiple NFRs prior to the static verification of web service compositions. We illustrate how our approach can be used to verify security requirements for an Emergency Management System.

We then focus on families of highly-customizable, composed web services where repeated verification of similar sets of NFRs can waste computation resources. We introduce a new approach to extend software product line engineering techniques to the web service composition domain. The resulting technique uses a partitioning similar to that between domain engineering and application engineering in the product-line context. It specifies the options that the user can select and constructs the resulting web service

compositions. By first creating a web-service composition search space that satisfies the common requirements and then querying the search space as the user makes customization decisions, the technique provides a more efficient way to verify customizable web services. A decision model, illustrated with examples from the emergency-response application, is created to interact with the customers and ensure the consistency of their specifications. The capability to reuse the composition search space is shown to improve the quality of the composite services and reduce the cost of re-verifying the same compositions. By distinguishing the commonalities and the variabilities of the web services, we divide the web composition into two stages: the preparation stage (to construct all commonalities) and the customization stage (to choose optional and alternative features). We thus draw most of the computation overhead into the first stage during the design in order to enable improved runtime efficiency during the second stage.

A simulation platform was constructed to conduct experiments on the two verification approaches and three strategies introduced in this dissertation. The results of these experiments were analyzed to show the advantage of our automaton-based model in its verification efficiency with scoping information. We have shown how to choose the most efficient verification strategy from the three strategies of verifying multiple NFRs introduced in this dissertation under different circumstances. The results indicate that the software product line approach has significant efficiency improvement over traditional on-demand verification for highly customizable web service compositions.

CHAPTER 1. INTRODUCTION

As service oriented architectures gain more popularity in industry, most existing practical mechanisms for synthesizing composite services have been deployed taking into consideration their desired functional requirements [14][28]. Functional requirements (FRs) describe how a system should behave during operation, while non-functional requirements (NFRs) describe constraints on the quality attributes of the system's operation [13]. NFRs can be broadly classified as soft and hard constraints. A constraint is a restriction on the degree of freedom in developing a software system [1]. Hard constraints refer to the NFRs that must be satisfied by a composite service, while soft constraints deal with user preferences and trade-offs among NFRs [13]. The soft constraints can only be satisficed, and sometimes the values for a soft constraint can be only partially ordered [52]. Our research focuses on hard constraints and quantifiable NFRs (i.e., quality of service (QoS) attributes). Assurance of NFRs in our research means the satisfaction and fulfillment of all the hard constraints specified as NFRs in the composed web services.

In contrast to modeling techniques for functional requirements which are well-developed, most of the non-functional requirements for composite web services can still be represented only on the modular service level rather than over the entire composition. The desire for a non-functional requirements model and verification method for

composite web services has never been fulfilled, since existing approaches to modeling and verifying non-functional requirements for composite web services are limited either in their representational power or their verification efficiency. Commercialized models [63] for non-functional requirements typically use simple logics that are easy to use and verify, such as WS-Security etc. The techniques proposed in research work [14][28] are more complex and advanced in their representational power, but less efficient for industrial use.

We identified two issues in non-functional requirements modeling for composed web services that can improve the modeling and efficient verification of NFRs: (1) scoping of non-functional requirements constraints and (2) consistency checking for multiple NFRs. A scope of a NFR constraint on a web service composition is the effective range of the NFR constraint on the sub-workflows and modular services of the web service composition. It allows more precise description of a NFR constraint. When multiple NFRs constraints exist and overlap in their scopes, a consistency check is necessary to avoid wasting verification resources on conflicting constraints. In order to precisely define the scope of our research, in this dissertation we mainly focus on the quality of service (QoS) in NFR.

Accordingly, our research is to develop an automata-based model to represent QoS constraints for composite web services, such as security, that takes the following two problems into consideration:

1. The ability to represent the scoping information of the constraints
2. The ability to conduct the consistency checking of multiple constraints.

To solve the above problems, we develop an algorithm to verify the QoS constraints within their scopes and introduce a strategy to handle consistency checking of multiple QoS constraints.

As we can see, these two problems focus on how to verify a single QoS constraint on a web service composition (WSC) and how to verify multiple QoS constraints on a WSC. The next step was to focus on how to improve the constraint verification on multiple WSCs.

In modern web industry, commercial web services often have a very large user base. How to customize these web services according to the users' individual requirements becomes increasingly difficult as the number of users increases. One problem during composite web service deployment is the frequent requirements customization offered to, or desired by, the users. In order to customize a web service in a specific domain application (e.g., a travel agency service), FRs and NFRs may both vary somewhat among individuals. With existing methods of composing web services on FRs and NFRs [28][59], the repeated verification of the slight variations in the services tends to incur a heavy overhead. Even with an efficient verification algorithm for newly composed services, there still is waste of computational resources from verifying the same set of customized requirements repeatedly when users make similar choices. For a specific domain, the verification results for the valid sets of the customizable user requirements can be reused for greater efficiency. To pursue the reusability of the composite web service verification results, the third problem we address is:

3. To extend product line engineering techniques to enable reuse of some verification results for highly-customizable web service compositions. The product line engineering techniques can distinguish which NFRs on the service compositions are shared among all users and which vary across different users. By identifying this separation, a reuse-oriented process can be designed for efficiently verifying the web service compositions.

To summarize, our work is to introduce a non-functional requirements model that incorporates scope information and enables consistency checking among multiple NFRs, and a lower-overhead, product-line-based solution to generate verified web service compositions satisfying users' customized requirements for composite web services.

This chapter begins with the motivation and targets of our research, continues with a brief introduction of our solutions to the problems, and finally describes the contribution of this work. To solve our research problems, first, an automaton-based NFRs model for composite web service is introduced. Then the verification algorithm of this NFRs model is presented. Three different strategies of verifying multiple NFRs are designed and compared to allow a more efficient verification process. To reuse the NFRs verification results across multiple WSCs, the software product-line based engineering process are extended into the web service composition verification domain.

This chapter concludes by providing the outline of my dissertation and of its research contributions.

1.1 Automata-Based Verification of Security Requirements of Composite Web Services

Our overall approach to assembling a composite service that satisfies both the functional and non-functional requirements (in the case of security related NFRs) is shown in Figure 1. The functional composition algorithm [45] uses a goal model in the form of an automaton to encode the user-specified FR. A composite service is represented as an automaton where states represent the component services participating in the composition and inter-state transitions represent composition of the corresponding component services. The functional composition algorithm assembles a composite service that verifiably satisfies the FRs by exploring the space of candidate compositions.

The focus of this chapter, however, is on verifying that a composite service assembled by the functional composition algorithm also satisfies the NFR. We use an automaton to represent the non-functional requirements of a composite service. Labels on transitions of the NFR automaton encode the set of non-functional constraints that are satisfied by the services that are connected by the respective transitions.

A composite service is said to satisfy a given set of NFRs if and only if the sequence of properties over non-functional constraints represented by the NFR-automata is also realized by the automaton that encodes the composite service. A composite service conforming to desired NFRs is obtained by computing and verifying the product of the automaton representing the composite web service and the automaton representing the corresponding NFR. This lifts the NFRs analysis from the level of individual services to

the level of the search space of candidate compositions obtained from the functional requirements.

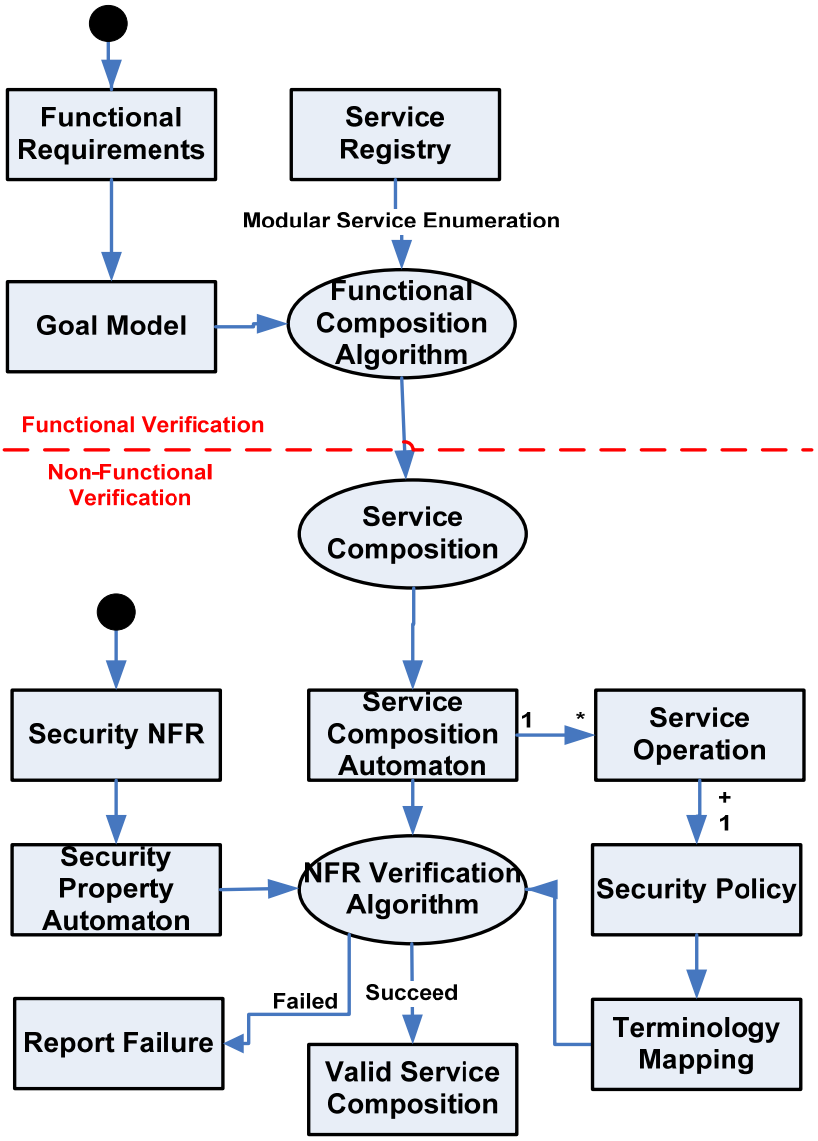

Figure 1. Overview of non-functional requirements verification process for web service composition

The non-functional properties, e.g., security policies, of the component services are extracted from their WSDL specifications. A domain terminology mapping table is used to translate the user-specified NFR, e.g., "messages shall be highly-encrypted" into the terminology used in the WSDL specification ("messages must use standard Basic128Sha256 encryption algorithm with a 256-bit key").

The verification of a composite web service consists of two main parts, as shown in Figure 1. The first part is the functional composition of the component web services. This is prior work by Pathak, Basu and Honavar [45]. The goal model is an automaton constructed to model the functionalities of the required system. With the goal model as input, an iterative algorithm generates a composite web service that verifiably satisfies the functional requirements.

The second part of the process, and the contribution of our work, is the verification of the security non-functional requirements, shown in the bottom half of the diagram. The security NFRs are modeled in an automaton-based model. The composite web service derived from the first step can then be verified against this security automaton. During the verification, the security policies of the service operations are pulled from the appropriate WSDL files. The domain terminology mapping table is used to translate the security policies into an algorithm-readable input and to provide the algorithm with the aggregation rules needed to handle security NFRs.

It must be noted that the focus of this approach is on static verification of NFR. Verification of NFRs whose satisfaction is contingent on certain runtime constraints on the performance of the component services being met (e.g., whether an ambulance

reaches its destination in time to meet the response time NFR) that can only be dynamically verified at runtime are beyond the scope of this research.

Chapter 4 will introduce this automaton-based NFRs model and its verification in detail. A case study of this approach on the Emergency Management System (EMS) is conducted as evaluation.

The next section uses this NFRs model and verification techniques as fundamental work and introduce the verification process of multiple WSCs by extending software product line engineering concepts and techniques.

1.2 Product-Line-Based Requirements Customization for Web Service Compositions

The non-functional requirements are customizable for mass-user based web applications. Software product line engineering offers a way to reduce the on-demand user requirements verification on newly constructed composite web services [58]. Possibly customizable functional and non-functional requirements are organized into variabilities in a service product line. The functional and non-functional requirements are pre-verified for future use. The verification results are stored and reused when different customers need to verify the same web service compositions. This work improves the customization efficiency of domain-specific, mass-user based, customizable web services. By constructing a decision model [64][65], we can hide the complexity of domain and application knowledge from the user and give the user a trouble-free way of generating a customized web service. By distinguishing the commonalities and the variabilities of the

web services, we can successfully divide the web composition into two stages: the preparation stage (to construct all commonalities) and the customization stage (to set all variabilities).

Figure 2 shows an overview of this product-line engineering based approach of verifying NFRs for multiple WSCs, which is described briefly here. The steps are labeled in the figure with the number of the subsection that describes that step.

Our approach contains two workflows, one for the web service product developer (shown at the top of the figure) and one for the user (shown at the bottom). This approach provides a two-stage process. From the point of view of product line development, the development process is divided into the domain engineering phase and the application engineering phase. From the point of view of web-service design, the process is divided into the preparation stage (before the construction of the search space) and the customization stage (after the construction of the search space).

At the top half of the figure, i.e., the domain engineering phase, the developer performs a commonality and variability analysis (CVA) of the system requirements. The results of the CVA include a formalized specification of all the common and variable functional and non-functional properties. Each variability has some associated parameters to configure, called parameters of variation. Some of these parameters may have dependencies or tradeoffs with other parameters of variation [65]. We model the dependencies among variations in a variability dependency graph.

Each commonality and parameter of variation is also associated with a set of service compositions that satisfy them. We call this relation the mapping-relation. The results of the CVA are captured in a product-line decision model. Given the system requirements, the developer can identify the component web services that are relevant to the domain and the system.

For the application engineering phase, the services retrieved from the service broker are composed according to the common functional requirements by means of a modified version of the goal model and a functional service composition algorithm from [44]. The algorithm outputs all possible service compositions that satisfy the common functional requirements. By verifying all the compositions of this set against the common NFRs, we prune the composition set into a smaller set in which each composition satisfies all commonalities. This set serves as the composition search space for the variability, and we call it the commonality composition set (subset).

In order to improve the verification efficiency for the runtime customization, we do an indexing on each parameter of variation mapping to a subset of the search space. A composition subset associates with a parameter of variation if and only if all compositions in this subset satisfy the variability set by this parameter. The result of this preparation phase is the composition search space that is ready for runtime user customization.

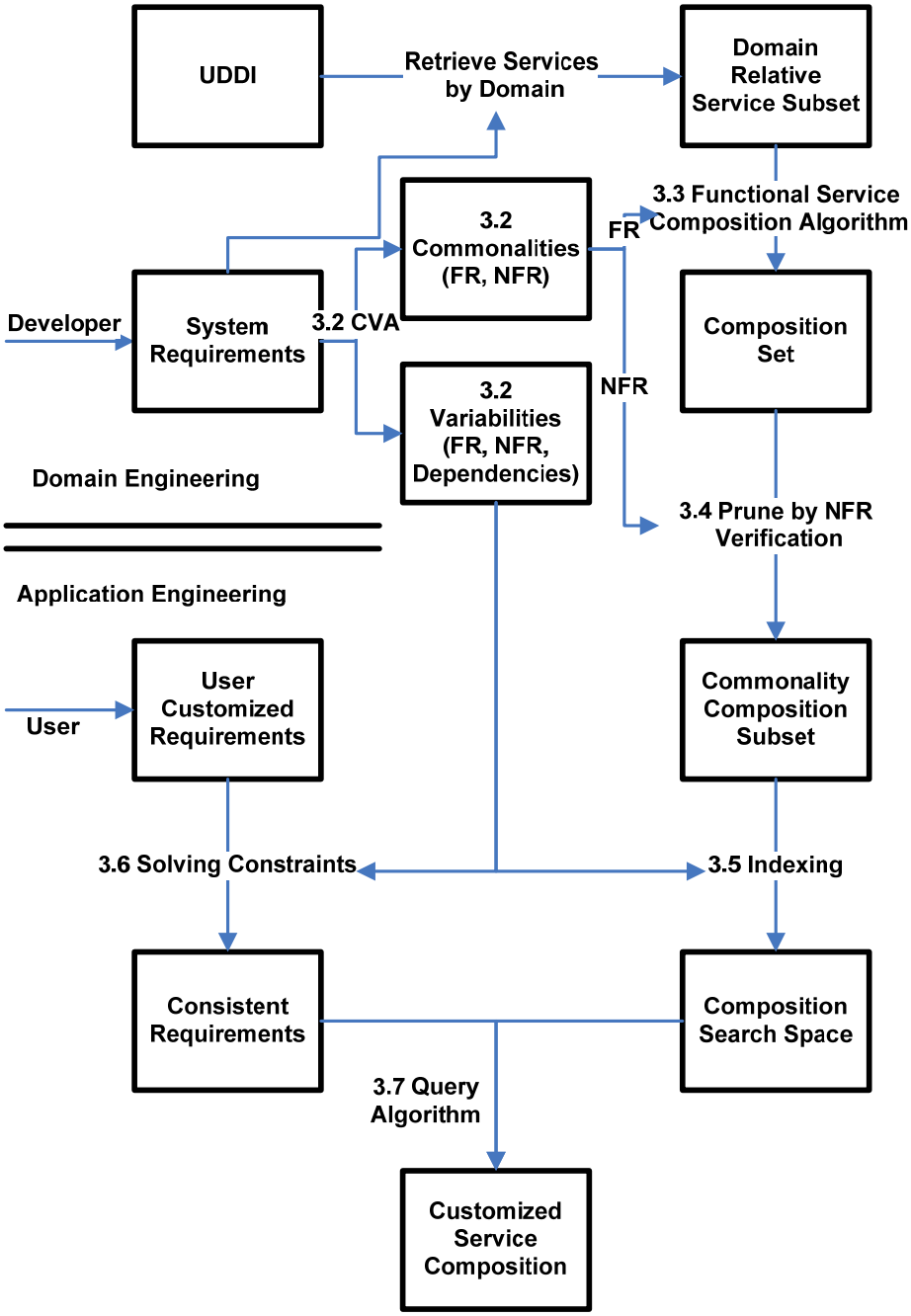

Figure 2. Overview of approach

After implementation of the web services, the user can access a default web service with basic functionalities. The user can then customize the functionalities and non-functional properties by setting all the parameters of variation in the decision model. By interacting with the decision model, the input requirements from the user are always kept consistent through solving the constraints in the variability dependency graph. The consistent specification of the variabilities is fed into a query algorithm to search valid compositions in the composition search space. The output of this algorithm is either the customized service composition satisfying all the requirements from the user or a report to the user of a failed composition attempt.

Chapter 5 describes this software product-line based approach of NFRs verification for multiple WSCs in detail on the Emergency Management System (EMS) as a case study.

The next section will summarize the contributions of our two new approaches.

1.3 Contributions

This research addresses the problem of NFR modeling and verification for web service composition. Existing NFR verification techniques of web service compositions are restricted in their efficiency and accuracy. For example, the context based NFR model [34][68] is verified by matching the keywords in the NFRs with those in the web service profiles and specifications. The drawbacks for context-based matching are limited efficiency of searches and inaccuracies in the key word identification. The axiom-based NFR models [47][59][69] require expert knowledge to build and verify the formal model of the NFR properties. There is a heavy initial overhead to build the NFR model.

Moreover, these solutions are not readily compatible with the existing functional web service composition approaches nor compatible with industry web service specifications such as WS-Policies and WS-Security.

Thus, the industrial and academic world lacks a NFR modeling and verification solution that is accurate, efficient and compatible with existing functional web service composition techniques and industry web service specifications. Since industry uses web service compositions extensively, reuse-oriented optimizations in their verification process are advantageous.

Our solution uses an automaton-based NFR model to represent quantitative QoS hard constraints with scoping information incorporated. An algorithm to verify that a proposed service composition satisfies the modeled NFRs is developed. Then, we propose three different strategies to verify multiple NFRs for a web service composition. The consistency among multiple NFRs thus can be checked prior to their verification. Efficiency and other advantages of these strategies are analyzed and compared with each other. We then focus on the process-oriented efficiency improvement in verifying multiple NFRs on multiple WSCs. We extend software product-line engineering techniques into web service composition and verification process to enable reuse of the verification results.

The contributions of this work are:

- Introducing an automaton-based NFR model which is able to represent quantitative QoS hard constraints for web service compositions.

- Incorporating scoping information into the NFR model for web service composition.
- Enabling consistency checking of multiple NFR models prior to the verification of the NFRs.
- Introducing an algorithm to verify that an NFR represented in an automaton model is satisfied in a web service composition, represented as an automaton model
- Presenting three strategies for verifying multiple NFRs in a web service composition automaton and comparing their advantages and disadvantages under different circumstances.
- Extending software product-line techniques to web service composition verification, to support reuse of intermediate verification results for verifying multiple NFRs on customizable web service compositions.
- Calculating the complexity of the product-line-based requirements verification process for web service compositions to analyze the efficiency advantages of this approach.
- Implementing a web service composition and verification simulation platform to simulate and evaluate the advantages of the two proposed approaches and the alternative verification strategies. The simulation results are analyzed to support the contribution of the dissertation.

The introduction of the approaches, the details of the case study results, the experimental results and the contributions will be described in the reminder of this dissertation.

1.4 Outline

Chapter 2 introduces the background research and reviews the related work in service oriented architecture, web service composition, functionalities and non-functional requirements of the web service compositions, verification of web service compositions and software product line engineering techniques. We also identify the differences of our work from existing approaches.

Chapter 3 describes a small system, the Emergency Management System (EMS), that we constructed based on [8], to illustrate the basic concepts and conduct case studies of our approaches.

Chapter 4 details our NFRs modeling and verification methodology for single or multiple NFR(s) verification. This approach introduces a semi-formal automaton-based NFRs model, which is capable of representing quantitative QoS hard constraints. Scoping information can be described along with the constraints for each NFR, which allows a precise definition of NFRs for a web service composition and a more efficient verification process. Three different strategies of verifying multiple NFRs are presented and compared for their advantages and disadvantages under different circumstances. Consistency checking is enabled by our NFRs model and considered in our verification to avoid wasting verification resources on inconsistent NFRs.

Chapter 5 introduces a new way to customize and verify composite web services by incorporating a software product-line engineering approach into web-service composition. The approach uses a partitioning similar to that between domain engineering and application engineering in the product-line context. It specifies the options that the user can select and constructs the resulting web-service compositions. By first creating a web-service composition search space that satisfies the common requirements and then querying the search space as the user selects values for the parameters of variation, we provide a more efficient way to customize web services.

Chapter 6 presents the results of the simulation experiments for the above two approaches introduced in Chapter 4 and Chapter 5. The simulation data and their analyses support the expectations and the conclusions of the proposed approaches.

Chapter 7 concludes the dissertation with a summary of our research contributions and directions for future work.

CHAPTER 2. RELATED RESEARCH

Prior work relevant to our research can be roughly divided into three areas: service oriented architectures, non-functional requirements, and product-line engineering. Their backgrounds and sub-domains are briefly described in this section.

Definition of Terms.

Non-functional requirement (NFR): A requirement defines the constraints on how the software-to-be should satisfy its functional requirements or how it should be developed [62]. In our research, NFRs can be interpreted as a NFR constraint.

Quality of service (QoS): QoS [24] refers to a set of quality requirements on the operation of a system, which can be considered as a subset of NFRs. In our research, we focus on the QoS requirements in the NFRs, so the NFRs constraints can be interpreted as QoS constraints.

NFR/QoS Constraints: A constraint is a restriction on the degree of freedom in developing a software system [1]. NFRs constraints consist of the hard constraints that must be satisfied by the system and the soft constraints that can only be satisficed [13]. In our research, we focus on hard constraints, so the term NFR/QoS constraints can be interpreted as NFR/QoS hard constraints.

NFR/QoS Attribute: An attribute refers to one type of the quality requirements, such as security or reliability. A value can be assigned to an attribute.

NFR/QoS Property: In our research, a property means an attribute with a value assigned, such as "Encryption Strength =128 Bit".

Composition Structure Pattern: A composition structure pattern describes what structure the multiple modular services use to compose with each other [35], such as sequential composition or parallel composition.

Aggregation Rule: The aggregation rule [24][69] defines the way of calculating a QoS property of a composite web service in each composition structure pattern, provided with the QoS properties of all modular services in the composition.

2.1 Service Oriented Architecture

Service-oriented architecture (SOA) is a group of software system design principles, which gains its popularity as the internet industry grows. SOA [41] is used in two software engineering phases: the system development phase and the system integration phase. SOA is a loosely-coupled software design pattern. SOA based system divides the functionalities into independent web services, which can be developed by different vendors and be distributed over the internet.

The most widely used solutions for SOA based systems are REST (REpresentational State Transfer) architectural style and SOAP(Simple Object Access Protocol) style [37]. REST web service is used by Yahoo Twitter, and Flickr. For SOAP based web service, Google is a primary supporter. Some IT companies, such as Amazon

and Ebay have implemented their web services in both REST and SOAP. As a comparison, REST based web service is easier to develop for the developers while SOAP based web service is easier to consume by the users. Both REST and SOAP based web services can be described by WSDL 2.0 (Web Services Description Language), which allows the consumers of the web services to have a unified implementation entry. In our research, we use SOAP based web service and WSDL 1.1 files to apply our approach.

Figure 3 shows the elements of SOA [29] in a tree structure. A web service consists of a service contract, the implementation of the service and the interface. The service contract defines the functionalities and non-functional properties that the service must provide to the service consumers. The contract defines the functionalities by service operations including methods, actions and parameters, and describes the invocation URL and interfaces. The contract defines the non-functional properties of the service as security constraints, quality of service, process, semantics and performance (in service level agreement). The implementation of service can be in either REST or SOAP architecture and consists of logics and data. The interface of the web service defines how and what the service consumers can access the service, which is usually described in WSDL file. Service providers publish and register their web services in service repository, which can be queried by the potential consumer. The Web service frontend is a wrapper, which allows the end users to access the functionality of the service as a standalone system. For example, a weather forecast web service's front-end is like a webpage, where the end user can access it on the browser by submitting a location and a date. Another

webpage will be shown to provide the weather forecast information the end user requested.

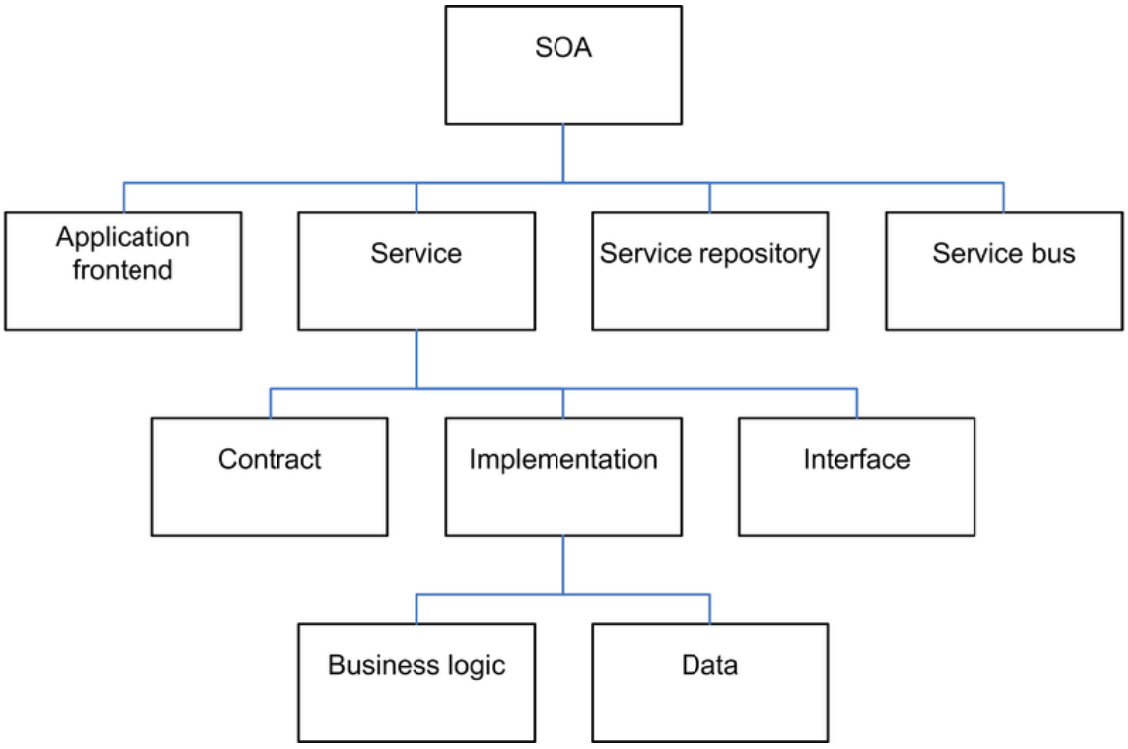

Figure 3. Elements of SOA (Cited from [29])

2.2 Web Service Composition

A composite web service with complex functionality can be realized by composing or assembling the modular web services developed by disparate vendors. For example, a travel alert web service can retrieve the weather information from a modular service published by a weather broadcasting company and retrieve the traffic information from a modular service published by the traffic control agency.

Two types of web service composition mechanism are widely used: the orchestration-based composition and the choreography-based composition [14]. The orchestration-based composition assumes that there is a central transaction controller, the conductor, to describe the process flow among the component services and the service user. On the other hand, the choreography-based composition describes the message interaction of the component services without a central component to hold the entire conversation. In BPEL4WS, this is realized by automated execution of the service workflow. In our work we concern choreography-based composition.

From the composition time of the modular services, the web service composition can be classified into static composition and dynamic composition. Static composition approaches take the functional requirements and non-functional requirements as input to generate a composite web service from the modular service repository to satisfy the requirements. Then the composed web service is deployed on the server and starts taking requests from the users. The static composition approaches don't consider the contents of the messages sent among the modular services at the service composition phase. On the other hand, the dynamic composition approaches may need to recompose itself upon different message received. In our research, we focus on static composition rather than dynamic composition.

Niola Milanovic and Miroslaw Malek [35] identified six basic composition structure patterns in their work, represented in operators:

1. Sequence ∇ executes two or more services in a sequential order.

2. Choice ◊ executes two or more services in parallel, and then non-deterministically chooses output of one and only one of them.

3. Selection ⊗ selects one of the candidate services and executes it.

4. Parallel with communication $\|_p$ executes two or more services concurrently, and then performs a logical operation on their outputs, and based on the operations results, chooses one of the outputs.

5. Parallel without communication ‖ executes two or more services concurrently, without any communication and synchronization between them.

6. Loop $\propto_p$ executes a service iteratively, until an exit condition is met.

However, at the static composition stage, we cannot foresee the values in the messages, so "Choice", "Parallel with communication" and "Parallel without communication" can only be taken as one composition pattern: parallel. Thus, in our research of NFRs verification of static web service composition, we only consider two composition structure patterns: sequential and parallel.

2.3 Non-Functional Requirements Monitoring in Web Services

Requirements monitoring aims to assure satisfactory compliance of the requirements by inserting codes into running systems [17]. Manual updates are necessary when requirements violations occur in the system [17]. Later, a goal model based reconciliation method [16] is introduced to monitor requirements compliance, in order to handle the runtime system derivation from the original requirements. A system adapts to

a change of requirements by re-selecting the components, based on the options in the goal model.

Robinson [49][50] extended the requirements compliance monitoring techniques to web services, allowing the web system to record statistics of compliance or observance of QoS requirements and other non-functional attributes for analysis purposes. He identifies scenarios and QoS attributes that can be monitored in a web service system, including Service Failure, Tardy Service (Delay), etc. In [50], Robinson uses web service monitoring techniques to ensure regulation of business behaviors, such as that an order for an item placed on Saturday must be shipped out by Monday.

Industry solutions [51] for monitoring web service requirements take place in three application environments: development, quality assurance and production. The quality of service attributes that can be monitored include availability, performance (throughput and latency), stress test, reliability, scalability, integrity, and corrective measures.

Research on monitoring requirements sets up the fundamentals for our research on adaptive maintenance of web service compositions, which allows us to obtain QoS information for each modular service in a composite web service for analysis and improvement purpose. Our approach is to assume the existing requirements monitoring mechanism and then build on it.

2.4 Verification of Web Service Compositions

2.4.1 FRs and NFRs Models for Web Service Compositions

Most existing web service composition and verification approaches [14] focus on satisfying the functional requirements. Some previous work has approached the problem of verifying NFRs in web service compositions from several different perspectives. Quantitative preference techniques [54] use utility theory to try to optimize the selection of web services. However, such techniques are restricted to NFRs that can be adequately represented in quantitative terms. Qualitative preference techniques [52][69] incorporate preference attributes but do not yet handle global nonfunctional constraints (that must hold over the entire composed service).

Constraint satisfaction techniques [53] optimize based on preferences, but do not currently implement heuristics aimed at satisfying hard constraints. In addition, constraint satisfaction techniques require a specialized, distinct preference language. That is, the user must specify the FRs and NFRs using different representations.

Other researchers [24][34][68][69] have also described NFRs in the context of Quality of Service (QoS).

In general, verification of web service compositions, the functional/non-functional properties are commonly modeled in formatted context-based description [34][68], axioms (for theorem proving and mathematical methodologies) [47][69], LTL/CTL (temporal logic languages for model checking) [46] and state machines [18][24][45].

Context-based property models can be verified by context-based matching, including syntactic matching and semantic matching, in order to ensure compliance of the composition to the functional and non-functional requirements [34][68]. The drawbacks for context-based matching are limited efficiency of searches and inaccuracies of key word identification.

Axiom-based property models.

Rao, Kungas and Matskin introduce a method for semantic web service composition based on Linear Logic theorem proving [47]. Zeng et al. [69] introduce optimizing preferences over NFRs based on utility functions in linear programming. However, the Linear Logic prover requires expert knowledge to construct property models.

Temporal logic-based property models are commonly used in planning and model checking based web service composition [59]. The obvious obstacles are the state explosion problem during verification and assurance of consistency among the property models.

Automata-based property models.

Modeling NFRs as automata facilitates collaboration with other functional verification approaches. Foster, Uchitel, Magee and Kramer, e.g., describe a model-based approach (LTSA-WS) to verify composition implementations [18]. The automaton-based property model is extracted from the NFRs context. The verification mechanism is a trace equivalence check, which composes labeled transition systems (LTS) based on pre-

constructed finite state process models. This research also shows that web service compositions modeled in the widely-used standard, BPEL4WS, can be translated into LTS in a finite domain.

Pathak, Basu and Honavar [45] introduced a combined approach for both the functional and non-functional requirements during service composition. The functional requirements are modeled in a LTS-based goal model. During composing component services towards the goal model, a quantified NFRs threshold is also considered. However, its ability to handle NFRs is limited in that it only handles one NFR, these NFRs must be quantified, and the NFRs must have a global scope.

Oster [43] shows how a single buffer can help find a substitute for a service component in an asynchronous service composition under functionality constraint. This analysis shows that two service components can have the same functionalities, and substitute for each other even when they have different message sequences, under an asynchronous composition environment. This work can be used as the basis for us to find functionality substitutes for a modular service. This work is different from our work in the following way:

1. We consider substitutes of QoS attributes of a modular service on a composition failure under QoS constraints, rather than in terms of functionality.
2. Our approach guides the process of choosing an alternative modular service to satisfy the QoS constraint, rather than only doing the equivalence check of two modular services.

Nadkarni [38] introduces an iterative process to recover from functional failures of web services compositions at composition time. It suggests that the user change the goal service by providing feedback from the failure analysis. Rather than considering functional failures at the time of composition, our research considers non-functional failures caused by requirements evolution, after service deployment. This work can be integrated into our research in order to have a unified strategy to handle functional and non-functional failures of web service compositions.

2.4.2 Verification of Security and Reliability Constraint

Security in SOA consists of confidentiality, integrity, and availability attributes, which approves the authorized actions and denies unauthorized actions [3]. Raya et. al. identified vulnerabilities for web service security including jamming, forgery, in-transit tampering, impersonation, privacy violation and on-broad tampering [48]. In existing solutions, the security requirements and solutions for the messages of each service are modeled in WS-Security, which is described within the WS-Policy embedded in the WSDL file [63]. However, the specification of the security requirements exists only on the operation level, which means that the global security requirements have to be divided into the security requirements for each message exchange operation. A strategy to assure the global security requirements of the web service composition is security pattern match [66]. Security requirements are modeled in patterns, and the service composition is considered as a whole system being checked satisfactory of these patterns. Moreover, the security level check in the software architecture domain refers to the situation that a more

secured component depends on a less secured component in a composed system, which is called security level jeopardization [57].

Another important aspect of the non-functional requirements of web service composition is service reliability. Active research work on service reliability includes the reliability analysis and prediction at the service composition stage. Foster [20] introduced a service behavior model to validate the composite web services to be free of deadlocks. To be able to predict and measure composite web service reliability, faults and failures are modeled and analyzed in their execution scenarios [2][67]. A transformation from BPEL to a petri-net model is also introduced to analyze the reliability of composed web services [70]. In [32], redundancy as a way to improve composite web service reliability is also discussed.

2.5 Software Product Line Engineering in Web Service Composition

Besides improvement of non-functional requirements verification efficiency in the representation and algorithm view, the on-demand verification overhead of composed web service can also be reduced or avoided from the process view. The introduction of product line engineering into the web service composition verification can avoid on-demand verification overhead by doing pre-verification in the domain analysis stage and it can maximize reuse of verification results across different NFRs verifications and different service compositions. That is why software product line engineering (SPLE) can help to solve our problem.

The SPLE process consists of two phases: a domain engineering phase and an application engineering phase. In order to simplify the product generation process and

hide the background complexity, a decision model is created during the domain engineering phase and reused during the application engineering phase. We here use the fully constructed decision model introduced in [65] with all its background relation models to generate verifiable compositions of customizable web service families.

Several researchers have applied product line engineering (PLE) to the web service domain in different ways. Karam, Dascalu, Safa, Santina and Koteich, [27] incorporated PLE into web service-based applications (WSbWAs). The web applications benefited from the reconfigurable, reusable pages, workflows and web services (WebPads as composite web services). These supported the common artifacts of the web development domain and the particular aspects of the application in the domain. Their work focused on the reuse of functionalities during product evolution rather than on the NFRs of the web applications.

Balzerani, Di Ruscio and Pierantonio [5] followed the FODA (Feature-Oriented Domain Analysis) [30] approach to SPLE to construct a reusability-oriented web application architecture. It took the bounded input parameters of the functional methods as variation points. New requests from the user were considered as different variation values for the product line to enable both design time reconfiguration and runtime reconfiguration. However, this work lacked a clear explanation of the domain engineering phase to distinguish the commonalities of the web applications from the variabilities.

Capilla and Topaloglu [9] introduced a way of applying SPLE into web service composition. The authors identified types of variation points that can be used in a web

service based product line: the order of the composition in an orchestration composition, the flow conditions in a message path, the service alternatives, the exception handling possibilities and the quality of service choices. These types of variations served to customize web services during the design and implementation phases.

Liu et. al. [31] introduced a state-based model to ensure the safety properties when a software product line evolves. This state-based model captures the feature interactions of the software product line by utilizing the fault tree analysis techniques.

CHAPTER 3. CASE STUDY: THE EMERGENCY MANAGEMENT SYSTEM

To illustrate our approaches in case studies, we describe a web service dispatching system, the Emergency Management System (EMS), based on [8]. The key functional requirements of the EMS are to *dispatch ambulance(s), fire truck(s) and police to a location upon request using available resources*. The chapter begins with a section introducing real world EMSs and mishaps of these systems due to insufficient NFRs verifications. Then the next section describes the EMS used in our research in detail with the architecture design, functional requirements and non-functional requirements.

3.1 EMS Examples

EMS is a high-assurance system and a safety critical system [56], because failure to meet its functional (e.g., dispatch ambulance to an accident victim) or non-functional (e.g., keep patient information confidential) requirements can result in serious harm to people. The reliability of the system, the availability of the services, the accuracy of resource allocation and the security of the communications are crucial to public safety. Various EMS (also called Incident Command System) implementations in existence include the Incident Command System of the U.S. Federal Emergency Management

Agency (FEMA) [7] and the National Incident Management System (NIMS) for major, multi-site incidents [42].

- Many incidents in emergency management systems have historically been caused by incorrect request/dispatch information or insufficient non-functional constraints verification. Two well-known examples are:

1. London Ambulance System (LAS) Failure in 1992 [33]: Due to the poor design and implementation of new LAS, the performance and the dispatch delay wasn't guaranteed. A massive delay in ambulance assigning wasn't discovered in time, which leads to a maximum ambulance delay to 12 hours and patients' death.
2. Australian Emergency System Failure in 2006 [22][40]: A request to dispatch ambulance sent by the police department wasn't delivered because of the firewall, which causes the ambulance dispatch delayed for a day and leads to the patient's death. This incident was partially a result of insufficient delivery assurance verification of the service.

The next section introduces the EMS built based on a real world incident management system and used in our research as a case study.

3.2 Emergency Management System Case Study

We constructed a small version of the Emergency Management System (EMS), based on [8], to illustrate the basic concepts of our approach. EMS consists of several different units: the Field Officer Service, the Request Dispatch Service (Dispatcher for short) and services for emergency handling, including an Ambulance Dispatch Service, a

Fire Station Dispatch Service and a Police Dispatch Service. The functional requirements are to dispatch ambulance(s), fire truck(s) and police to a location upon request. These requests are specified and sent by the Field Officer through a service in a mobile terminal or a PDA.

The Dispatcher has three types: the normal dispatch service, which is responsible for routine situations; the speed-line dispatch service, which has low communication delay, compared to the normal dispatcher and is used for urgent dispatches; and the highly-secured dispatcher, which is used for national security related cases. The dispatcher service can also invoke a GPS-MAP service and other third party services. Figure 4 shows a sample structure for EMS service composition. The services connected by the dashed lines represent the optional services in a valid composition. In other words, we can call a automaton that represents all possible service composition functionalities as a service composition goal. An implementation of this composition goal may only have a subset of the elements in the composition goal.

EMS is a critical software system because failure to meet its functional (e.g., dispatch ambulance to an accident victim) or non-functional requirements (e.g., keep patient information confidential) can have catastrophic consequences. The reliability of the system, the availability of the services, the effectiveness of resource allocation and the security of communications are crucial to public safety. For example, messages in an EMS must be secured against malicious eavesdropping, interception and falsification before deployment. Hence, security NFRs for the EMS might require that the messages in different scenarios be sent at different encryption levels depending on the type of

emergency incident. For national security related incidents, messages and service operations may need to be protected with stronger encryption than in the case of civilian incidents. An example of a security NFRs for EMS is that *a request to dispatch police shall be processed using highly-encrypted message paths*.

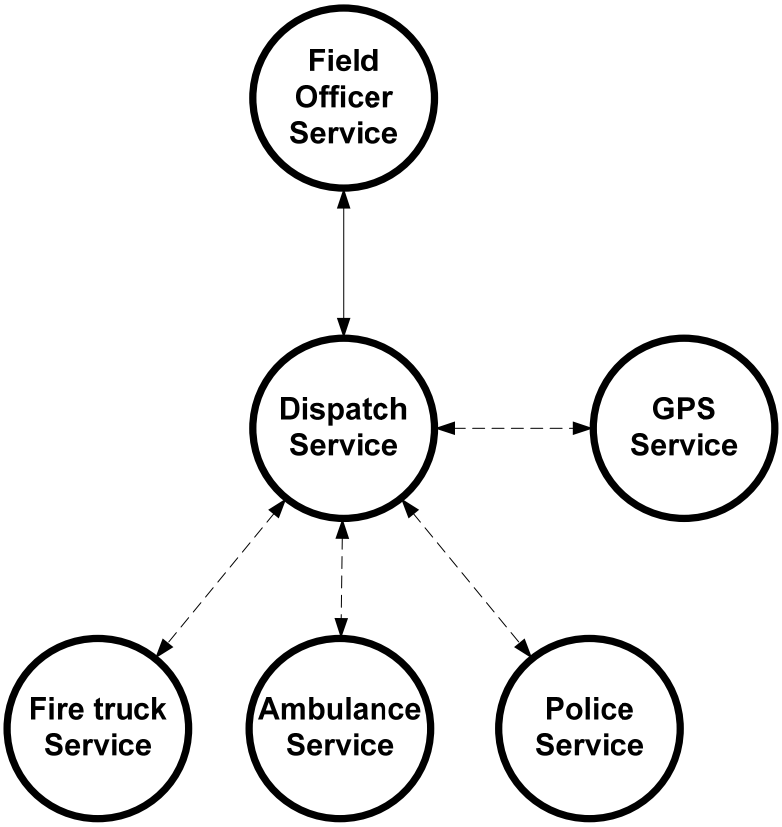

Figure 4. Sample composition for EMS

Security guarantees for encryption and authentication of the messages for *each* service are typically specified using the web service security policy (WS-Security and WS-Policy) which is included in the WSDL specification for the service [63]. The security NFRs for the EMS, however, applies to the *composite* service which consists of the multiple component services that make up the EMS. This presents us with the

challenge of verifying that the security guarantees available for the individual component services that make up the composite service indeed satisfy the security NFRs of the composite service. For a composite web service, the security requirements can apply either globally (to all services in the composite web service) or locally (to more than one service operation but not to all). Service operations are the external functionalities of a component service defined in their public interfaces. A service operation takes an input, takes actions on the input and returns the output. A component service can have one or more service operations.

CHAPTER 4. AUTOMATA-BASED VERIFICATION OF SECURITY REQUIREMENTS OF COMPOSITE WEB SERVICE[1]

With the increasing reliance of complex real-world applications on composite web services assembled from independently developed component services, there is a growing need for effective approaches to verifying that a composite service not only offers the required functionality but also satisfies the desired *non-functional requirements (NFRs)*. In high-assurance applications such as traffic control, medical decision support, and coordinated response to civil emergencies, of special concern are NFRs having to do with security, safety and reliability of composite services. Current approaches to verifying NFRs of composite services (as opposed to individual services) remain largely ad-hoc and informal in nature. In this chapter we develop techniques for ensuring that a composite service meets the user-specified NFRs expressible in the form of hard constraints e.g., "response time has to be less than 5 minutes." We introduce an automata-based framework for verifying that a composite service satisfies the desired NFRs based on the known guarantees regarding the non-functional properties of the component

[1] The work presented in this chapter is adapted from Sun, H., Basu, S., Lutz, R. and Honavar, V. (2010). Automata-Based Verification of Security Requirements of Composite Web Services. 21th IEEE International Symposium on Software Reliability Engineering (ISSRE'10), San Jose, CA, USA.

services. We further show how to improve the efficiency of verifying that a composite service indeed satisfies a desired set of NFRs by: (i) Exploiting information about the applicability of specific NFRs (e.g., security) only to certain subsets of the component services that make up a composite service to minimize the verification effort and (ii) Identifying inconsistencies between NFRs with overlapping scopes. We illustrate how our approach can be used to verify the security requirements for an Emergency Management System. We also show how the approach can be used to verify whether a composite service satisfies any desired set of NFRs that can be expressed in the form of hard constraints of a quantitative nature.

4.1 Introduction

As web technologies become increasingly widespread, there is a proliferation of independently developed web services in many application domains. Complex real-world applications typically rely on composite web services assembled from multiple independently developed composite services. Consequently, a variety of approaches have been developed for assembling composite services that satisfy the user-supplied functional specifications [14][35]. Functional requirements (FRs) describe how the composite service ought to process its input so as to generate the desired output. A number of techniques are available for verifying whether a composite service satisfies the user-specified FRs [14][35] [45].

However in many applications, a composite service needs to satisfy not only the desired FRs but also the user-specified *non-functional* requirements (NFRs) [36]. NFRs typically specify constraints that must be met with respect to the security, safety and

reliability of composite services. NFRs can be hard constraints or soft constraints. Hard constraints refer to constraints that *must* be satisfied e.g., "response time has to be less than 5 minutes". Soft constraints on the other hand specify user preferences over non-functional attributes e.g., "the lower the cost, the better". NFRs that specify quality requirements e.g., with respect to response time, are often also called Quality of Service (QoS) requirements [61].

Ensuring that a composite service satisfies not only the desired FRs but also NFRs is especially critical in the case of high assurance applications such as traffic control, medical decision support, and coordinated response to civil emergencies [3]. Consider for example, an Emergency Management System (EMS) [8]. The key FRs of the EMS (See Figure 5) is to *dispatch ambulance(s), fire truck(s) and police to a location upon request using available resources*. An EMS consists of several components: the *Scheduler* Service that takes the request messages from the field officers' mobile terminals; the Emergency *Resource* Services (e.g., Police.A and Police.B) that interact with the resource databases and manage the local resources of ambulances, fire trucks and police; and the *Dispatcher* Service that interacts with ambulances, fire trucks and police cars and dispatches them to the target location.

EMS is a high-assurance system because failure to meet its functional (e.g., dispatch ambulance to an accident victim) or non-functional requirements (e.g., keep patient information confidential) can have disastrous consequences. The reliability of the system, the availability of the services, the effectiveness of resource allocation and the security of communications are crucial to public safety. For example, messages in an

EMS must be secured against malicious eavesdropping, interception and falsification before deployment. Hence, security NFRs for the EMS might require that the messages in different scenarios be sent at different encryption levels depending on the type of emergency incident. For national security related incidents, messages and service operations my need to be protected with stronger encryption than in the case of civilian incidents. An example of a security NFRs for EMS is that *a request to dispatch police shall be processed using highly-encrypted message paths*.

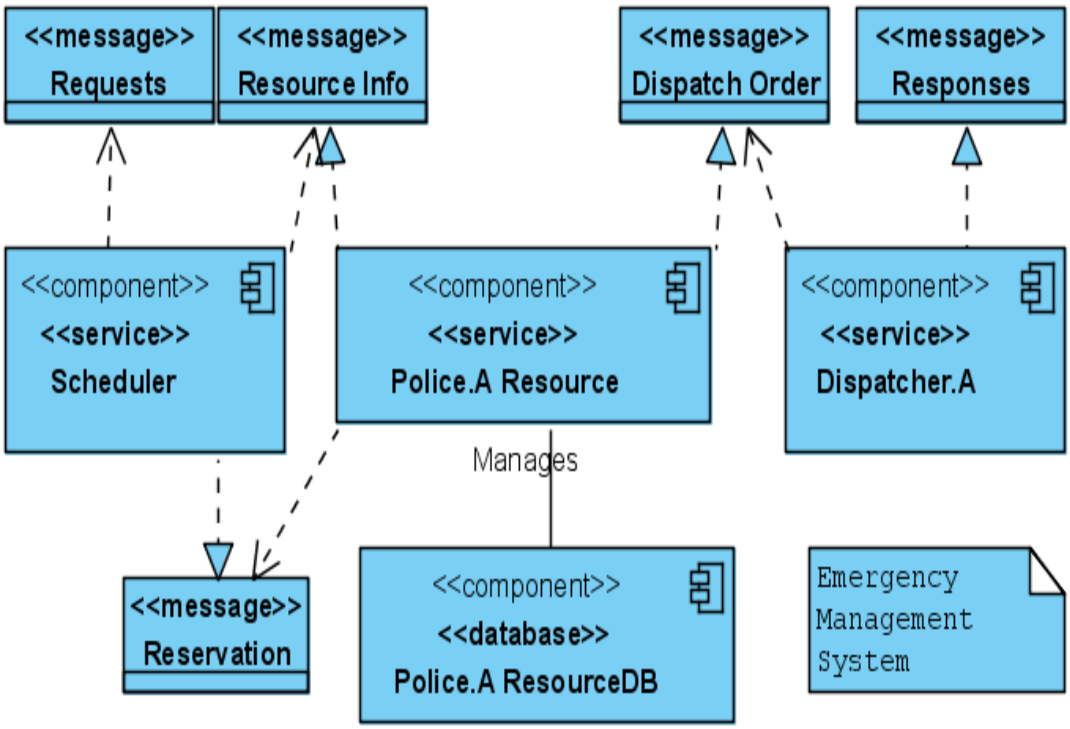

Figure 5. An illustrative example: excerpt of a web-based Emergency Management System

Security guarantees for encryption and authentication of the messages for *each* service are typically specified using the web service security policy (WS-Security and WS-Policy) which is included in the WSDL specification for the service [63]. The security NFRs for the EMS, however, applies to the *composite* service which consists of the multiple component services that make up the EMS. This presents us with the challenge of verifying that the security guarantees available for the individual component services that make up the composite service indeed satisfy the security NFRs of the composite service. For a composite web service, the security requirements can apply either globally (to all services in the composite web service) or locally (to more than one service operation but not to all). Service operations are the external functionalities of a component service defined in their public interfaces. A service operation takes an input, takes actions on the input and returns the output. A component service can have one or more service operations.

Other similar emergency management systems in existence include the Incident Command System of the U.S. Federal Emergency Management Agency (FEMA) and the National Incident Management System (NIMS) for major, multi-site incidents [42]. Many incidents in emergency management systems have historically been caused by incorrect request or dispatch data (e.g., the London Ambulance System in 1992 and the Australian Emergency System in 2006) [40].

Current approaches to verifying non-functional requirements (NFRs) of composite services (as opposed to individual services) remain largely ad-hoc and informal in nature. In this chapter we develop techniques for ensuring that a composite

service meets the user-specified non-functional requirements expressible in the form of hard constraints. We introduce an automata-based framework for (1) representing NFRs that can be expressed in the form of hard constraints and (2) verifying that the composite service satisfies the NFRs based on the known guarantees regarding the non-functional properties of the component services. We illustrate how our approach can be used to verify the security requirements for an emergency management system. However, the proposed approach can be used to verify whether a composite service satisfies any desired set of non-functional requirements that can be expressed in the form of hard constraints of a quantitative nature.

The proposed approach to static verification of NFRs of composite services incorporates two novel elements:

Scoping of NFRs:

In many situations, specific NFRs (e.g., with respect to security) may be applicable only to specific subsets of the component services that make up the composite service. Moreover, the subset of services that need to satisfy NFRs may differ across the different NFRs (e.g., response time, security). Existing quality of service models for composite web service are not expressive enough to represent NFRs that apply to specific subsets of the services that make up a composite service [24][68][69]. To allow modeling of NFRs that apply to different subsets of components of a composite service, we introduce the notion of scope. The increased precision in the description of NFRs enables more efficient verification of NFRs.

Consistency checking of NFRs:

NFRs with overlapping scopes can be inconsistent (and hence unsatisfiable) in the case of some candidate composite services. This scenario is not uncommon in settings where attempts to achieve some NFRs (say, with respect to security) may rule out the achievement of other NFRs (say, with respect to response time) [13]. Hence, when the scopes of different NFRs overlap, checking the consistency of NFRs can help avoid effort that would otherwise be wasted in exploring composite services that would violate the NFRs constraints. Our automata-based model permits the consistency checking of NFRs with overlapping scopes.

We introduce and compare three alternative strategies in the case that multiple NFRs exist and analyze their relative advantages and disadvantages under different scenarios. This approach to verifying the NFRs can also support efficient re-verification of composite services as needed when NFRs are updated.

While we focus on security requirements in this work, the approach described here can handle any other quantifiable quality of service requirement on web service composition that is checkable using information available at composition time. Thus, for example, performance and reliability constraints referring to available, historical data (e.g., average throughput) can be verified in our approach, while performance and reliability constraints that rely on execution time data (e.g., current throughput) must be

handled dynamically and are not suitable for this approach. This is discussed further in Sect. 4.4.5.

The rest of this chapter is organized as following. Section 4.2 provides an overview of our approach. Section 4.3 briefly reviews an approach to exploring the space of candidate compositions. Section 4.4 describes our approach to modeling NFRs and several strategies for verifying whether a candidate composition satisfies the NFRs. Section 4.5 shows the design of the implementation and Section 4.6 discusses the related researches. Section 4.7 concludes with a brief discussion and limitations and directions for further research.

4.2 Overview

Our overall approach to assembling a composite service that satisfies both the functional and non-functional requirements (in the case of security related NFRs) is shown in Figure 1. The functional composition algorithm [45] (briefly reviewed in Section 4.3 below) uses a. goal model in the form of an automaton to encode the user-specified FR. A composite service is represented as an automaton where states represent the component services participating in the composition and inter-state transitions represent composition of the corresponding component services. The functional composition algorithm assembles a composite service that verifiably satisfies the FRs by exploring the space of candidate compositions.

The focus of this chapter, however, is on verifying that a composite service assembled by the functional composition algorithm *also* satisfies the NFR. We use an automaton to represent the non-functional properties of a composite service (See Section

4.4 for details). The latter correspond to sequences of properties of non-functional attributes of the component services that make up a composite service. Labels on transitions of the NFRs automata encode the set of non-functional *constraints* that are satisfied by the services that are connected by the respective transitions.

A composite service is said to satisfy a given set of NFRs if and only if the sequence of properties over non-functional constraints represented by the NFR-automata is also realized by the automaton that encodes the composite service. A composite service conforming to a desired NFRs is obtained by computing and verifying the product of the automaton representing the composite web service and the automaton representing the corresponding NFR. This lifts the NFRs analysis from the level of individual services to the level of the search space of candidate compositions obtained from the functional requirements.

The non-functional properties, e.g., security policies, of the component services are extracted from their WSDL specifications. A domain terminology mapping table is used to translate the user-specified NFR, e.g., “messages shall be highly-encrypted” into the terminology used in the WSDL specification (“messages must use standard Basic128Sha256 encryption algorithm with a 256-bit key”).

The verification of a composite web service consists of two main parts, as shown in Figure 2. The first part is the functional composition of the component web services. This is prior work [45], briefly reviewed in Section 4.3 below. The goal model is an automaton constructed to model the functionalities of the required system. With the goal

model as input, an iterative algorithm generates a composite web service that verifiably satisfies the functional requirements.

The second part of the process, and the contribution of this chapter, is the verification of the security non-functional requirements, shown in the bottom half of the diagram. The security NFRs is modeled in an automaton-based model, as described in Section 4.4 below. The composite web service derived from the first step can then be verified against this security automaton. During the verification, the security policies of the service operations are pulled from the appropriate WSDL files. The domain terminology mapping table is used to translate the security policies into an algorithm-readable input and to provide the algorithm with the aggregation rules needed to handle security NFRs.

It must be noted that the focus of this research is on *static* verification of NFR. Verification of NFRs whose satisfaction is contingent on certain runtime constraints on the performance of the component services being met (e.g., whether an ambulance reaches its destination in time to meet the response time NFR) that can only be *dynamically* verified at runtime are beyond the scope of this chapter.

4.3 Functional Composition

In our approach, we take the pre-construction of the composite web service meeting the FRs as fundamental. This is illustrated in the top half of the overview in Figure 1. In our illustration, we use the algorithm described in previous work [45] to achieve this construction, but other approaches, such as those in [14][35], would also work.

The goal model in Figure 6 is constructed as a first step in composing the component services into a composite service. The goal model captures two types of information required by the FRs: the workflow of the composite service and the component level functionality for each service operation. The states represent component services and the transitions represent the input/output messages for the service operations. Note that one component service can have more than one operation, so may be represented by multiple states. For example, the ambulance service can have service operations to provide availability information regarding ambulance resources and to reserve these resources.

The label on each transition represents a guard condition or service operation functionality required to be realized by a component service, such as Scheduler_Reserve. Each operation can take message inputs and produce message outputs. The goal model identifies the service operations and workflows of EMS as a functionality template for required EMS candidates. The functionalities, such as scheduling, reserving and dispatching firetrucks, ambulances, and police are specified in the goal model.

The goal of the functional service composition algorithm is to find component services providing the required functionalities. The Iterative Forward and Backward Search Algorithm [45] is invoked with the goal model in Figure 6 as input to generate the composite web service. The algorithm maintains a composition result set during the computation and tentatively searches forwards and backwards in the component service registry for a service that satisfies the functional requirement. The result set moves forward if the current component service satisfies the service operation specified in the

goal model and stores it in the result set; otherwise, it moves backwards and tries another component service. The algorithm uses an exhaustive search to find all composite web services satisfying the goal model finally.

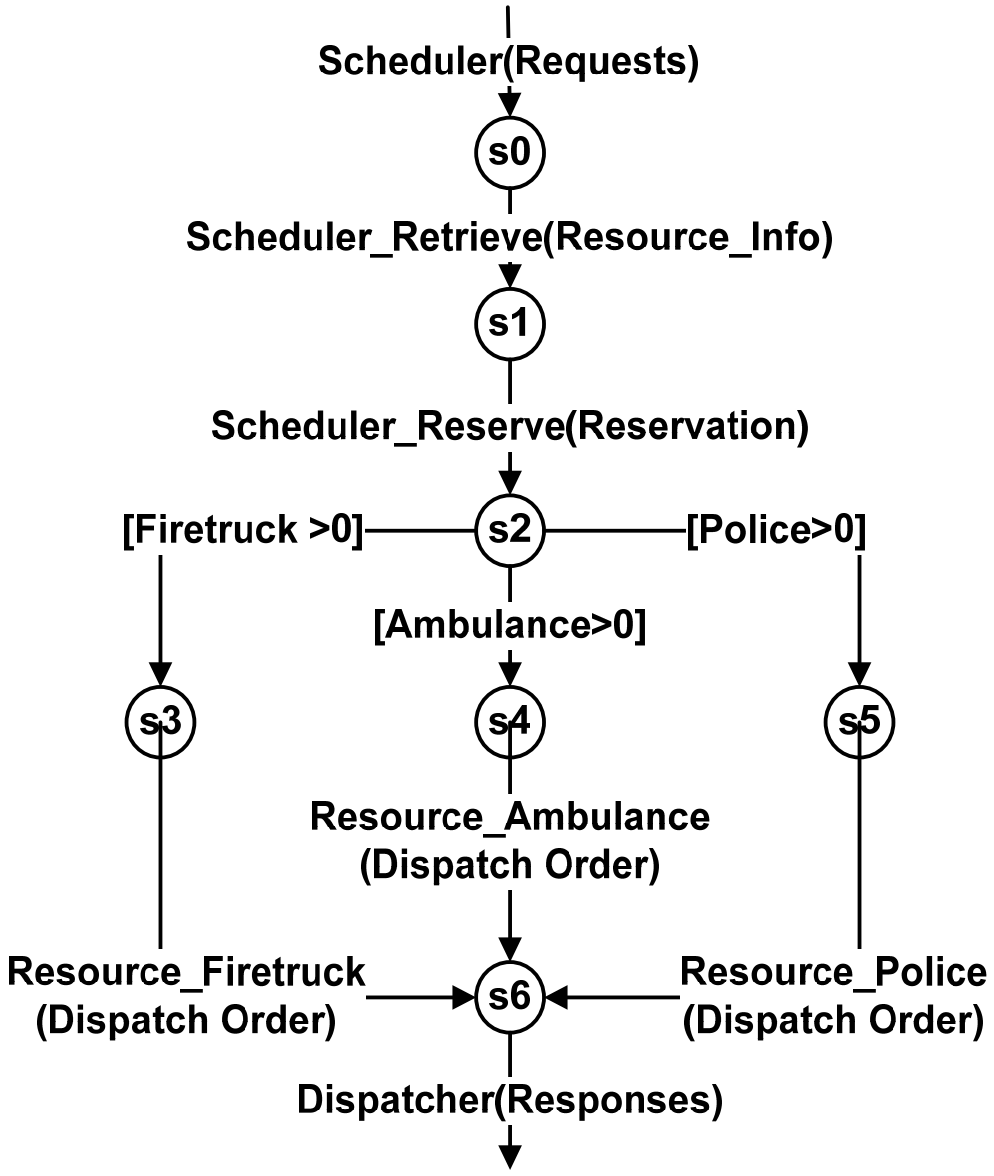

Figure 6. Functional goal automaton of the EMS

4.4 Security Requirement Verification

To be able to verify the NFRs, the first step is to model them. As shown in the overview in Figure 1, each security NFRs is modeled as a security property automaton.

4.4.1 NFRs Automata Derivation

We thus first define the finite state automaton used to model the composite web service and the NFRs.

Definition 1: A finite state automaton is a tuple FSA = (S, s_0, Δ, P, F) where S is the finite set of states, $s_0 \in S$ is the start state, and $\Delta \subseteq S \times 2^P \times S$ is the transition relation of the form $s - \phi \rightarrow s'$ such that s, $s' \in S$, and $\phi \in 2^P$ is a subset of propositions P. Finally, $F \subseteq S$ is the set of final states.

A finite sequence is said to be accepted by the FSA if and only if the sequence starts from s_0 and terminates in any of the final states in F. The NFRs and composite web services are described using FSA.

Figure 7 shows a composite web service automaton derived from the FRs goal model in Figure 6. This candidate represents an instance of the EMS where only police need to be dispatched. The composite web service is here augmented with its associated security NFRs on the transitions.

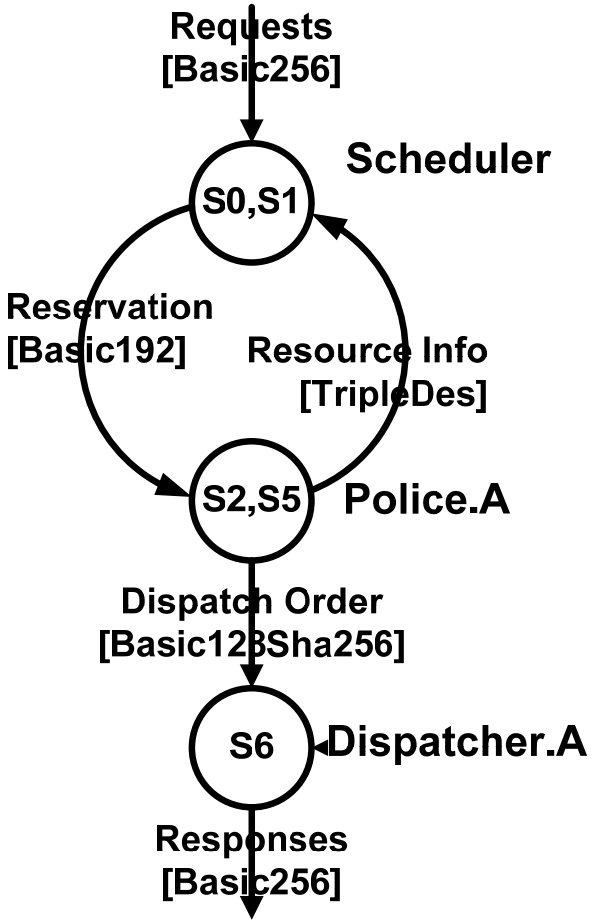

Figure 7. A composite web service candidate automaton of EMS with security policies associated

We will also model the NFRs property constraints using the automaton in Definition 1. Note that an automata-based property model can represent both safety and liveness properties [12][21], where a safety property is of the form "a program never enters an undesirable state" and a liveness property is of the form "a program eventually enters a desirable state". If a state violating a safety property is encountered during composition of the composite web service automaton and the NFRs automaton, or if a state satisfying a liveness property is not reached in any branch at the end of the

composition, we say that this composite web service violates the NFR. To achieve compositional verification of the multiple NFRs properties, we unify the types of properties by converting the liveness properties to safety properties via use of an additional trap state π to capture those undesirable final states in the liveness properties. This is described more fully in [12].

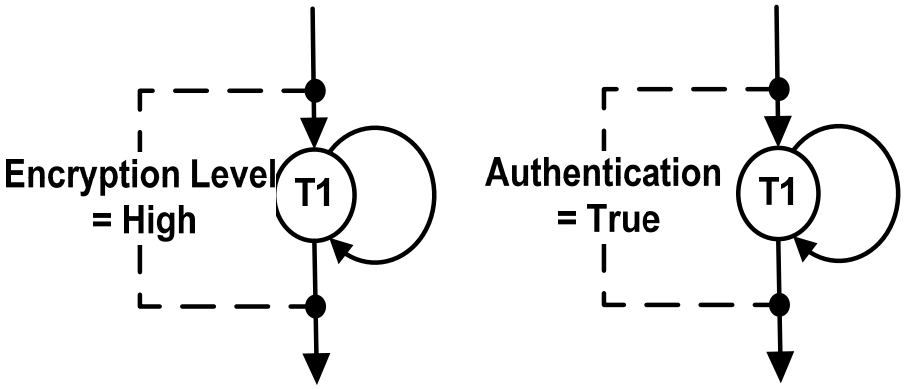

Figure 8. Global security requirements: (1) all service operations shall employ highly-secured messages; (2) all service operations shall have authentication.

Scoping the NFRs: The scope of each NFRs constraint is specified as the dotted line as shown in the excerpts in Figure 8. A global NFRs property can be described as a single self-loop with a NFR property constraint on it. For example, Figure 8 shows two global security properties. One requires that all service operations shall employ a high encryption level and the other requires that all messages and operations shall be authenticated. The concept of "Encryption Level = High" and "Authentication = True" in the constraints will be defined in the domain terminology mapping table in Section 4.4.2.

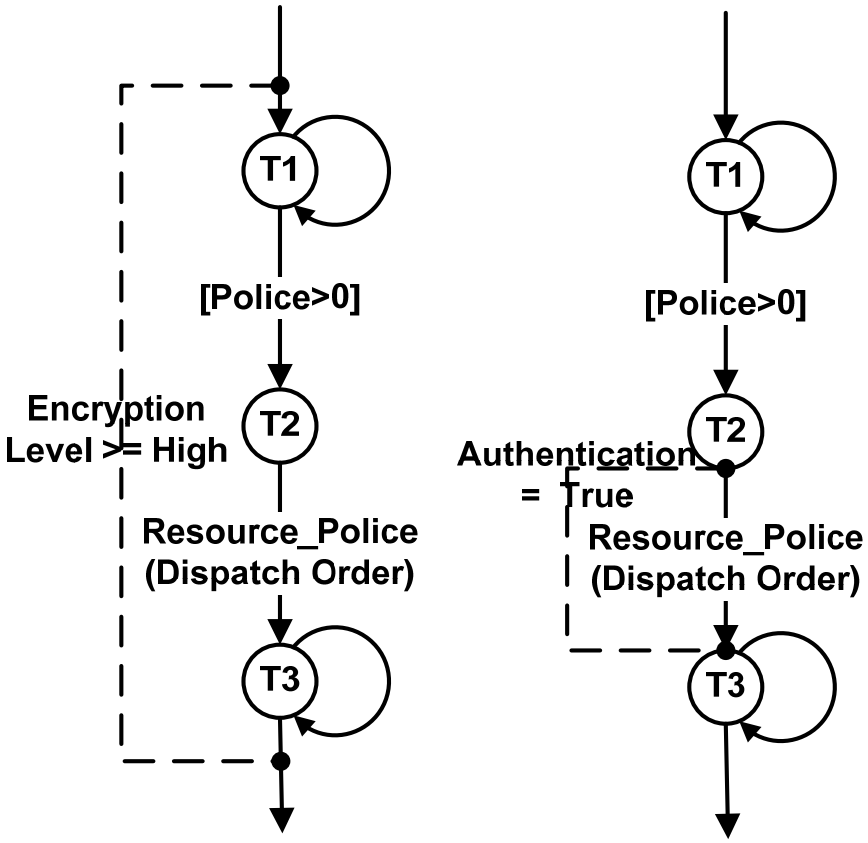

Figure 9. Locally scoped security requirements: (1) all requests to dispatch police shall employ highly-encrypted messages; (2) messages to dispatch the police shall be authenticated.

In the composite web service, NFRs may refer only to the properties or behaviors of some services or some message paths. A *scope* of a NFR property refers to a user-defined subset of the services or service operations to which the NFRs property actually applies. For example, a user of the EMS for a search and rescue incident may require only the service operations of reading and writing the Police Resource to be highly encrypted, rather than globally requiring all service operations to be highly encrypted.

Similarly, that user may require only the message to dispatch the police to be authenticated. In the excerpts in Figure 9, the dotted lines describe the local scopes of these security properties.

The FRs goal model introduced in Section 4.3 is used here as a template to specify NFRs constraints and their scoping information. Since all composite web service candidates have the same workflow as that specified in the goal model, a trace equivalence check can be performed to verify that the composite web services satisfying the FRs also satisfy the required security constraints.

We now derive the security NFRs constraint model from the goal model in Figure 6. We illustrate the process with the NFRs described in Section 4.1, i.e., that a request to dispatch police shall be processed using highly-encrypted message paths. The construction of a NFR property constraint consists of the following steps:

1. Identify the scope in the goal automaton (Figure 6) for this NFRs constraint. For the security NFRs of concern here, we identify the path with the Police_Resource operation in it to be the scope.
2. Label the NFRs constraint for the scope to cover all the operations within the scope. Here we label the path we identified in step 1 as “Encryption Level>=High”.
3. Simplify the model, if possible, by merging the states and transitions unrelated to the scope into a single state with a self-loop.
4. Prune unrelated states and paths. Here we remove the other branches from the graph, yielding the security NFRs automata in Figure 9.

To verify that the composite web service satisfies the security NFR, we must first retrieve the security guarantees of the individual component services. The composite web service produced by the functional composition, e.g., Figure 7, is stored in Web Service Business Process Execution Language (WSBPEL) [65] file. This WSBPEL file records the workflow of the participating component services and associates with their WSDL files, which further describes their service operations. Each service operation, as shown in Figure 7, is associated with a set of NFRs attributes in the WSDL file, namely the security property described in the WS-Security. The security policies named in Figure 7, such as Basic192, are different security policies which each contain the description of an encryption algorithm, a signature key length, a symmetric or asymmetric key and other security attributes. The following XML excerpt shows how a security policy is associated with a service operation:

Security Policy Binding to Service Operations

```
<Policy wsu:Id="medium_secure">
  <ExactlyOne>
<sp:Basic192 ... />
   </ExactlyOne>
</Policy>

<wsdl:binding name="SecureBinding" type="tns:ReservationInterface" >
  <PolicyReference URI="# medium_secure " />
  <wsdl:operation name="Reservation" >...</wsdl:operation>
  ...
</wsdl:binding>
```

In the first part of this piece of code, a Basic192 security solution (described in [4]) is defined in the security policy. In the second part, this policy is bound with the Reservation operation in the WSDL file of the police service. This reservation operation reserves the police resources to dispatch and prevents concurrent allocation of the same units. The binding between the security policy and the service operation allows our approach to retrieve the security policy for each operation in a composite web service for verification.

As shown in Figure 7 and the XML code, the security policies are specified for each operation of each component of a composite service. Note that the security policies of different operations can differ from each other. The resulting sets of security policies (and the associated enforcement algorithms) are collectively used to construct a domain terminology mapping table that allows the different security policies to be compared with each other.

4.4.2 Domain Terminology Mapping

During verification, the security policies of the service operations associated with a service are retrieved from the WSDL specifications of the corresponding service. Construction of a domain terminology mapping table is necessary when the NFRs cannot be simply quantified or is too complex to handle by single variables. The domain terminology mapping table is designed to bridge the gap between the user's requirements (Low, Medium, High, or Critical level encryption) and the specifications described in the web services (algorithms and minimum key lengths). As in the security policy domain,

first we gathered all the security algorithms the service operation can apply, shown in Table 1 [4]. The details of these algorithms and their comparison are described in the WS-Security specification [4]. We categorized the encryption algorithms as medium, high, or critical, based on their minimum symmetric key length. Operations without any encryption algorithm applied are classified as low encryption level. Table 2 defines, for our illustrative example, an excerpt of the different encryption levels specified in WS-Policy [4] and a Boolean option for an authentication feature for the composite web service. These choices associate with different security policies in WS-Security used by the web services.

The security solutions specified in WS-Security are mapped to an encryption level based on the minimum key length of their encryption algorithms. By using a symmetric encryption algorithm, the message sender and the receiver can share a key, in order that only the genuine sender can encrypt and the real receiver decrypt the communication [55]. The optional authentication feature is mapped to the security policies based on whether a symmetric encryption is applied. These mappings are stored in a table for use and reuse in verification that candidate composite services satisfy the security NFRs.

Table 1: Security algorithms in WS-Security (Cited from [4])

Algorithm Suite	[Dig]	[Enc]	[Sym KW]	[Asym KW]	[Enc KD]	[Sig KD]	[Min SKL]
Basic256	Sha1	Aes256	KwAes256	KwRsaOaep	PSha1L256	PSha1L192	256
Basic192	Sha1	Aes192	KwAes192	KwRsaOaep	PSha1L192	PSha1L192	192
Basic128	Sha1	Aes128	KwAes128	KwRsaOaep	PSha1L128	PSha1L128	128
TripleDes	Sha1	TripleDes	KwTripleDes	KwRsaOaep	PSha1L192	PSha1L192	192
Basic256Rsa15	Sha1	Aes256	KwAes256	KwRsa15	PSha1L256	PSha1L192	256
Basic192Rsa15	Sha1	Aes192	KwAes192	KwRsa15	PSha1L192	PSha1L192	192
Basic128Rsa15	Sha1	Aes128	KwAes128	KwRsa15	PSha1L128	PSha1L128	128
TripleDesRsa15	Sha1	TripleDes	KwTripleDes	KwRsa15	PSha1L192	PSha1L192	192
Basic256Sha256	Sha256	Aes256	KwAes256	KwRsaOaep	PSha1L256	PSha1L192	256
Basic192Sha256	Sha256	Aes192	KwAes192	KwRsaOaep	PSha1L192	PSha1L192	192
Basic128Sha256	Sha256	Aes128	KwAes128	KwRsaOaep	PSha1L128	PSha1L128	128
TripleDesSha256	Sha256	TripleDes	KwTripleDes	KwRsaOaep	PSha1L192	PSha1L192	192
Basic256Sha256Rsa15	Sha256	Aes256	KwAes256	KwRsa15	PSha1L256	PSha1L192	256
Basic192Sha256Rsa15	Sha256	Aes192	KwAes192	KwRsa15	PSha1L192	PSha1L192	192
Basic128Sha256Rsa15	Sha256	Aes128	KwAes128	KwRsa15	PSha1L128	PSha1L128	128
TripleDesSha256Rsa15	Sha256	TripleDes	KwTripleDes	KwRsa15	PSha1L192	PSha1L192	192

Table 2. Security terminology mapping table excerpt

Encryption Level	Security Algorithm	Min Key Length	Symmetric
Low	None	0	No
Medium	Basic128	128	No
	Basic128Sha256	128	Yes
High	Basic192	192	No
	Basic192Sha256	192	Yes
	TripleDes	192	No
	TripleDesSha256	192	Yes
Critical	Basic256	256	No
	Basic256Sha256	256	Yes

Table 2 (Continue). Security terminology mapping table excerpt

Authentication	Security Algorithm	Min Key Length	Symmetric
False	None	0	No
	Basic128	128	No
	Basic192	192	No
	TripleDes	192	No
	Basic256	256	No
True	Basic128Sha256	128	Yes
	Basic192Sha256	192	Yes
	TripleDesSha256	192	Yes
	Basic256Sha256	256	Yes

4.4.3 NFRs Aggregation Rules

An aggregation rule combines the valuations of non-functional attributes of component services that fall within the scope of the corresponding NFRs [69]. For example, in the case of the EMS encryption requirement, the aggregation rule is defined

as: the encryption level of the composite service is the minimum encryption level of the component services that fall within the scope of the NFRs for EMS encryption. We implement this aggregation rule in the following aggregation script, which is used by the security NFRs verification algorithm.

Encryption Level Aggregation Script:

```
Define Low=0
Define Medium=1
Define High=2
Define Critical=3
Define Initial_Security_Level=3
Int Aggregate(Int current_security, Int new_security)
{if(current_security>new_security){
current_security=new_security; }
return current_security;
}
Bool isSatisfied(Int current_security, Int desired_security)
{
If(current_security< desired_security) return false;
return true;
}
```

4.4.4 NFRs Verification Algorithm

In order to verify that NFRs is satisfied in a candidate composite web service known to satisfy the FRs, we compose the automata representation of the composite web service and the NFRs property. In our approach, all automata compositions are synchronous, i.e., multiple automata can make progress in parallel for each step [23]. The problem of whether a composite web service conforms to a desired NFR is addressed by calculating the synchronous product of automata representing the composite web service with those representing the corresponding NFR. The composite web service is said to satisfy the NFRs if and only if the sequence of properties over non-functional attributes as represented by the NFR-automata is also present in the automaton of the composite web service. The verification process can be viewed as an equivalence check.

Definition 2: Given two automata, $FSA_i = (S_i, s_{0i}, \Delta_i, P_i, F_i)$, for i ∈ {1, 2}, their product is another FSA denoted by $FSA_1 \times FSA_2 = (S_{12}, s_{012}, \Delta_{12}, P_{12}, F_{12})$, where $S_{12} \subseteq S_1 \times S_2$, $s_{012} = (s_{01}, s_{02})$, $P_{12} = P_1 \cup P_2$, $F_{12} = \{(s_1, s_2) \mid s_1 \in F_1, s_2 \in F_2\}$. Finally, $s_1 - \phi_1 \rightarrow s_1' \in \Delta_1$ and $s_2 - \phi_2 \rightarrow s_2' \in \Delta_2$ and $(s_1, s_2) - \phi_1 \wedge \phi_2 \rightarrow (s_1', s_2') \in \Delta_{12}$.

The automata composition concept can be used in two ways: (1) to verify a NFR automaton with a composite web service, and (2) to verify the consistency of two or more NFRs automata if two or more NFRs properties need to be combined.

When the automata composition algorithm encounters any service within the scope, then, if there is an aggregation rule associated with that NFR, the aggregation script that implements the aggregation rule is included in the verification algorithm.

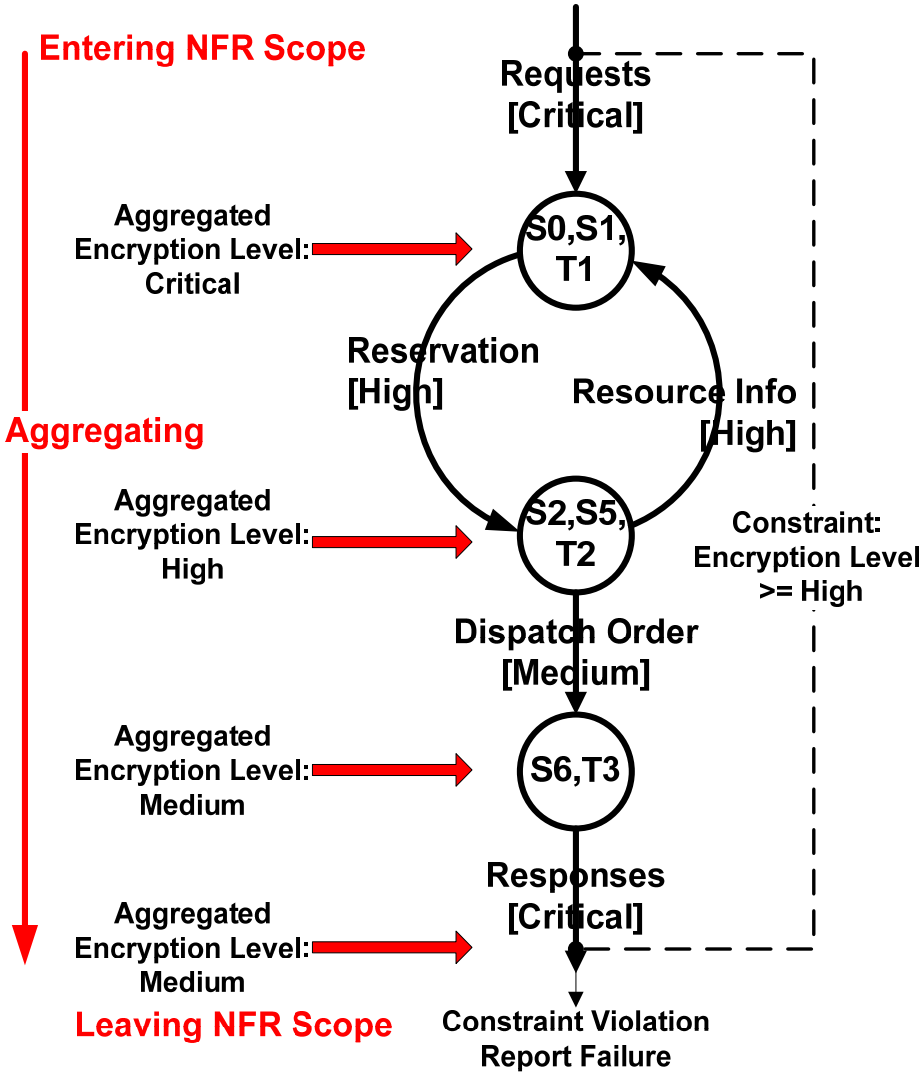

Figure 10. Security property aggregation and composed automata

The verification algorithm uses an entrance condition to detect entry into the relevant scope for the current NFR, and an exit condition to trigger evaluation of whether

the current NFRs has been satisfied in the composite web service. Figure 10 shows the composed automaton and how aggregation is applied to the scope range for the security NFRs in the EMS example during automata composition. During the composition of the composite web service automaton and the security property automaton, the states and transitions of both automata merge when they share the same predicates. When the composition enters the scope of the security constraint, the encryption level value starts to aggregate for each merged transition according to its aggregation rule and the terminology mapping table. When the composition algorithm leaves the scope, the aggregated encryption level value is compared to the value required by the constraint to evaluate its satisfaction (here, a High encryption level). All these tasks are described inside the aggregation script for the implementation.

The algorithm to verify a NFR property automaton against a composite web service automaton is described in pseudo code as follows:

Security Level Verification Algorithm:

Let $FSA_c = (S_c, s_{0c}, \Delta_c, P_c, F_c)$ *be the composite web service automaton.*

Let $FSA_n = (S_n, s_{0n}, \Delta_n, P_n, F_n)$ *be the security constraint automaton.*

Let $FSA_p = \emptyset$ *be the initial composed automata.*

1: Load Security Level Aggregation Script
2: Set current_security= Initial_Security_Level
3: Initial state of FSA_p *:* $s_{0p} = \{s_{0c}, s_{0n}\}$
4: Call Combine_States(s_{0c}, s_{0n}, s_{0p} *)*
5: If isSatisfied(current_security, required_security) report success, else report failure.

6: proc Combine_States(s_{0c}, s_{0n}, s_{0p}){

*7: /*Select the states with satisfied guard conditions */*

8: ForEach $\phi_c \in 2^{Pc}$, $s_c' \in S_c$ *s.t.* $\forall s_n' \in S_n$: $s_{0n} - \phi_n \rightarrow s_n'$, $s_{0c} - \phi_c \rightarrow s_c'$ *and* $\phi_c \subseteq \phi_n$

9: Create a new state $s_p' = \{s_c', s_n'\}$ *for* FSA_p

10: If s_n' *is a trap state, return and report failure.*

11: Create a new transition $s_{0p} - \phi_c \rightarrow s_p'$ *for* FSA_p

12: If InScope,

13: If the guard condition ϕ_c *is a service operation, retrieve its security policy name as SName from its WSDL file.*

14: Retrieve the security level as new_level by searching for SName in the security terminology mapping table.

15: current_security=Aggregate(current_security, new_level)

16: Call Combine_States(s_c', s_n', s_p')

17: End ForEach

18: }

Note that a verification of the security property in Figure 10 on the composite web service in Figure 7 returns a "Failure" result, because the aggregation result of the encryption level for this composite web service candidate is Medium in its required scope, which is lower than the required High.

4.4.5 Generalization and Limitation

Beside security constraints, our QoS constraint model is also able to handle other QoS quantitative hard constraint for web service compositions, such as performance and reliability, which can be checked using information available at composition time. The modeling process is the same as modeling a security constraint. The only things to

modify in order to handle other QoS constraints are the terminology mapping table and the aggregation rules. The terminology mapping table varies among projects and depends on the domain engineer's decisions. For example, for the QoS attribute throughput, we can map the requirement terminologies to the implementation level terminologies as shown in Table 3.

Table 3: Performance terminology mapping table excerpt

Performance	Throughput
High	> 1 Mbits/s
Medium	100 Kbits/s to 1 Mbits/s
Low	< 100 Kbits/s

Some common aggregation rules for QoS attributes for web service compositions are provided in Jaeger et. al. [24]. We can then use the aggregation rule of throughput defined by Jaeger et. al. [24] as:

- For sequential or parallel composition, the aggregated throughput is always the lowest throughput in the concerned set of component services.
- For a loop, the aggregated throughput is the same as the path throughput inside of the loop.

The construction process for the throughput constraint automaton is exactly the same as the process for constructing the security constraint automaton. Figure 11 gives an example of a globally scoped performance constraint.

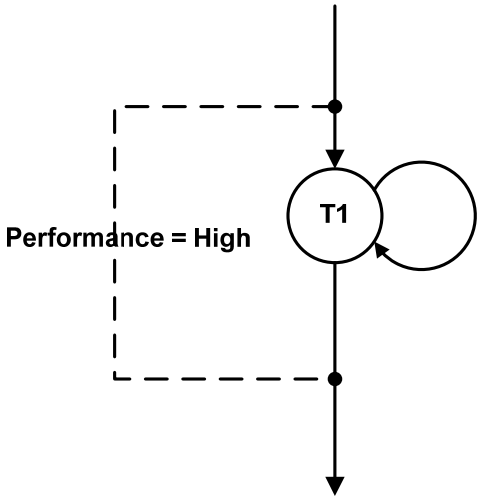

Figure 11. Global performance constraint

However, our approach is also constrained by our research scope. Because our model focuses on QoS quantitative hard constraints for static web service compositions, we are unable to handle non-functional properties outside of this scope. If the QoS information for component web services is only available for dynamic web service composition, such as the message delay, our approach is unable to handle this QoS constraint at static composition time. For example, the requirement that "All messages shall be received with in x milliseconds after being sent" can only be checked at runtime. If the QoS constraint is a qualitative preference, such as "high throughput is preferred to low throughput", our model cannot represent this constraint.

4.4.6 Handling Multiple Properties

In complex composite web services, there often exists more than one user specified non-functional requirement to verify, such as security or availability constraints with different scopes. An interesting issue is how to handle the multiple property automata efficiently. Given a candidate composite web service from the search space, the most straight-forward way to verify each property automata is to verify the properties independently and sequentially. This strategy, Independent Composition (INC), is shown in the first column in Table 4. INC calculates each property independently with the candidate composite web service and discards the intermediate calculation results after each inner loop. INC returns verification failed at the first unsatisfied NFRs property. It is most suited to a search space with few candidates. A disadvantage of INC is that even if the properties are inconsistent, such that no composite web service can satisfy them all, the algorithm has to explore the entire search space before telling the user that no composite web service satisfies the NFR.

To overcome this weakness of the INC algorithm, we introduce the Two-Stage Composition (TSC) algorithm (the second column of Table 4) which detects property inconsistency before performing the verification. TSC composes all property automata into one combined property automaton prior to verification. In this way, the first inconsistency found during property composition will terminate the verification process, and the property automaton causing this failure will be captured and returned to the user for possible modification of the NFR. In addition, TSC can assist with efficient

verification, since property automata may share the same predicates on the transitions. The more predicates shared by the property automata, the fewer states the combined automaton will have. Another advantage of using TSC is that the combined property automata can be reused to verify multiple candidate composite web services or when NFRs are updated.

A third strategy, the Big-Bang Composition (BBC) algorithm, replaces the sequential verification of the properties in INC with parallel verification (the third column in Table 4). BBC composes all the automata including the candidate composite web service and the property automata synchronously so that the earliest reachable trap state in any of the properties can terminate the verification by returning a failure.

A combined strategy can be applied using a smart switcher, which pre-calculates the verification overhead when provided a group of composite web service candidates and a group of NFRs properties. Generally speaking, INC and BBC are useful for detecting inconsistencies in the candidate composite web services quickly, whereas TSC is most useful in quickly locating inconsistencies in the user-defined NFRs.

Table 4. Three strategies of verification on multiple properties

	Independent Composition	Two-Stage Composition	Big-Bang Composition
Pseudo-Code	1. For each candidate composition s_i in S 1) For each property p_j in P 2) Verify this property by calculating $s_i \times p_j$ 3) If the verification failed, jump to the next s_i 4) End For Each 2. Output s_i and terminate 3. End For Each 4. Output "No composition satisfies the NFR"	1. Take automata $q = p_0$ 2. For j = 1 to k 3. Calculate $q \times p_j$ and Save the result as q 4. If there exists only one termination state in q and it's a trap state in the safety property, Output "Property p_j is Inconsistent with the other properties" and Terminate 5. End For 6. For each candidate composition s_i in S 7. Verify the property by calculating $s_i \times q$ 8. If the verification failed, jump to the next s_i 9. Else Output s_i and terminate 10. End For Each 11. Output "No composition satisfies the NFR	1. For each candidate composition s_i in S 2. Verify the property by calculating $s_i \times p_1 \times p_2 \times \ldots \times p_n$. All automata are composed synchronously rather than pair-wisely. When a trap state is reached, the composition terminates and returns failure. 3. If the verification failed, try the next s_i 4. Else Output s_i and terminate 5. End For Each 6. Output "No composition satisfies the NFR"
Complexity	$O(\sum_{i=1}^{n}\sum_{j=1}^{k}(\lvert s_i\rvert \times \lvert p_j\rvert))$	$O(\sum_{i=1}^{n}(\lvert s_i\rvert \times (\prod_{j=1}^{k}\lvert p_j\rvert)))$	$O(\sum_{i=1}^{n}(\lvert s_i\rvert \times \prod_{j=1}^{k}\lvert p_j\rvert))$

S is the set of all compositions. S = {s1, s2, s3, …..sn}

n is the number of compositions in the search space

P is the set of all NFRs. P = {p1, p2, p3, …..pn}

k is the number of NFRs

4.5 Design of Implementation

Figure 12 presents the implementation design of the NFR verification system for WSC.

The left side of the figure shows the four input data of the verification system. The service provider profile retrieved from UDDI could have non-functional properties, such as location, about the web service. The WSDL provides the functionalities provided by the web service including operations, actions and parameters. WSLA and WSPolicy provide other QoS properties related to the web service.

The right side of the figure shows the two inputs from domain knowledge. The QoS domain knowledge model includes the terminology mapping tables, which quantify the QoS attributes. The aggregation rules, introduced in Section 4.4.3, define how to aggregate and evaluate the QoS attributes of the WSC.

The upper portion of the figure describes the service composition input, which begins with a functional goal model. We then define our QoS constraints for the WSC from this goal model using the approach described in Section 4.4.1. The service compositions are generated from the goal model using functional composition algorithms.

The core asset of this system is the constraint verification module, which takes all the inputs and calls the verification algorithm described in Section 4.4.4. Finally, the algorithm outputs the verification results: either verification failure or a set of valid WSCs.

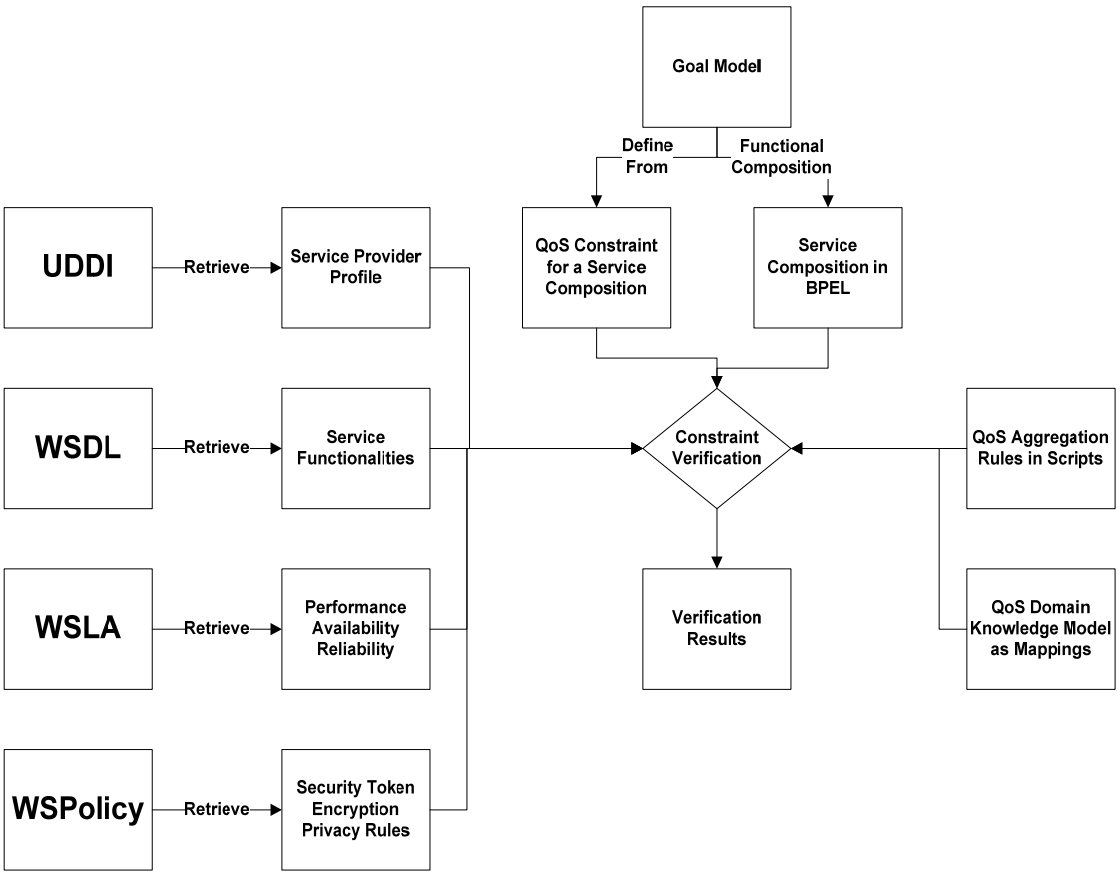

Figure 12. Implementation design for the NFRs verification system for WSC

4.6 Related Researches

In this subsection, we compare our approach with existing solutions on verifying the web service compositions.

Foster et. al. [18][19] have introduced a similar approach to verify the properties of web service compositions. The web service composition described in the WSBPEL is translated into a finite state process (FSP) model. The behavior property of the web service composition is modeled in a message sequence chart (MSC). This MSC model is then translated into a FSP model as well. An equivalence check of the two FSP models is

conducted to verify the compliance of this property in the web service composition. Our model has greater expressive power in the following three aspects: Being able to calculate QoS aggregations along the message paths and verify the QoS constraints, considering the cases in which some properties are locally scoped and in handling multiple properties and their consistency checking.

Additionally, the construction of the MSC model is required to build from scratch and defined for each web service composition. Our property model is build from the goal model. The goal model captures all the functionalities and the business process of all valid web service compositions. Thus, our property model is easier to define and more general in verifying multiple web service compositions derived from the same goal model.

Jaeger et. al. [24][25] has introduced the QoS aggregation rules for web service compositions. They took the functionalities of the web service composition as a hard constraint and compared several different composition optimization algorithms based on the aggregated QoS values. All QoS attributes are considered as qualitative soft constraints for optimizations where our approach focuses on the QoS quantitative hard constraints.

4.7 Conclusion

This chapter shows how an automatically generated composite web service of independently developed web services can be verified to meet the non-functional security requirements imposed by the user as hard constraints. The approach described here enables this verification by lifting the NFRs analysis from the level of individual services to the level of the search space of candidate composite web services obtained from the

functional requirements. The primary limitations of this approach are (1) that it currently can only handle those types of NFRs which can be specified in WSDL and WSDL's auxiliary specifications, such as WS-Policy, WS-Security, WS-Trust and WSLA, and (2) that domain expert knowledge is needed to build the terminology mapping table. We hope to ease these restrictions in future work.

CHAPTER 5. PRODUCT-LINE-BASED REQUIREMENTS CUSTOMIZATION FOR WEB SERVICE COMPOSITIONS[2]

Customizing web services according to users' individual functional and non-functional requirements has become increasingly difficult as the number of users increases. This chapter introduces a new way to customize and verify composite web services by incorporating a software product-line engineering approach into web-service composition. The approach uses a partitioning similar to that between domain engineering and application engineering in the product-line context. It specifies the options that the user can select and constructs the resulting web-service compositions. By first creating a web-service composition search space that satisfies the common requirements and then querying the search space as the user selects values for the parameters of variation, we provide a more efficient way to customize web services. A decision model, illustrated with examples from an emergency-response application, is created to interact with the customers and ensure the consistency of their specifications. The capability to reuse the composition search space may also help improve the quality

[2] The work presented in this chapter is adapted from H. Sun, R. Lutz and S. Basu, "Product-Line-Based Requirements Customization for Web Service Compositions", 13th International Software Product Line Conference (SPLC '09), 2009.

and reliability of the composite services and reduce the cost of re-verifying the same compositions.

5.1 Introduction

Commercial web services often have a very large user base. How to customize these web services according to the users' individual requirements becomes increasingly difficult as the number of users increases. This chapter introduces a new way to customize composite web services to users' requirements by applying a software product line engineering approach to the web service composition domain.

Most existing practical mechanisms for synthesizing composite services have been deployed taking into consideration their desired functional requirements [14][28]. Functional requirements (FRs) describe how a system should behave during operation, while non-functional requirements (NFRs) describe constraints on the quality attributes of the system's operation. NFRs can be broadly classified as soft and hard constraints. Hard constraints refer to the set of NFRs that must be satisfied by a composite service, while soft constraints deal with user preferences and trade-offs among NFRs [58]. In this chapter, we are concerned with hard constraints.

In order to customize a web service in a specific domain application (e.g., a travel agency service), FRs and NFRs may both vary somewhat among individuals. With existing methods of composing web services on FRs and NFRs [28][59], the verification of the variations in the services tends to incur a heavy overhead that can reduce the performance of the composed services. We instead seek to exploit the reuse of certain web service compositions, where the composition has already been verified. The goal is

for each small variation to no longer trigger an entirely new verifiable composition process. Thus, for customizable web services, we need a lightweight and low-overhead solution to generating web service compositions satisfying users' customized requirements.

A web service composition shares several similarities with a product line. A commercial web service provider has a specified, shared set of functional and non-functional requirements that should be fulfilled by every possible service composition provided to the user. For example, a travel agency web service has to include a membership service and a payment service as common functionalities. Secured web service communication is also a common NFR that exists in all compositions.

These common FRs and NFRs can be mapped to commonalities in a product line by taking all possible compositions that satisfy the common FRs and NFRs as products of a product line. Each user of the web service can also impose customized FRs and NFRs, according to his or her own preference, within the variations provided by, e.g., the travel-agency service. Thus, a user may require flights to be on a specific airline or airplane model for the flight-booking service. These variations in FRs and NFRs, if predictable by the service provider, can be specified as variabilities of a product line. The user then only needs to decide the value for each variation point in order to generate a customized service composition.

In previous chapter, we have separated the verification of NFRs for a web service composition from the verification of FRs in order to reduce the complexity of requirements verification. In pursuing a two-stage solution we have subsequently tried to

isolate the requirements likely to be changed, or customized, by the users. Product line engineering offers a strong framework for this purpose.

Software product line engineering (SPLE) identifies the common assets of a series of products as commonalities and views the optional and alternative assets as variabilities. Taking advantages of these two concepts, SPLE reduces the development cost and time by reusing the commonalties and the variabilities across the product line [64]. We build in this work on the FAST (Family-Oriented Abstraction, Specification, and Translation) approach to constructing a product line. FAST uses three key artifacts to enable rapid generation of a product given a partially ordered specification from the customer [64][65]:

1. The *commonality and variability analysis* (CVA), which identifies the commonalities and variabilities for the product line and the dependencies and constraints among them.

2. The *mapping relations* between the values of the variations and the modules of the product lines (which may be one-to-one, one-to-many, or many-to-many mappings) and the *uses-relations* of the modules and their implementations (which describe the other modules used if one module is selected). For example, one module may have different ways of being implemented under the constraints of performance or platforms.

3. The *decision model* with which the customer interacts to specify and construct a new product in the product line.

We extend the process by adapting the workflow used to compositionally generate a product-line system to the customized composition of web services:

- The user decides the values of the variabilities of a product in either a random or preferred sequence.
- The values of these variabilities are used to prune and determine other variabilities by following the dependencies and constraint rules.
- A consistent value set for the variabilities is thus obtained. By tracing the mapping between the values of variabilities and the components of the product line, a component set is selected.
- By tracing the uses-relations of the selected component set, all necessary components for a product are selected.
- The implementation of the necessary component set is gathered, compiled and published as the final product.

Web service composition takes advantage of SOA (Service Oriented Architecture) [15] to achieve complex functional and non-functional requirements by selecting and composing qualified component services provided by service providers. A service broker (UDDI) is responsible for the registration and look-up of all component services with descriptions.

By considering a customizable web service composition as a product line, we are better able to handle the customized requirements from the users. Normally, web service

compositions involve verification of the functional and non-functional requirements on demand. This means that whenever there is a variation, change, or new requirement for the services, the composition algorithm needs to re-compose and re-verify the web services against the variation, change or new requirement.

Using the SPLE approach, we show how we can successfully divide the composition and verification process into two stages. The first stage treats the construction of the functional *commonalities* of the web service compositions and verifies them against the common NFRs. The second stage takes as input the verified compositions from the first stage and then verifies the candidate composition in the result set to meet the *variable* functional and non-functional requirements from the user.

Since part of the computation has already been done in the first stage as pre-processing, the verification overhead caused by the customizations in the mass-user-based web services is much lower than the traditional one-stage composition and verification process. The two-stage SPLE approach has better efficiency because, for a customization requirement, the second stage acts like a service composition verifier rather than a service composition generator. It essentially hides the large computation overhead in the first stage when constructing the search space for the second stage.

The rest of the chapter is organized as follows. Section 5.2 describes our approach to applying SPLE to web service composition and illustrates it by application to an emergency-response service. Section 5.3 discusses the advantages and limitations of our approach and describes the steps needed for future evaluation. Section 5.4 provides a brief conclusion.

5.2 Approach

Figure 2, duplicated here as Figure 13 for convenience, shows an overview of our approach, which is described briefly here. Subsequent subsections provide a more detailed account of each of the major steps in Figure 13. The steps are labeled in the figure with the number of the subsection that describes that step.

Our approach contains two workflows, one for the web service product developer (shown at the top of the figure) and one for the user (shown at the bottom). This approach provides a two-stage process. From the point of view of product line development, the development process is divided into the domain engineering phase and the application engineering phase. From the point of view of web-service design, the process is divided into the preparation stage (before the construction of the search space) and the customization stage (after the construction of the search space).

At the top half of the figure, i.e., the domain engineering phase, the developer performs a commonality and variability analysis (CVA) of the system requirements. The results of the CVA include a formalized specification of all the common and variable functional and non-functional requirements. Each variability has some associated parameters to configure, called *parameters of variation*. Some of these parameters may have dependencies or tradeoffs with other parameters of variation. We model this dependency of variations in a *variability dependency graph*.

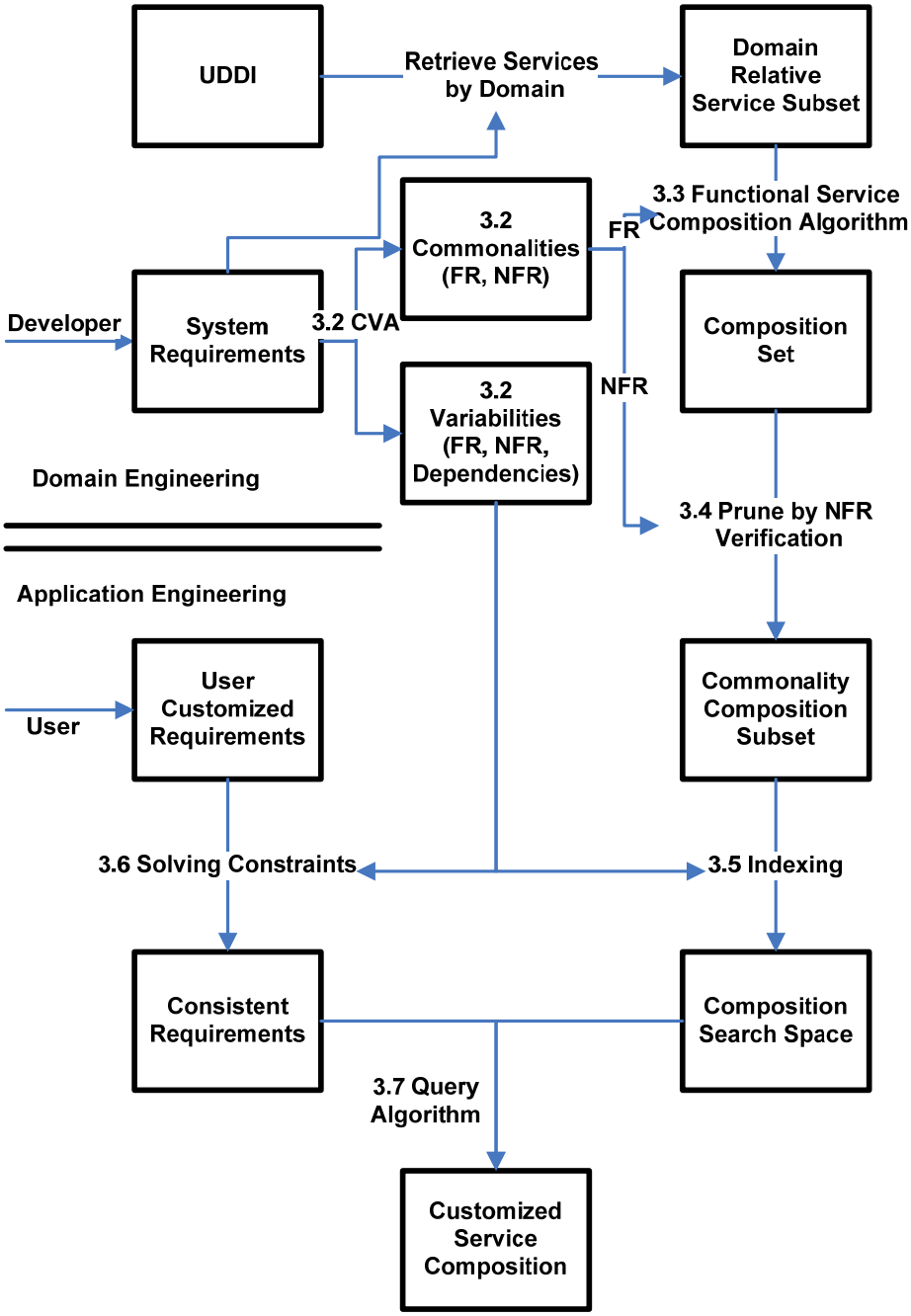

Figure 13. Overview of approach

Each commonality and parameter of variation is also associated with a set of service compositions that satisfy them. We call this relation the *mapping-relation*. The results of the CVA are captured in a product-line decision model. Given the system requirements, the developer can identify the component web services that are relevant to the domain and the system.

For the application engineering phase, the services retrieved from the service broker are composed according to the common functional requirements by means of a modified version of the goal model and a functional service composition algorithm from [44]. The algorithm outputs all possible service compositions that satisfy the common functional requirements. By verifying all the compositions of this set against the common NFRs, we prune the composition set into a smaller set in which each composition satisfies all commonalities. This set serves as the composition search space for the variability, and we call it the *commonality composition set* (subset).

In order to improve the verification efficiency for the runtime customization, we do an indexing on each parameter of variation mapping to a subset of the search space. A composition subset associates with a parameter of variation if and only if all compositions in this subset satisfy the variability set by this parameter. The result of this preparation phase is the composition search space that is ready for runtime user customization.

After implementation of the web services, the user can access a default web service with basic functionalities. The user can then customize the functionalities and non-functional requirements by setting all the parameters of variation in the decision

model. By interacting with the decision model, the input requirements from the user are always kept consistent through solving the constraints in the variability dependency graph. The consistent specification of the variabilities is fed into a query algorithm to search valid compositions in the composition search space. The output of this algorithm is either the customized service composition satisfying all the requirements from the user or a report to the user of a failed composition attempt.

5.2.1 Illustrative Example

We use the small system introduced in Chapter 3, the Emergency Management System (EMS) to illustrate the basic concepts of our approach.

As a reminder, an EMS consists of several different units: the Field Officer Service, the Request Dispatch Service (Dispatcher for short) and services for emergency handling, including an Ambulance Dispatch Service, a Fire Station Dispatch Service and a Police Dispatch Service. The functional requirements are to dispatch ambulance(s), fire truck(s) and police to a location upon request. These requests are specified and sent by the Field Officer through a service in a mobile terminal or a PDA.

The Dispatcher has three types: the normal dispatch service, which is responsible for routine situations; the speed-line dispatch service, which has low communication delay, compared to the normal dispatcher and is used for urgent dispatches; and the highly-secured dispatcher, which is used for national security related cases. The dispatcher service can also invoke a GPS-MAP service and other third party services. Figure 4 shows a sample structure for EMS service composition. The services connected by the dashed lines represent the optional services in a valid composition.

5.2.2 Commonality and Variability Analysis

The commonalities that are shared across all the EMS service compositions are listed according to their label, type (Functional requirements as "FR" and Non-functional requirements as "NFR"), description and service mappings.

Commonalities:

- C1
    - FR
    - There must be a Field Officer Service.
    - Field Officer Service = any service in Field Officer Service set {FO1, FO2, FO3…}.
- C2
    - FR
    - There must be a Dispatcher Service.
    - Dispatch Service = any service in normal Dispatcher Service set {DSN1, DSN2, …} or speed-line Dispatcher Service set {DSSL1, DSSL2, …} or Highly-secured Dispatcher Service set {DSHS1, DSHS2, …}.

- C3:
    - FR
    - There must be one or more emergency services.
    - An emergency service = any service in the Fire Station Dispatch Service set {FS1, FS2, …} or in the Ambulance Dispatch Service set {AS1, AS2, …} or in the Police Dispatch Service set {PS1, PS2, …}.
- C4:
    - NFRs
    - All services must support at least 128-bit encryption in its service description.

The variabilities of the EMS are listed according to their label, type, description, parameter(s) of variation and any dependencies among these parameters.

Variabilities:

- V1:
    - FR
    - Type of dispatch service
    - {Normal, Speed-line, Highly-secured}.
    - If V1 is Highly-secured, then V6 is High. If V1 is Speed-line, V7 is Low.

- V2:
    - FR
    - Existence of Fire Station Dispatch service
    - {True, False}
    - If V2 is False, (V3 or V4) is True
- V3:
    - FRS
    - Existence of Ambulance Dispatch service,
    - {True, False}.
    - If V3 is False, (V2 or V4) is True
- V4:
    - FR
    - Existence of Police Dispatch service
    - {True, False}.
    - If V4 is False, (V2 or V3) is True

- V5:
    - FR
    - Existence and type of a third-party service.
    - {N/A, GPS-MAP, TP1, TP2,…}.
    - If V5 is N/A, V8 is N/A.
- V6:
    - NFR
    - Security Level
    - {Medium, Med-High, High}.
    - If V6 is High, V1 is Highly-Secured and V9 is 256. If V6 is Med-High, V9 is 256. If V6 is Medium, V9 is 128.
- V7:
    - NFR
    - Delay of service communication
    - {Low, Med-Low, Medium}.
    - If V7 is Low, V1 is Speed-line.

- V8:
    - NFR
    - Range constraint between the field officer and the emergency service location
    - {N/A, Near, Medium, Far}.
    - If V8 is not N/A, V5 is GPS-MAP.
- V9:
    - NFRs
    - Type of encryption
    - {128, 256}. If V9 is 128, V6 is Medium. If V9 is 256, V6 is Med-High or High.

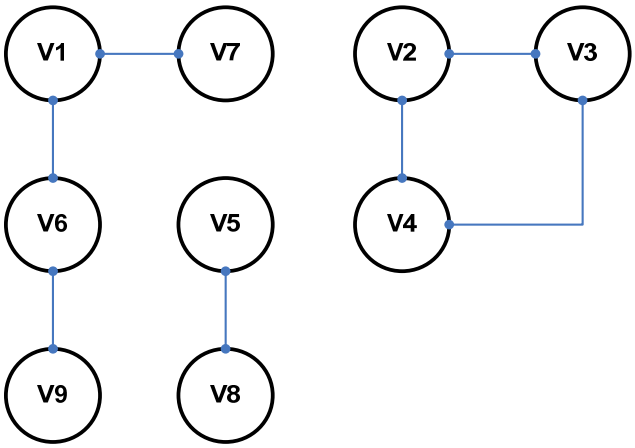

Figure 14. Dependency graph of variabilities

We model the dependencies of the parameters in the CVA in a dependency graph (see Figure 14). In the dependency graph, each node represents a variability and contains the information of all the parameters related to this variability. The edges between the nodes represent and contain the constraints between the two variabilities. The graph may not be fully connected. A graph-walk algorithm (described in later section) is used to traverse the sub-graphs in a random order or a user-preferred order to solve all the constraints.

To reduce the workload and complexity for the later verification process, we typically design the parameters of variation by translating integer or real number values into enumerated parameters by categorizing user requirements, as is done with the Range Constraint V8, above. The range can be an integer of miles from 1 to 100, but is represented as only three values: Near (within 10 miles), Medium (10 to 50 miles) and Far (51 to 100 miles).

5.2.3 Goal Model of the Common Functionalities

In order to generate all the service compositions that satisfy the commonalities, we need to represent the common functionalities, here C1, C2 and C3, in the goal model.

Figure 15 shows the goal model for the common functionalities in EMS. Note that we have preserved the possibility of using third-party services to implement the functionality ThirdPartyService (OtherData) in the goal model. The functional composition process finds appropriate component services to implement these abstract methods in the goal model in accordance with the service mapping table. This service mapping table is constructed together with the goal model by looking up the CVA results.

For example, a mapping is: ThirdPartyService = any element from {N/A, GPS-MAP, TP1, TP2, …}.

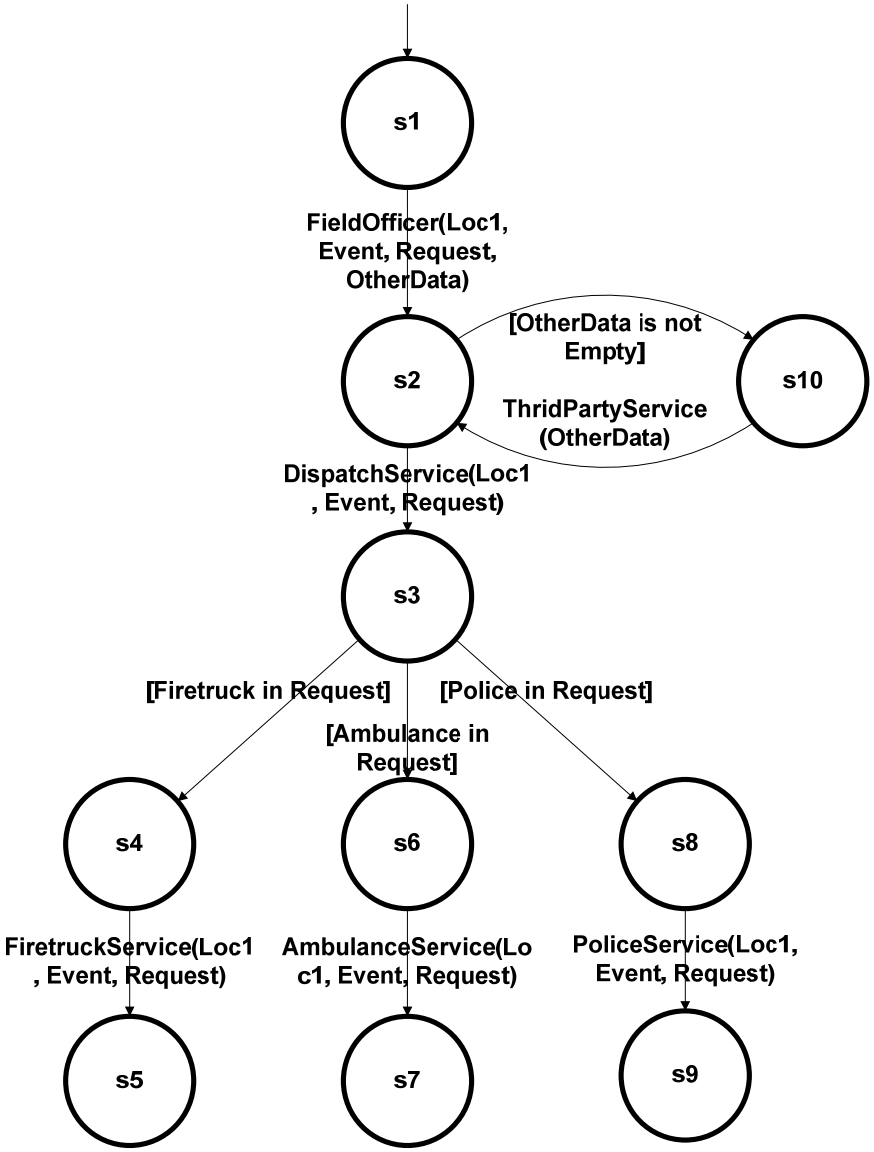

Figure 15. Goal Model for common functionalities

We apply a variant of the choreography-based web service composition algorithm from our previous work [44] on this goal model. The change is that we here take advantage of the availability of the service mapping table to generate all the compositions that satisfy the FRs commonalities rather than generating just one such composition. The result of this step, as shown in Figure 13, is a web service Composition Set containing all compositions satisfying the common functional requirements.

5.2.4 Verification Algorithm of Non-functional Constraints

The next step toward constructing a search space is to verify the common NFRs. The automation in Figure 16 shows an example of a NFR that requires that any ambulance service in the composition shall respond within 600 seconds.

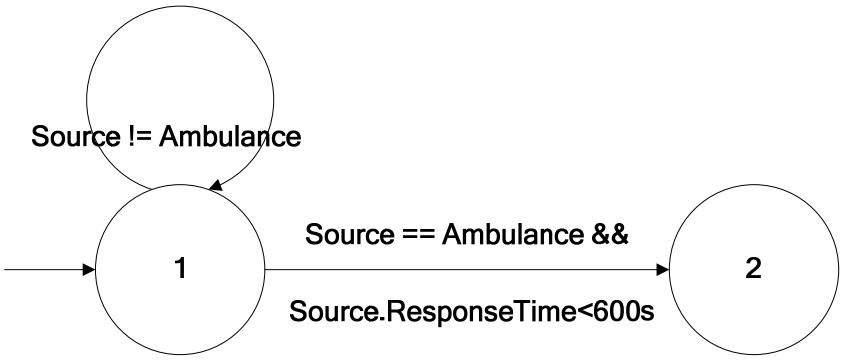

Figure 16. Transition guard with constraint on the service attributes

In previous section, we have described how to model both a service composition and a non-functional constraint as automata and how to verify the NFRs constraint by composing the two automata. The verification is an automata equivalence check. This verification method unifies the constraint models by converting all liveness constraints

into safety constraints. It also handles cases where there are multiple constraints to be verified.

$FSA = (S, s_0, \Delta, P, F)$ where S is the finite set of states, $s_0 \in$ S is the start state, and $\Delta \subseteq S \times 2^P \times S$ is the transition relation of the form $s - \phi \rightarrow s'$ such that $s, s' \in S$, and $\phi \in 2^P$ is a subset of propositions P. Finally, $F \subseteq S$ is the set of final states.

Given two automata, $FSA_i = (S_i, s_{0i}, \Delta_i, P_i, F_i)$, for $i \in \{1, 2\}$, their product is another FSA denoted by:

$$FSA_1 \times FSA_2 = (S_{12}, s_{012}, \Delta_{12}, P_{12}, F_{12}) \text{ where}$$

$$S_{12} \subseteq S_1 \times S_2, s_{012} = (s_{01}, s_{02})$$

$$P_{12} = P_1 \cup P_2$$

$$F_{12} = \{(s_1, s_2) | s_1 \in F_1, s_2 \in F_2\}$$

$$s_1 - \phi_1 \rightarrow s_1' \in \Delta_1$$

$$s_2 - \phi_2 \rightarrow s_2' \in \Delta_2$$

$$(s_1, s_2) - \phi_1 \wedge \phi_2 \rightarrow (s_1', s_2') \in \Delta_{12}$$

Figure 17 shows an example of the product of two automata with the service composition automaton on the left, the constraint automaton in the middle and their product on the right. During the calculation of the automata product, if a trap state of the safety constraint is reached, the verification has failed. Otherwise, the constraint is satisfied by the candidate composition.

For example, by applying this verification technique for the common NFRs C4 to every composition in the Composition Set retrieved, we prune out any compositions with service that violates the requirement for at least 128-bit encryption. The result of performing this verification on the other NFRs, as well, is a set of candidate compositions that satisfies all commonalities, both functional and non-functional, labeled the Commonality Composition Subset in Figure 13.

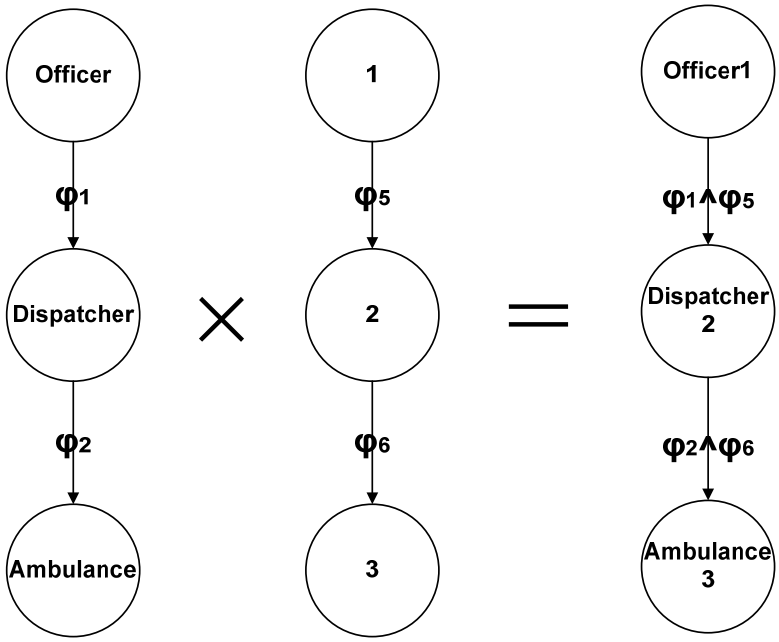

Figure 17. Product of a service composition and a constraint

5.2.5 Search Space Construction by Indexing

In order to construct the composition search space from the Commonality Composition Subset, we next need to index every parameter of variation. By indexing, we mean the creation of a mapping from each parameter of variation to a further subset of the Commonality Composition Subset such that that any composition in this subset

satisfies the variability set by this parameter. To do this, we apply the verification technique introduced in 3.4 to prune the common set to this subset. The results are stored in the Search Space table for each parameter of variation. If the value for a parameter of variation depends on other parameters of variation, we also check those dependencies, as shown in the following example.

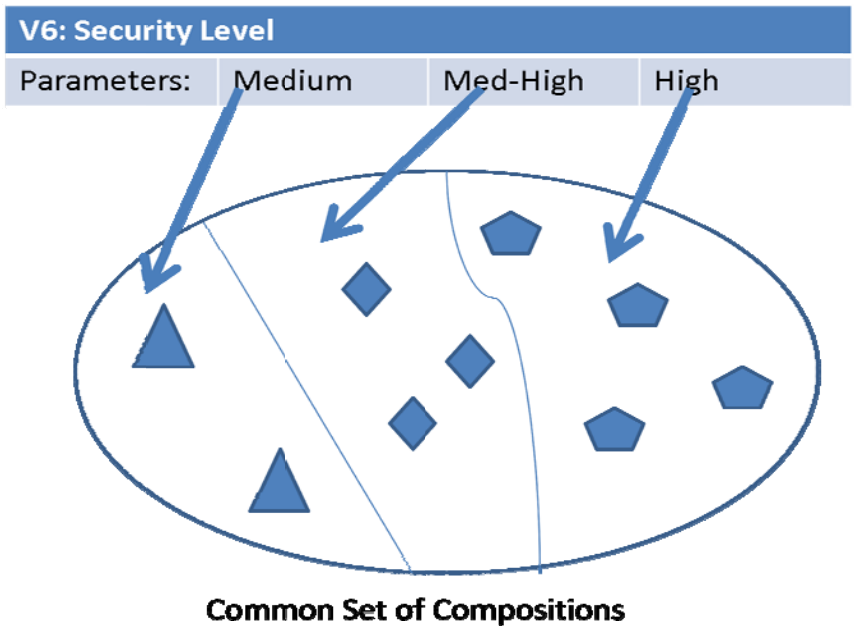

Figure 18. Indexing between parameters of variations and subsets of the commonality composition subset

Figure 18 shows an example of the mapping relation for variability V6 created by this indexing process. The user-selectable parameter of Medium security level maps to the commonality composition subset shown with two triangles, each of which represents a candidate service composition. In order to find these triangle compositions, we first verify the security level to find those compositions with a Medium security level tag in

all their services. Next, according to the dependency graph, a Medium security level requires an encryption length of 128 bits, so we verify whether all these compositions also have 128-bit encryption.

After creating the indexing for the parameters, the construction of the composition search space is completed and the application-engineering phase, i.e., the generation by the user of a service composition that verifiably satisfies his/her customized requirements, begins.

5.2.6 Solving Constraints

The user customizes the service composition's functional and non-functional requirements by setting the parameters of variation. However, since the user cannot be expected to handle the possibly complex dependencies among the variabilities, a specification from the user inputs may not guarantee consistency among the parameters. For example, an inconsistent specification is shown in Table 5. A Low parameter in the communication delay variability V7 constrains the choice of the Dispatch Service to be Speed-line rather than a normal one. In this case, the specification cannot result in a valid service composition.

Instead, in order to ensure the consistency of the user's choices, a dependency graph walking algorithm is applied to interact with and guide the user in solving the constraints. The constraint solving algorithm is as follows:

1. User picks a variability to start the process

2. User decides the parameter for this variability.

3. Locate the node of this variability in the dependency graph.

4. Check and apply the constraints on all the edges of current node. If the constraints force a variation value on any other node(s), mark them as explored.

5. Walk to the next unexplored node in the sub-graph and iterate from step 2.

6. If all nodes of the current sub-graph have been explored, pick a next sub-graph and start from step 2.

7. If all sub-graphs have been explored, then all constraints in the dependency graph have been solved.

Table 5: Excerpt of inconsistent specification

Variability	Value	Constraints
V1	Normal	If V1 is Highly-secured, V6 is High. If V1 is Speed-line, V7 is Low.
V7	Low	If V7 is Low, V1 is Speed-line

Without the product line concept of a dependency graph, the constraints among the different constraints would have to be solved later during the constraints' verification using higher-overhead verification techniques such as the one introduced in section 4.2.5. Solving the constraints using a dependency graph is more efficient than detecting the inconsistency in the later verification phase. Moreover, we will need this consistent

specification to apply the query algorithm to the composition search space in the next step.

5.2.7 Query the Composition Search Space

We now query the composition search space to find the subset of compositions that satisfy each user-selected variation parameter. Because the construction of the search space in the previous step has verified that the remaining compositions satisfy the common requirements and that the user's selection of variabilities is consistent, we need only to perform a simple look-up in the Search Space Table. Table 6 continues our example, showing two queries on the parameters of V1 and V7.

Table 6: Sample query in the composition search space

Variability	Value	Composition Subset
V1	Speed-line	CSet1
V7	Low	CSet2

We note that since (V1=Speed-line) and (V7=Low) have already been shown to be consistent by the constraint solving in section 5.2.6, if any element of CSet1 satisfies V1=Speed-line and any element of CSet2 satisfies V7=Low, then any element of CSet1∧CSet2 satisfies (V1=Speed-line) and (V7=Low). More generally: *For any two constraints P1 and P2, if P1 and P2 are consistent or independent, and any composition*

in SetA satisfies P1 and any composition in SetB satisfies P2, then any composition in SetA ∧ SetB satisfies both P1 and P2.

5.2.8 User Customization

The user can customize the web service functionalities and its NFRs by setting the values for the variabilities. However, we need to make sure these values of the parameters are consistent with each other and do not violate any constraint identified in the CVA. To do this, we use a decision model. The structure of the decision model is shown in Figure 19. A decision model [65] is a user-friendly, front-end model that consists of the commonalties, the variabilities, the parameters of variation together with their constraints/dependencies, the mapping relations and the algorithms to handle these complex relations. In both the product line and web-service context it interacts with the user and generates the user preferred product.

To customize the service composition, using prompts from the constraint solving algorithm in Sect. 5.2.6, the user successively selects values for the parameters of variation until all variabilities have been decided. The workflow for making decisions using the decision model is:

1. Apply constraint solving algorithm to interact with the user.
2. Report failure if no valid consistent specification exists with the current user decisions and start over.
3. If a consistent value set of the variabilities is obtained, query the search space with these parameters to retrieve a set of composition subsets.

4. Compute the conjunction of the composition subsets. If the conjunction yields a non-empty set, *any composition in this set will satisfy the user's common and variable (customized) FRs and NFRs*. If the conjunction is an empty set, it means no composition satisfies all the user's requirements.

5. Output the result. In the case of a non-empty set, we use the composition with the least services to avoid redundant services.

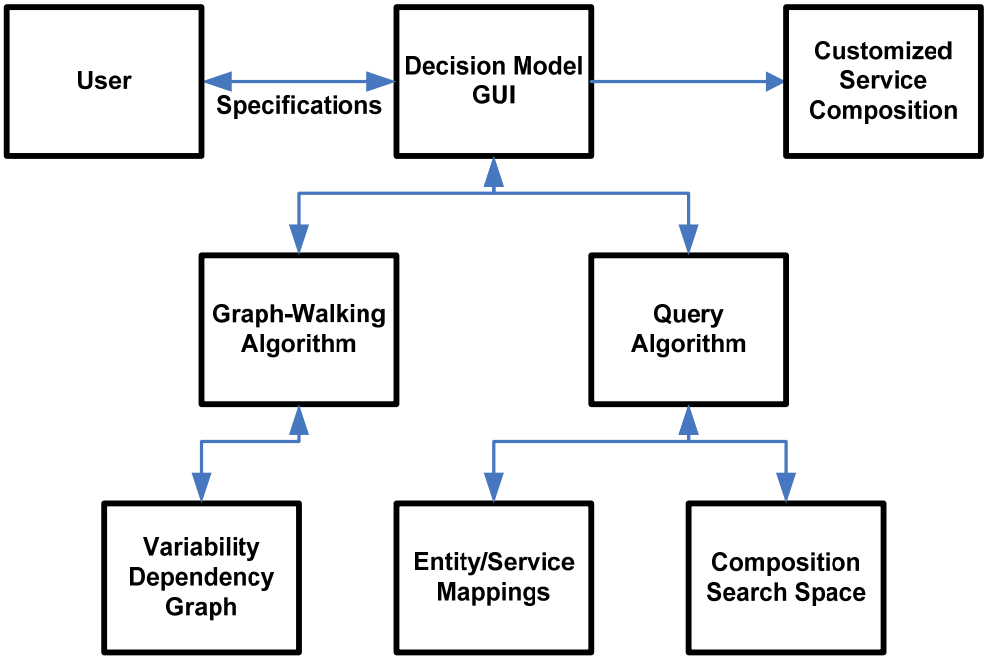

Figure 19. Structure of the decision model

5.3 Complexity of SPL-WSC

In this section we compare the worst-case complexity of the software product-line approach described above with that of the traditional, non-product-line approach in [45].

For a web service composition system, there are three cases in which we need to re-verify the NFRs for a web service composition: when there is an update to (1) a global QoS constraint, (2) a modular service QoS profile, or (3) an aggregation rule of a QoS attribute., Any of these three cases can cause a violation of QoS constraints.

Our assumption in analyzing the complexity of the SPL solution is based on the following observation regarding update frequency:

Frequency of QoS constraint change >> Frequency of single modular service QoS change>> Frequency of QoS aggregation rule change.

As the global QoS constraint change is far more frequent than the other two cases, this is the only situation we have considered in complexity comparisons.

5.3.1 Complexity of the SPL solution process

We divide the SPL approach process into two stages. The first stage, the preparation stage (akin to the domain engineering process for a software product line), conducts commonality and variability analysis (CVA), functionally composes all web service compositions and verifies every possible variability value in the CVA table in order to construct the indexed search space of web service compositions. The second stage, the post-deployment stage (akin to the application engineering process for a software product line), takes a to-be-verified QoS constraint as input, searches the CVA table for it, and returns a set of web service composition candidates. A further verification on this set of candidates may be required to determine the valid web service composition. Whether to conduct this verification on the candidate service composition set is

dependent on the type of the QoS attributes. We divide all QoS attributes into two categories:

(1) The QoS attributes whose value can be relaxed.

An example for a QoS attribute that can be relaxed is the message delay. If a service composition satisfies a constraint that requires the message delay to be less than 10 seconds, it must satisfy all constraints that require the message delay to be less than k seconds where k>10. This is often the case in setting the values for a variability in a CVA table. If the values of a QoS variability can be relaxed, the QoS constraint can map to many values of this variability. Thus, the CVA table needs to retrieve multiple sets of web service composition candidates with one relaxable QoS constraint as input.

(2) The QoS attributes whose value cannot be relaxed.

An example for this type of QoS attributes is the authentication requirement. If a web service composition satisfies the authentication requirement, it cannot also satisfy the anonymous requirement (without authentication). With the authentication requirement as input, the search result from the CVA table is a variability value and its corresponding service composition candidate set.

Preparation Stage

As Figure 20 shows, there are three steps in the preparation stage. The first is manual, while the second and third affect the complexity estimation. The explanation of these complexities follows the legends of the figure.

1. CVA
2. Functional composition of modular services (Composition Search Space Construction)
3. Verify and index each variability value (QoS Search Space).

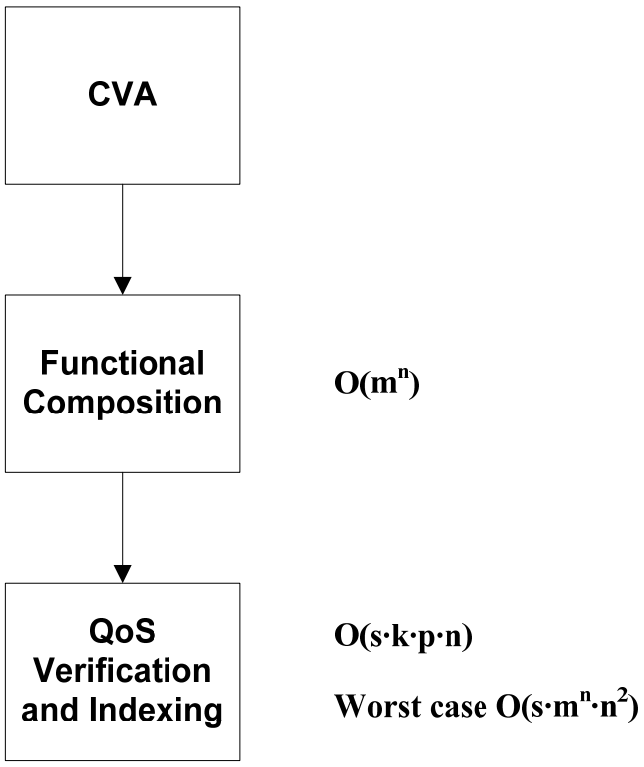

Figure 20. Complexity of the preparation stage of the SPL approach

Let n: the number of the functional operations in the goal model.

m: the number of candidate modular services for each functional operation.

s: number of variabilities in the CVA table.

v: average number of parameters for each variability in the CVA table.

p: number of states in the property automata. $p \leqslant n$.

k: number of composition candidates in the composition search space that satisfy the functional goal model.

Step 1. Because the CVA process is conducted manually in the domain engineering phase and requires expert knowledge, we do not include it in the complexity calculation.

Step 2. For the functional WSC, we choose a brute force strategy and try every possible combination of the possible candidate modular services, so the complexity for this process is exponential as $O(m^n)$. Since we take k as the number of composition candidates derived from this functional composition process, $k \leqslant m^n$. Thus, if every possible combination of modular services satisfies the goal model, $k = m^n$. Otherwise, $k < m^n$.

Step 3. During the QoS verification and indexing process, each candidate composition is verified against all constraints and maps to the right variability parameter in each variability. Since there are k candidate compositions Step 2, and s variabilities to verify, there are k·s QoS verifications that are performed. Each verification calculates the product of the service composition automaton and the property automaton. As a single state represents a modular service in the automaton, the number of states in a candidate service composition is the same as the number of states in the goal model, i.e., n. Since p is the average number of states in the property automata, each QoS automaton verification is $O(n \cdot p)$. Thus, the total verification complexity of this process is: $k \cdot s \cdot O(n \cdot p) = O(k \cdot s \cdot n \cdot p)$. Because we construct our QoS constraint from the goal model, we know $p \leqslant n$. Since $k \leqslant m^n$, the worst case complexity is $O(s \cdot m^n n^2)$.

Post-Deployment Stage:

The procedure to verify a new QoS constraint involves three steps (Figure 21):

1. Check QoS failure of current WSC on this update. When the current web service composition still satisfies the updated QoS constraint, we report success and exit the process. Otherwise we report a failure and continue the process.
2. Search CVA table for the results set of web service compositions.
3. If additional verification of the results set is necessary, verify the results set and update the QoS search space with the new mapping.

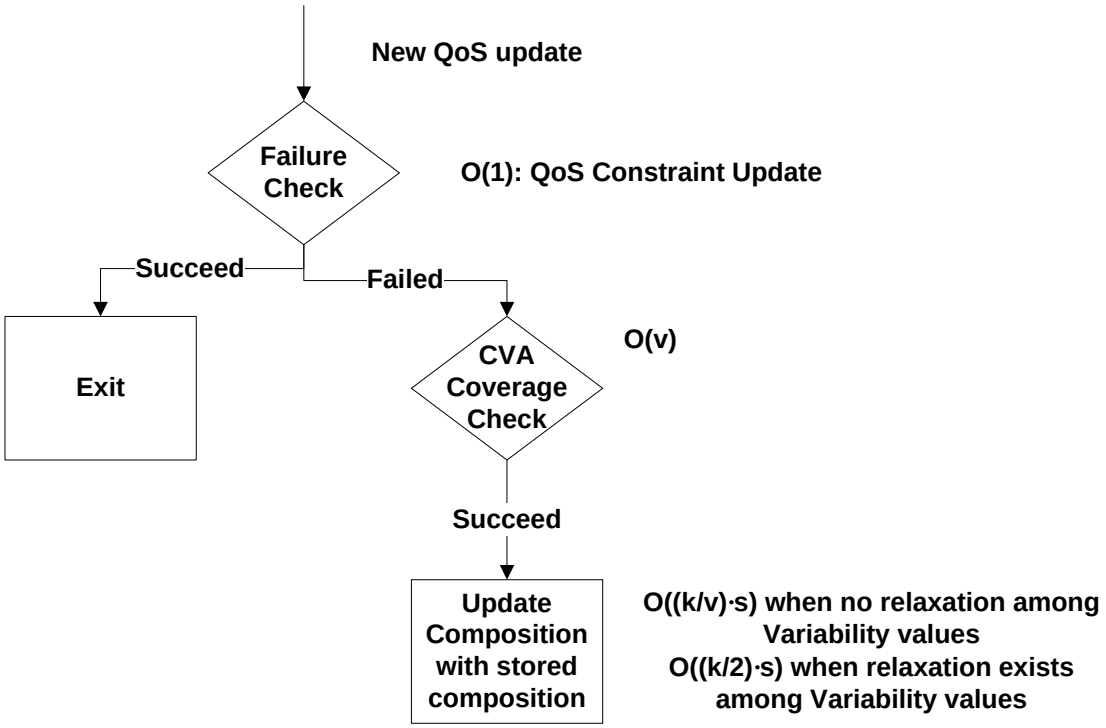

Figure 21. Process to verify a new QoS constraint with QoS-indexed composition search space.

The failure check when QoS constraint is updated costs O(1) time because we only need to check the stored and aggregated QoS property with the new QoS constraint. (Hash-table lookup costs O(1)).

The complexity of the CVA coverage check is O(v) because we need to check which parameter in the variability's parameter set the new QoS constraint belongs to.

Then we get a set of web service compositions that maps to the variability parameter. If this variability parameter is a ranged value, further verification sometimes is required to determine the subset of this result set that satisfies the QoS constraint. Since we need to compare the worst complexity, we here assume this further verification is required all the time.

The complexity of further verification of the results set returned by CVA table lookup is $O((k/v)\cdot s)$, because there are k/v service compositions for each variability value on average and there are s variabilities to verify. (All QoS constraints may need to be re-verified when one is updated in order to maintain consistency.)

However, when relaxation is possible for variability values (for example, a low latency is always valid for a high latency constraint), the average number of possible service compositions in the results set is k/2. Thus, the complexity becomes $O((k/2)\cdot s)$.

We assume that w-percent of QoS attributes can be relaxed and (1-w) percent cannot be relaxed, $0 \leq w \leq 1$.

The complexity of further verifying the results set returned by CVA lookup is:

$$w \cdot O((k/2) \cdot s)+(1-w) \cdot O((k/v) \cdot s)$$

In the rare case of a QoS aggregation rule change, we have to update the QoS search space by re-verifying and re-indexing all related mappings. This case is not considered in our complexity comparison because it rarely happens.

5.3.2 Complexity Comparison

The functional composition approach described in [45] with consideration of QoS constraints tries every possible service composition candidate in the search space. This process has similar complexity analysis as the brute force method used to index all QoS variability values. Its complexity with the verification of QoS constraints is thus $O(s \cdot k \cdot p \cdot n)$, and in the worst case, it is $O(s \cdot m^n n^2)$.

In the previous sections, we have explained that $k \leq m^n$. Because we need to compare the worst complexity in both cases, we need to reduce the number of variables by relaxing some variables to their worst cases. We know the number of functional composed web services is less than or equal to all possible compositions.

We use the goal model automaton as a template to derive the property constraint automaton and use a single state to represent the modular service in the final composition. Thus, the number of states in the property automaton is less than or equal to the QoS constraint automaton. Thus, $p \leq n$.

Considering the worst cases, we can derive the following complexity equations. Note that the complexity of checking QoS failure is equal in both solutions, so is not considered in the following analysis.

Complexity of the normal solution for one QoS constraint failure:

$$O(s \cdot m^n n^2)$$

Complexity of the normal solution for x QoS constraint failures:

$$O(x \cdot s \cdot m^n n^2)$$

SPL solution for QoS constraint update:

Complexity of the preparation stage:

$$O(m^n) + O(s \cdot m^n n^2) = O(s \cdot m^n n^2)$$

Complexity of the post-deployment stage for one QoS constraint failure:

$$O(v) + w \cdot O((k/2) \cdot s) + (1-w) \cdot O((k/v) \cdot s)$$

Overall for one QoS constraint failure :

$$O(s \cdot m^n n^2) + O(v) + w \cdot O((k/2) \cdot s) + (1-w) \cdot O((k/v) \cdot s)$$

Overall for x QoS constraint failures (only counting the preparation stage complexity once):

$$O(s \cdot m^n n^2) + x \cdot O(v) + x \cdot w \cdot O((k/2) \cdot s) + x \cdot (1-w) \cdot O((k/v) \cdot s)$$

For these to be an advantage in using the SPL solution, it must be the case that:

$$O(s \cdot m^n n^2) + x \cdot O(v) + x \cdot w \cdot O((k/2) \cdot s) + x \cdot (1-w) \cdot O((k/v) \cdot s)$$

$$< O(x \cdot s \cdot m^n n^2)$$

By applying $k=m^n$ as the worst case and solving this inequality, we get:

$$x > \frac{sn^2m^n}{sn^2m^n - s(\frac{w}{2} + \frac{1-w}{v})m^n - v}$$

Thus, x (number of times of reuse) must be over this threshold to ensure the SPL solution has an advantage over the normal solution. We name this minimum x to have this advantage as x.lim. The SPL solution has an advantage only when x.lim is positive.

$$x.lim = \frac{sn^2m^n}{sn^2m^n - s(\frac{w}{2} + \frac{1-w}{v})m^n - v}$$

n: number of functional operations in the goal model.

m: number of candidate modular services for each functional operation.

s: the number of variabilities in the CVA table.

v: the number of values for each variability in the CVA table.

w: $0 \leq w \leq 1$. A constant.

In the next section, we design and conduct experiments to analyze how the threshold x.lim varies accordingly to these four variables: n, m, s and v.

5.3.3 Investigation on Complexity Model

5.3.3.1 Investigation 1:

When the goal model and the modular service registry are fixed, n and m are determined. In different projects with different CVA tables, s and v are variable. Our target is to explore the effect of the number of variabilities, and the number of alternatives per variability, on the performance of the SPL-WSC approach.

Design of Investigation 1: With fixed n and m, we compare the x.lim balance point when s and v change. Let s= [1 … 10] and v= [1 … 200], then what is the value of x.lim?

Figures 22 to 29 show the results of Investigation 1 with different n, m and w settings.

Figure 22 shows how x.lim varies with s and v when n=2, m=3 and w=0. There are five shark-fin-like objects in the mesh. For each line when s is equal to 1 to 5 at x axis, the curve along the y axis is like a heart-pulse curve. It means when s is fixed, there is always an advantage in adopting the SPL approach with a number of reuses when the number of values in each variability v increases until the heart pulse. When v is close to the heart pulse, the number of required reuses for SPL to have an advantage increases steeply to a couple hundred times. When v increases more, this reuse number immediately becomes negative and remains negative. Note that the light orange zone after the shark-fins is the negative zone. This phenomenon occurs because when n and m are small, there are few composition candidates to verify. When the number of variability

values v increases to a very large number, the overhead of searching a variability value in the CVA table will also increase dramatically. At this point, the total cost of the SPL approach for each QoS verification will surpass the cost of the normal approach.

Figure 23 and Figure 24 show how x.lim varies with s and v when n=2, m=3, w=0.5 and w=1. As expected, when the percentage of relaxable QoS attributes increases, the shark-fin appears sooner on the y axis, which means the balance point appears at a smaller v number. Basically, the parameter w doesn't affect the layout of the relationships of the variables much. Thus, from this point on, we show all further figures only at w=50.

Figure 25 and Figure 26 show how x.lim varies with s and v when n=2, m=4 and n=2, m=5. We see that an increase in the number of candidate modular services for each goal model operation m can postpone the shark-fin effect, because an increase in m can increase the size of the candidate service composition search space, which can make the reuse in SPL approach more significant.

Figure 27 shows how x.lim varies with s and v when n=3, m=3. An increase in the number of service operations in the goal model n can also postpone the shark-fin effect, and its effect on postponing the shark-fin effect is more significant than variable m. This is because the increase of n reduces the advantage of verifying a simple service composition automaton in the normal approach.

Figure 28 and Figure 29 show how x.lim varies with s and v when n=3, m=4 and n=5, m=10. As expected, with a reasonable size of the goal model where the service

operations are more than 3, and a reasonable number of candidate modular services where the candidate modular services for each service operation are more than 4, the advantage of the SPL approach is obvious and significant.

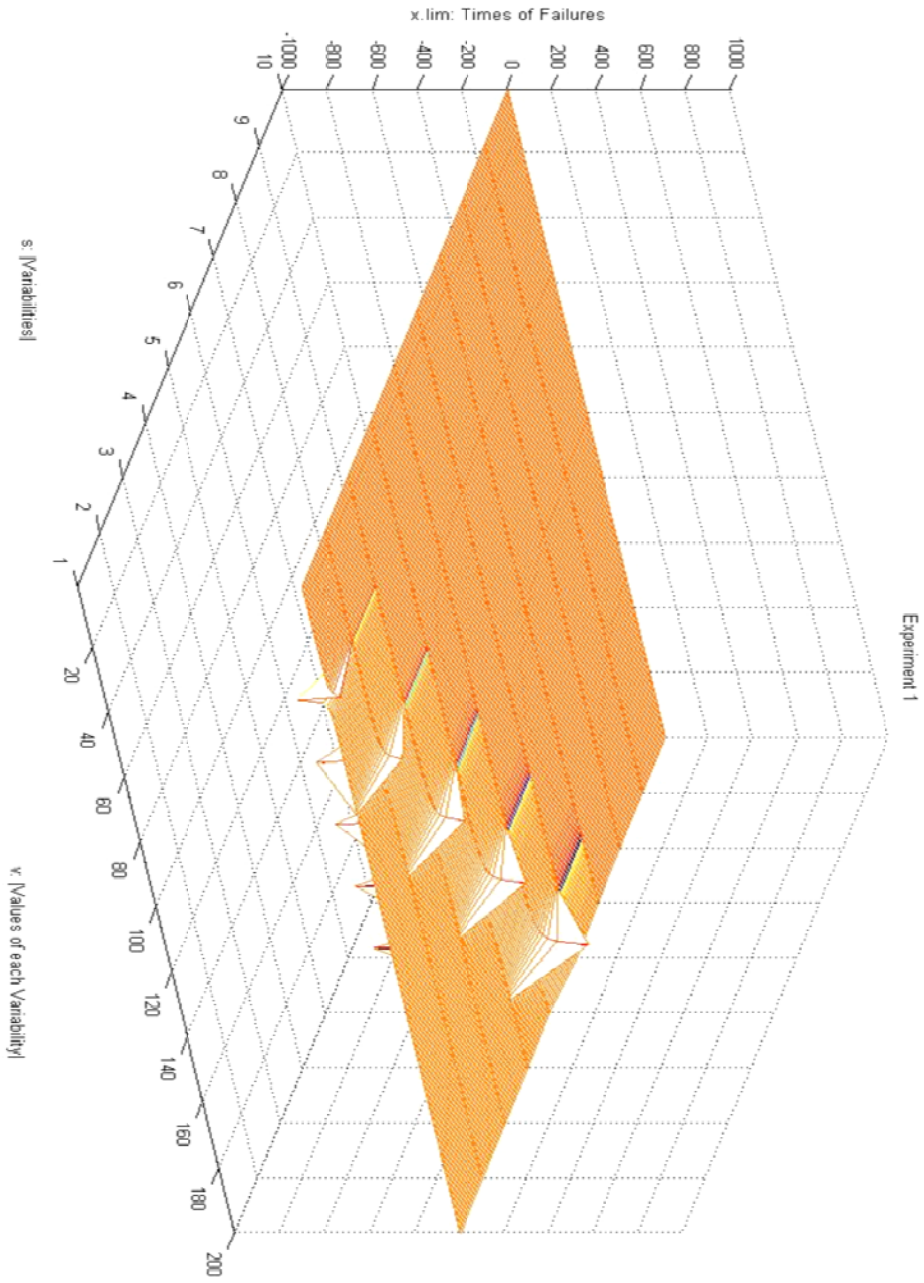

Figure 22. Investigation 1: n=2, m=3, w=0.

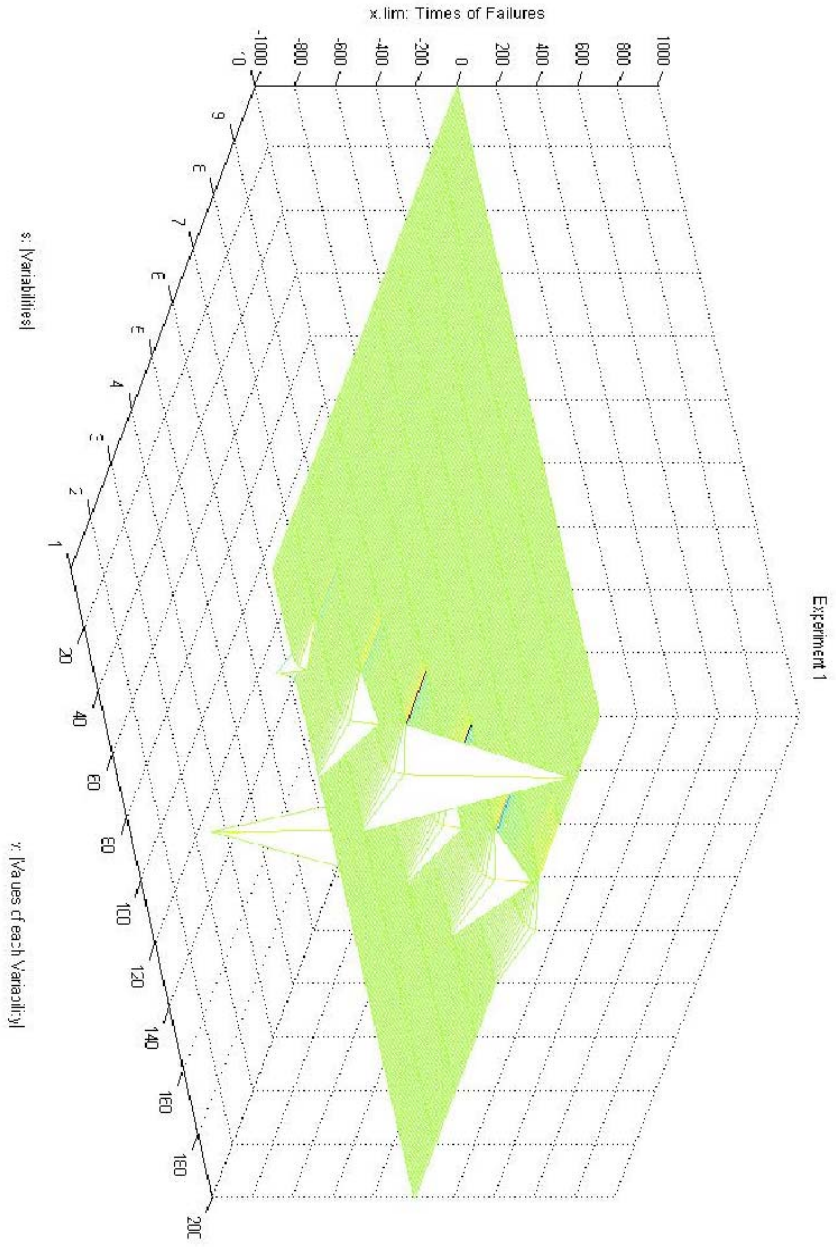

Figure 23. Investigation 1: n=2, m=3, w=0.5.

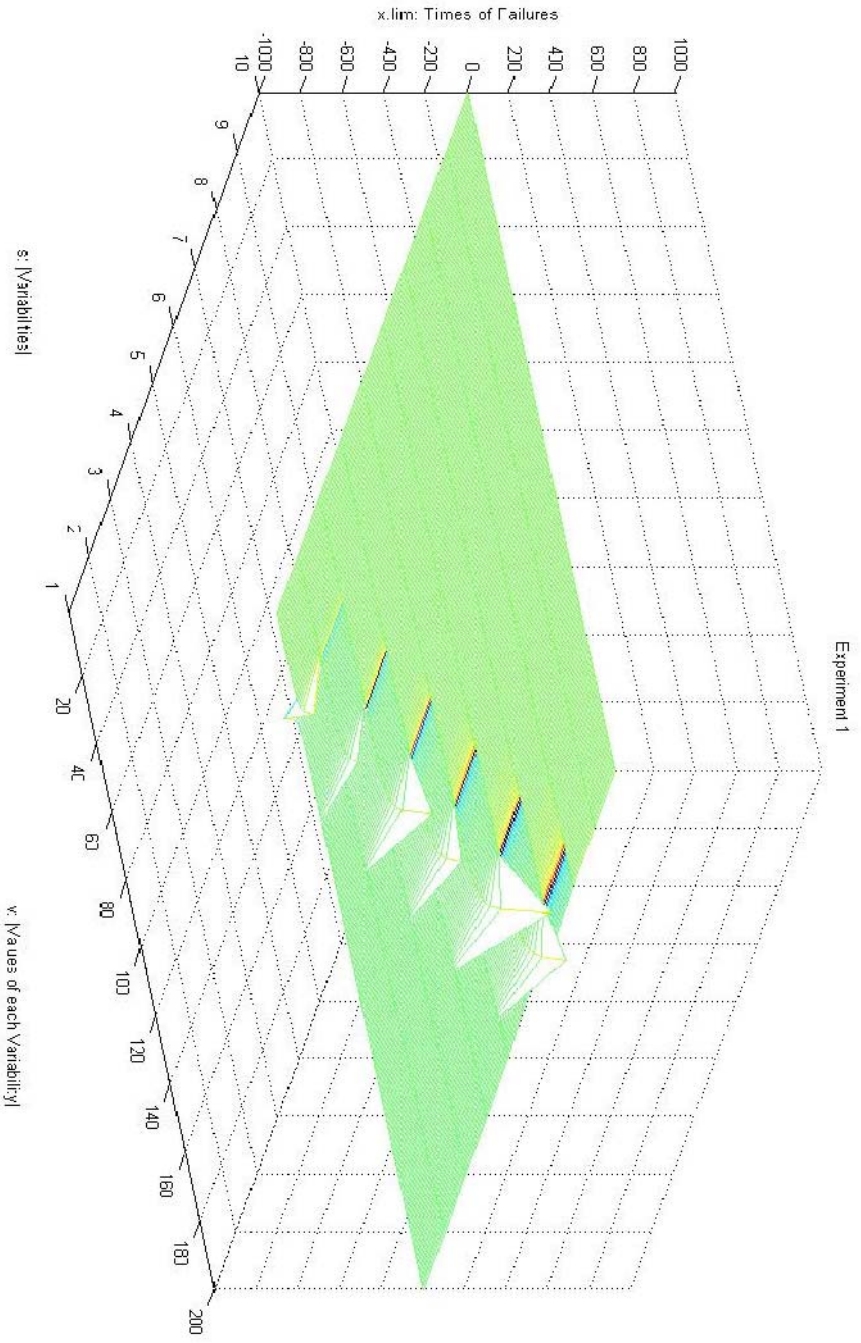

Figure 24. Investigation 1: n=2, m=3, w=1.

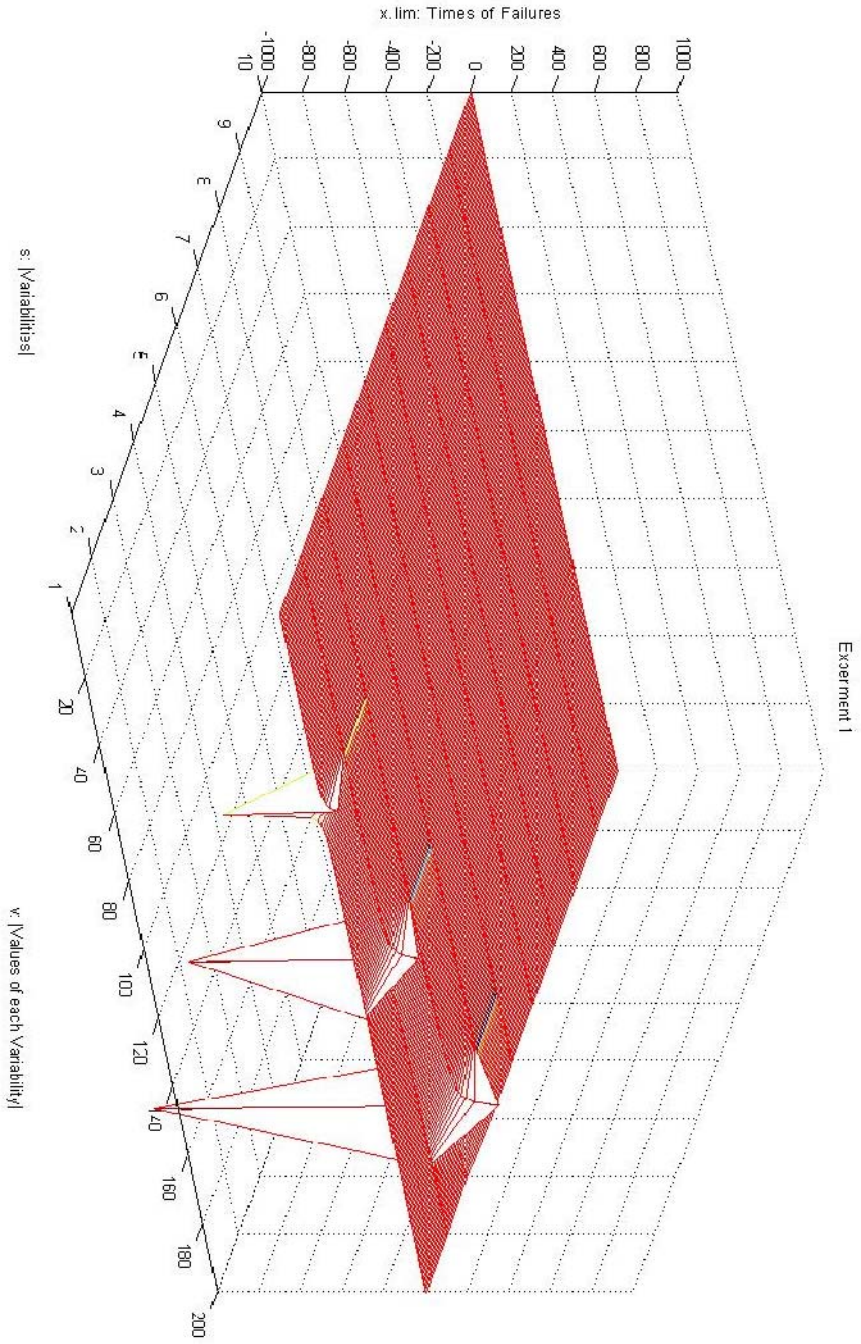

Figure 25. Investigation 1: n=2, m=4, w=50.

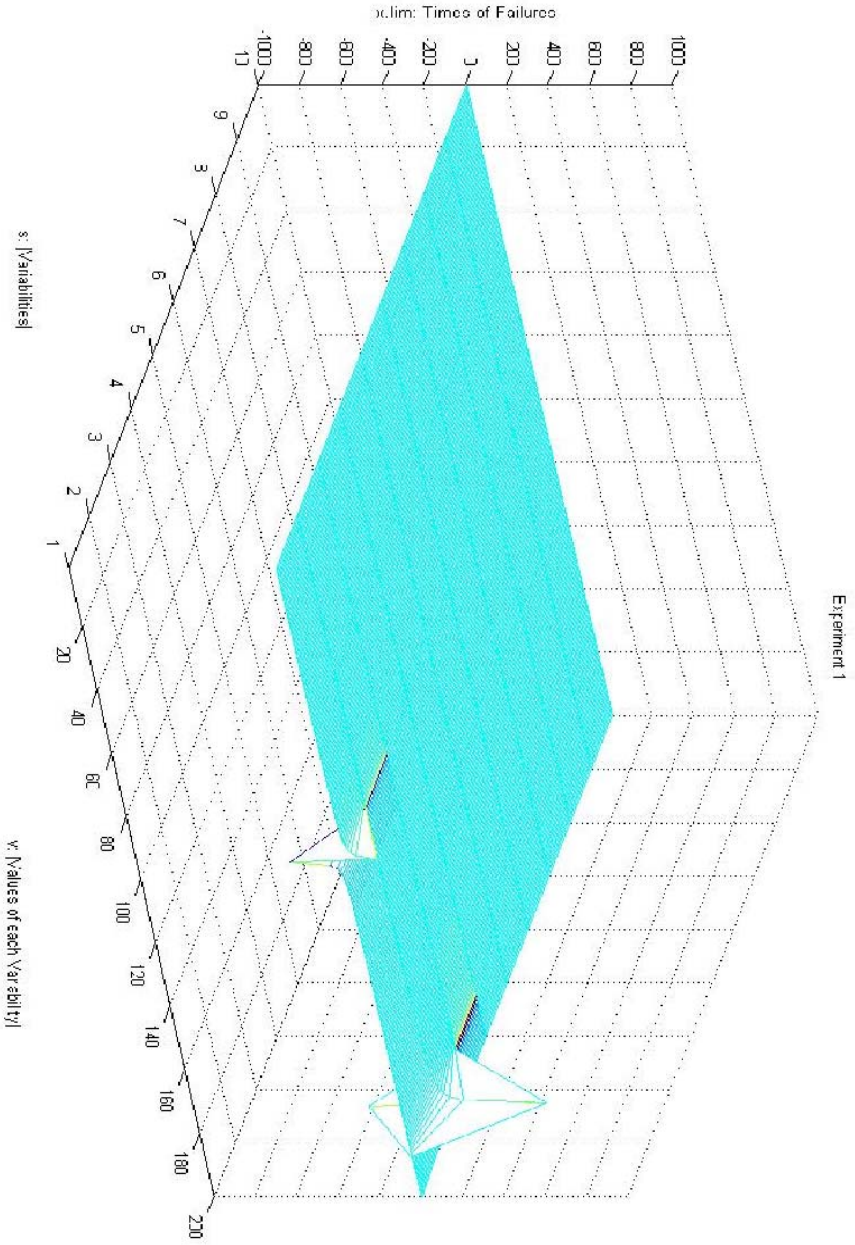

Figure 26. Investigation 1: n=2, m=5, w=50.

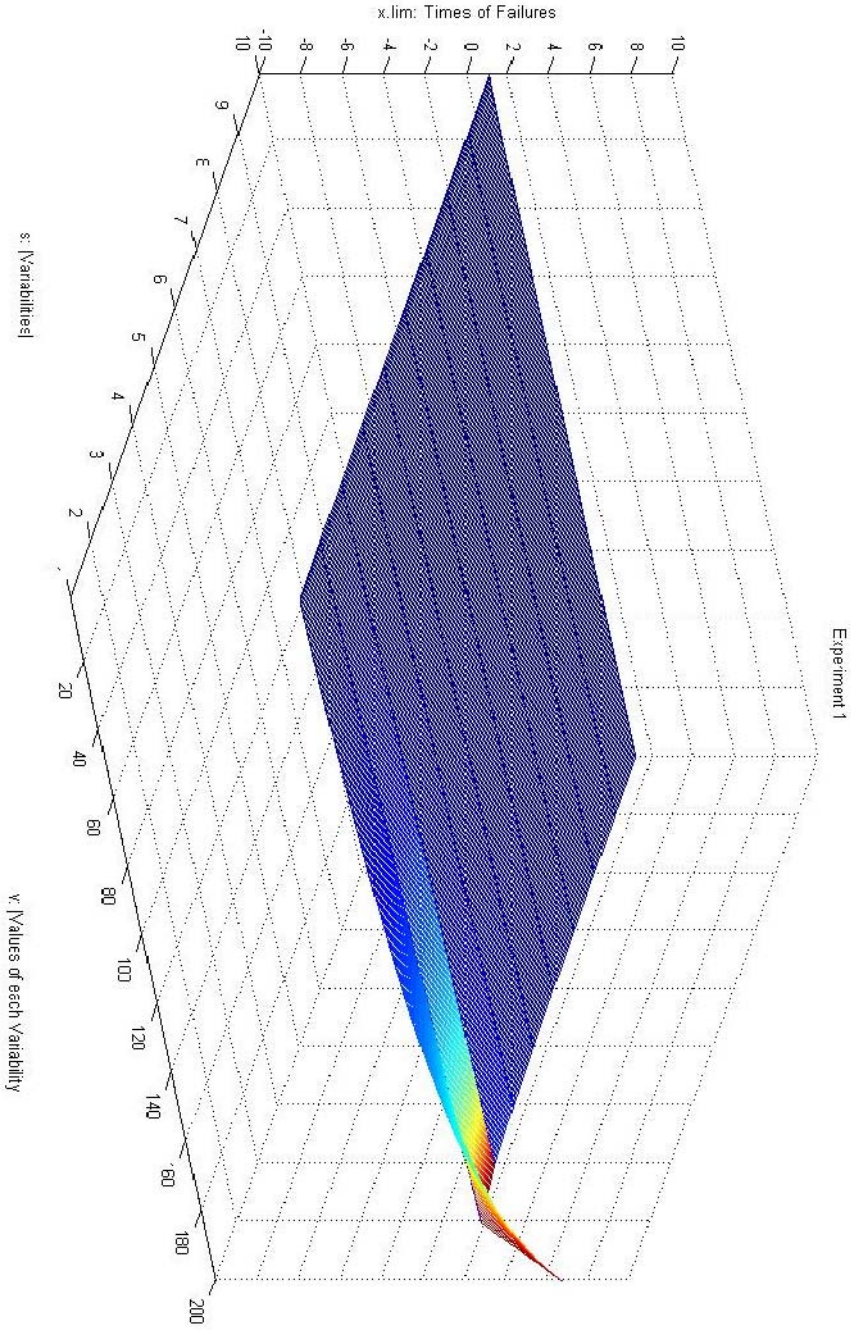

Figure 27. Investigation 1: n=3, m=3, w=50.

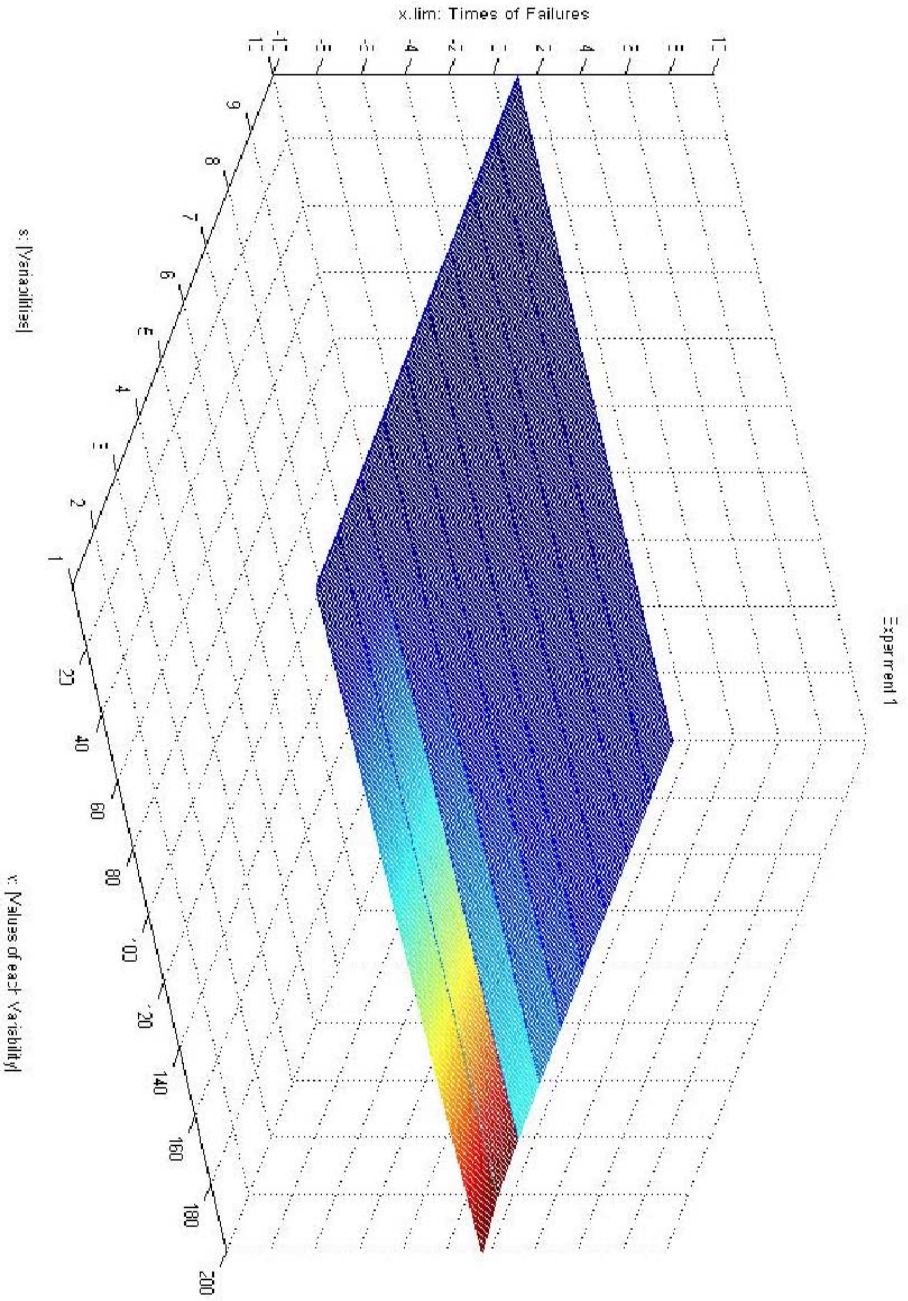

Figure 28. Investigation 1: n=3, m=4, w=50.

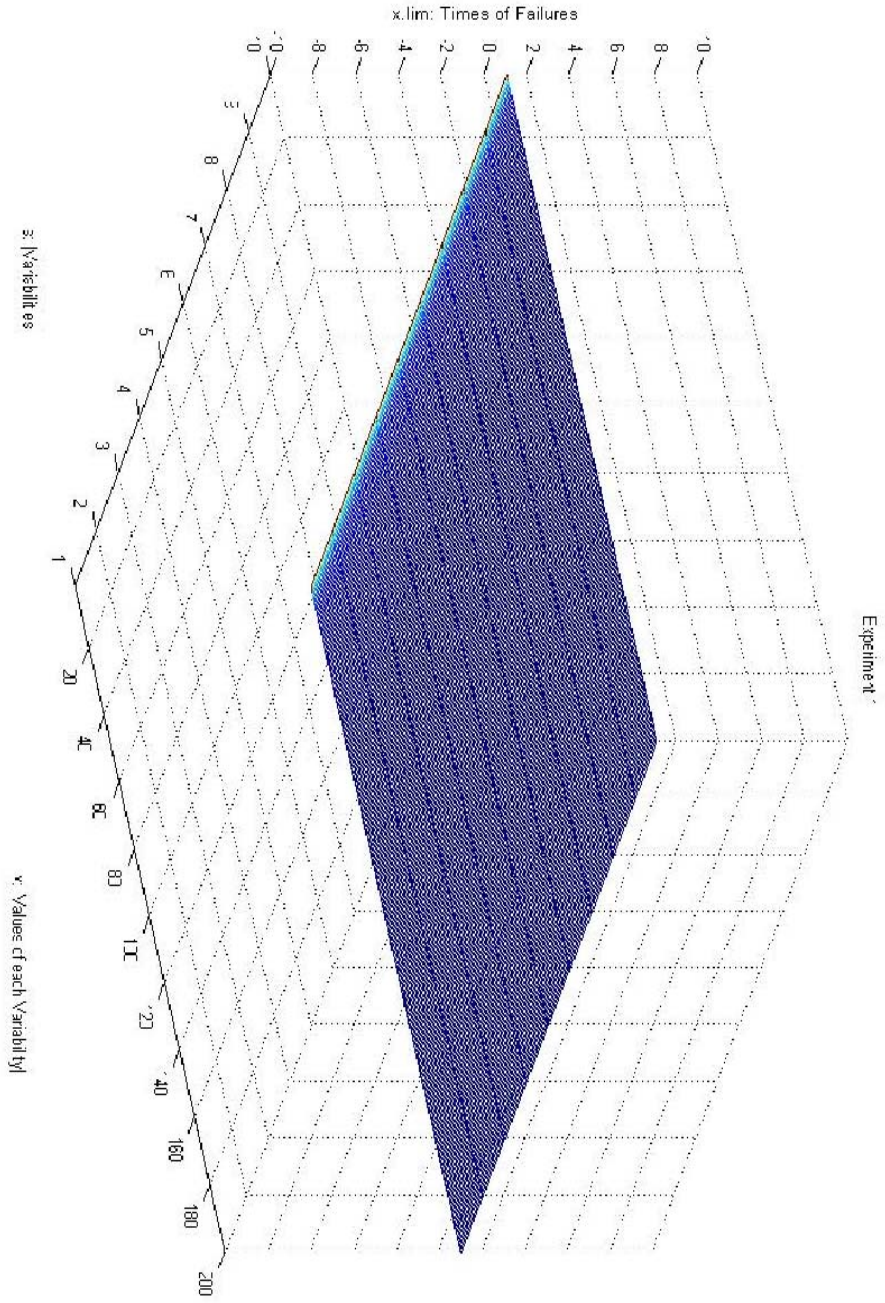

Figure 29. Investigation 1: n=5, m=10, w=50.

5.3.3.2 Investigation 2:

When the CVA is designed, s and v are determined and fixed. The size of the goal model can vary because there are different ways of implementations. One of the service registries can be chosen based on its size. Our target is to explore the effect of the size of the goal model and the size of the service registry, on the performance of the SPL-WSC approach.

Design of Investigation 2: When s, v and w are fixed, we compare the x.lim balance point when n and m changes. When n=[2 .. 10], m=[2 .. 50], what is the value of x.lim?

Figure 30 to Figure 34 show the results of Investigation 2.

Figure 30 to Figure 32 show how x.lim varies with n and m when s=5, v=5 and w= [0, 0.5, 1]. Shown in these three figures, except when there is a corner with shark-fin effect when n and m are small, the SPL approach has an advantage over the normal approach with a small number of reuses. With the increase of the w constant, this shark-fin effect is more obvious, which also confirms the conclusion reached in Investigation 1.

Figure 33 and Figure 34 show how x.lim varies with n and m when s=5, v=10, w=0.5 and s=10, v=20, w=0.5. The surface layout does not have any significant difference compared to those the previous three figures. It shows parameters n and m have greater impact on the SPL approach performance than s and v.

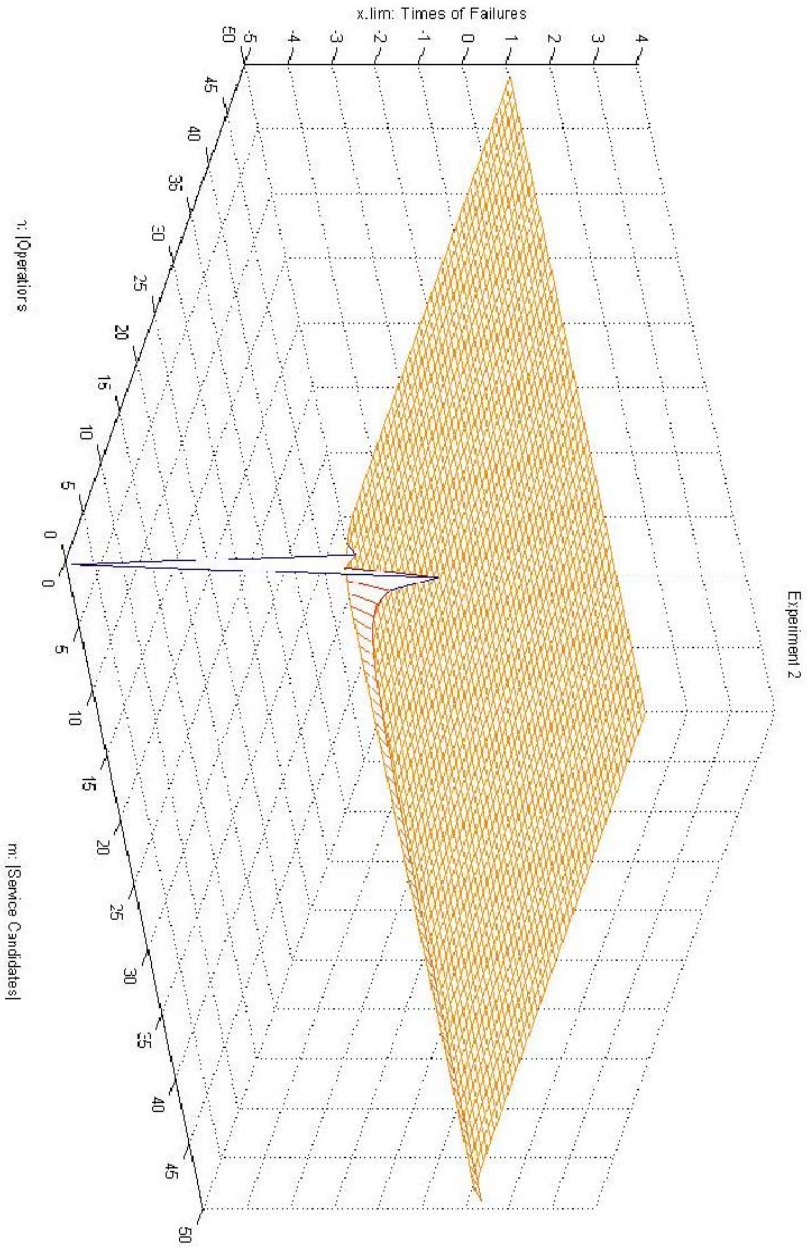

Figure 30. Investigation 2: s=5, v=5, w=0.

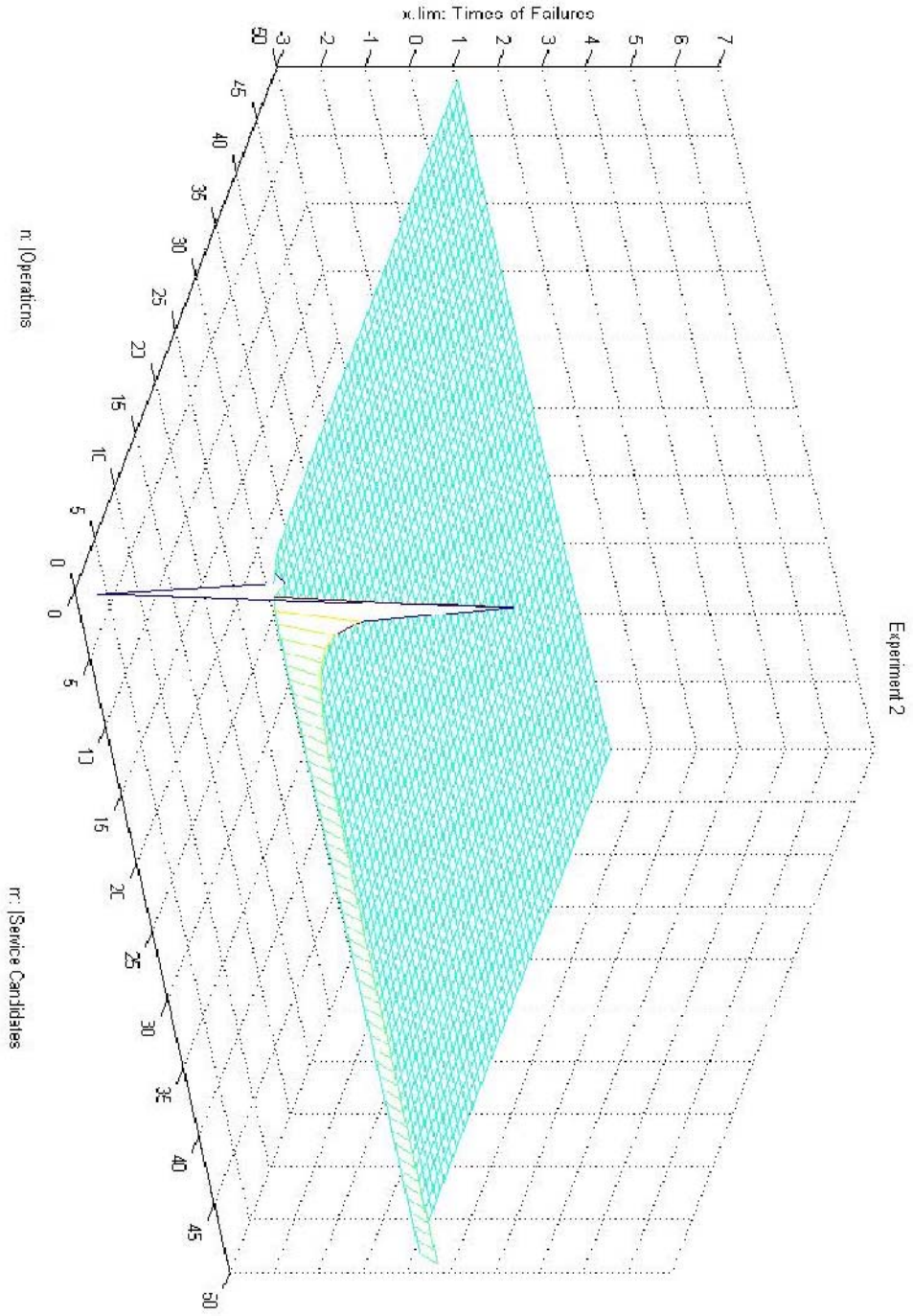

Figure 31. Investigation 2: s=5, v=5, w=0.5.

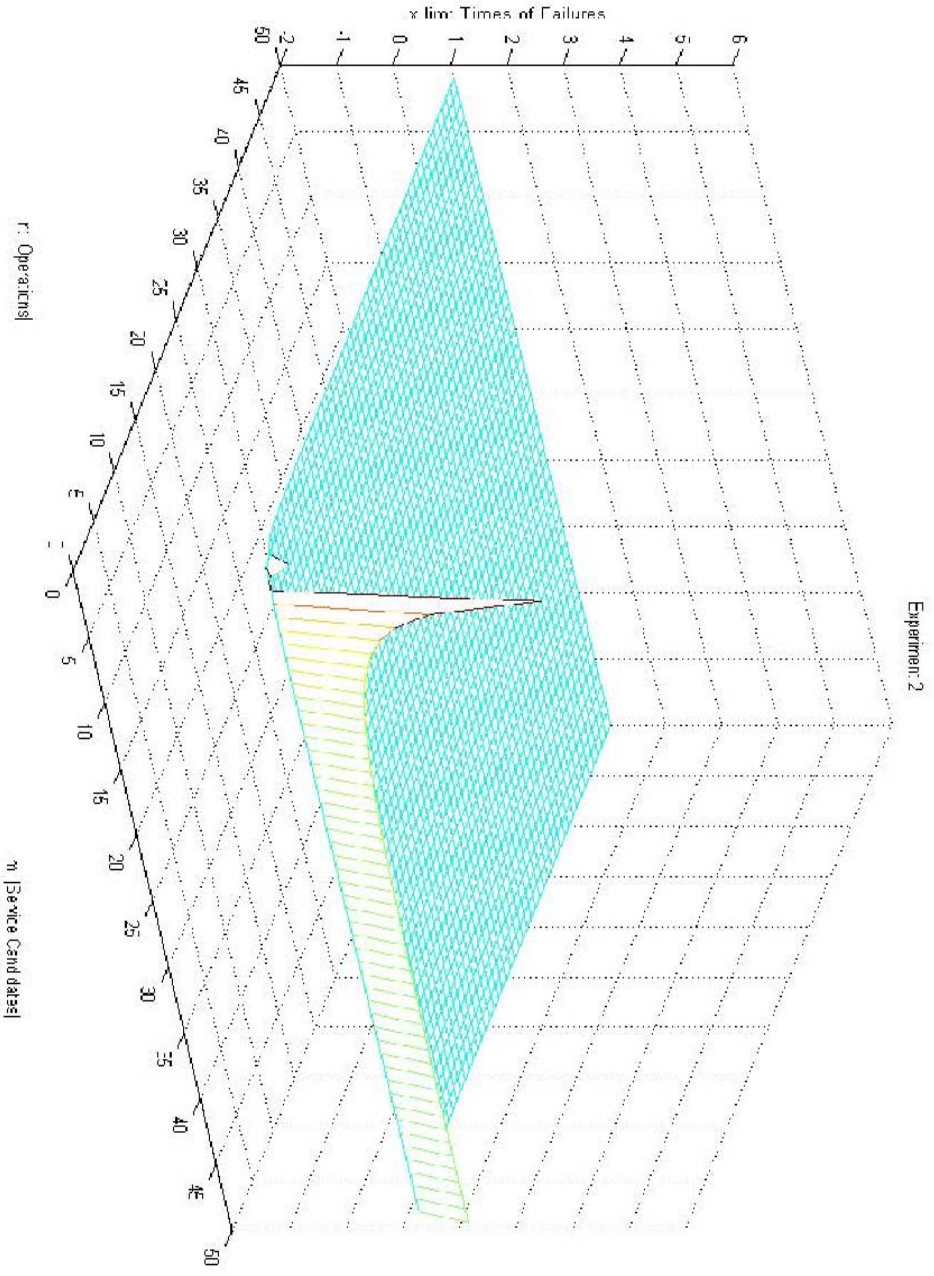

Figure 32. Investigation 2: s=5, v=5, w=1.

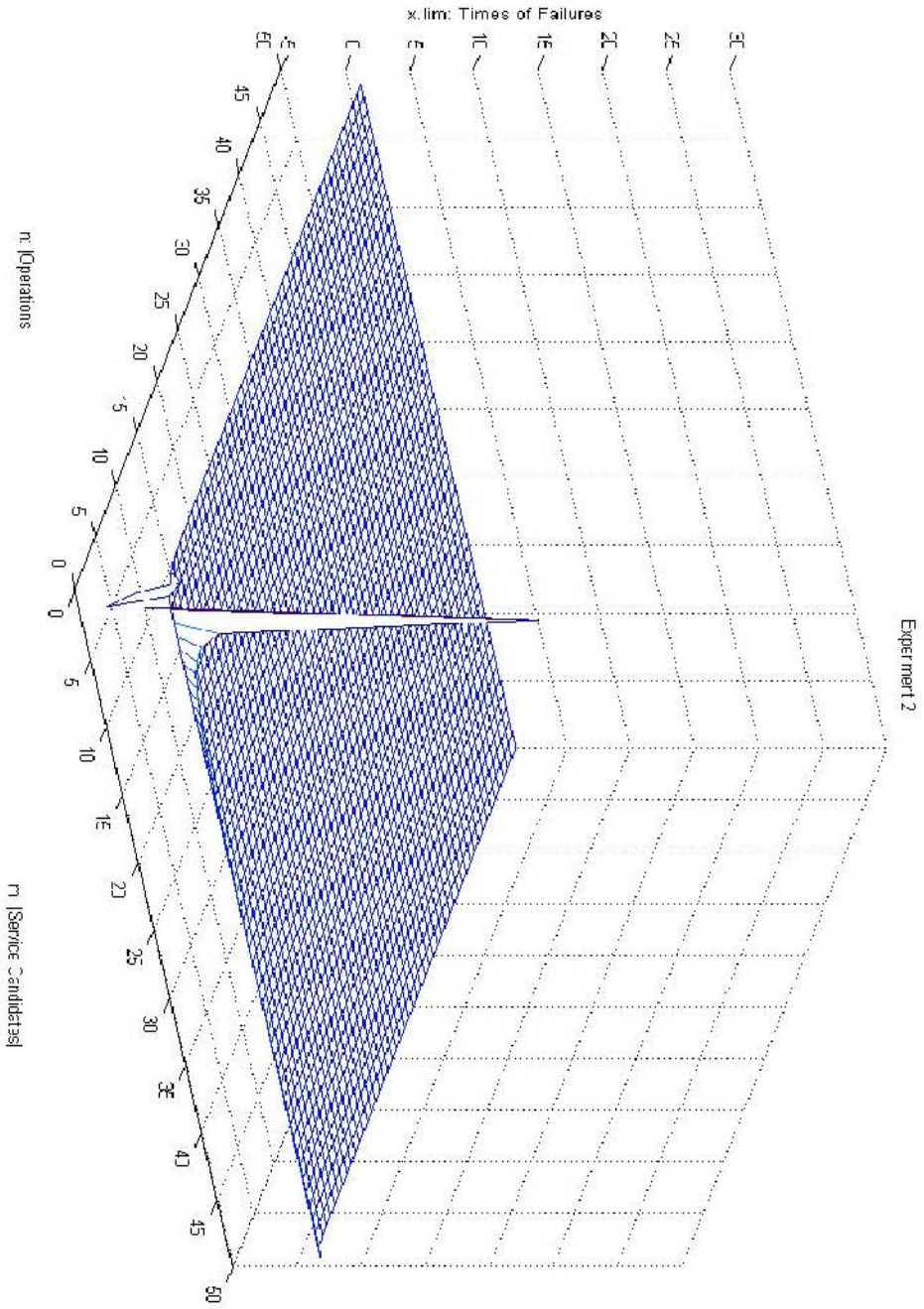

Figure 33. Investigation 2: s=5, v=10, w=0.5.

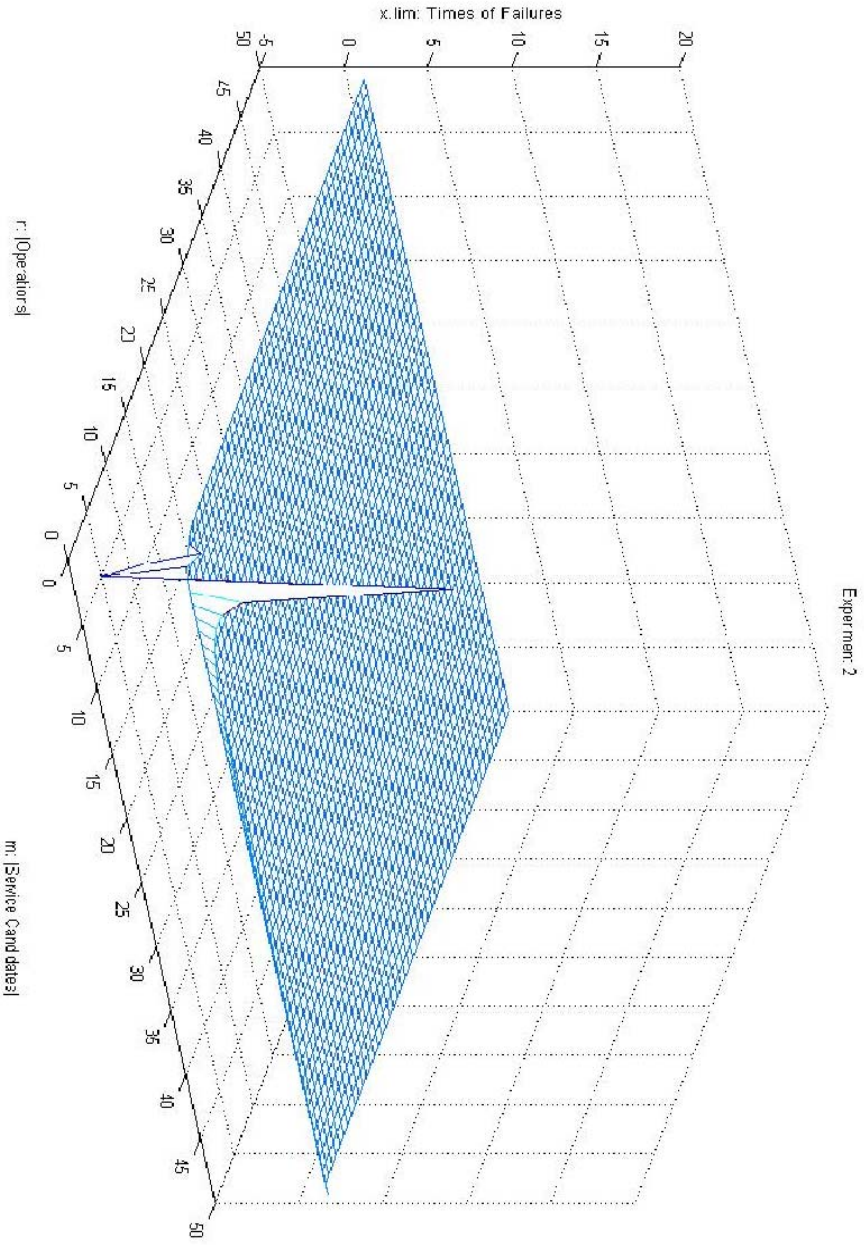

Figure 34. Investigation 2: s=10, v=20, w=0.5.

5.3.4 Conclusion of the Investigations

From Investigation 1 and Investigation 2, we can identify the impacts of parameters in the SPL approach on its advantages over the normal approach. The increase of the size of candidate compositions search space by either n or m can increase the reuse advantage of applying our SPL approach. Parameter s (number of QoS variabilities) has a positive effect on the SPL approach, while parameter v (size of variability parameters set) has a negative effect on the SPL approach when it is too large.

Overall, the SPL approach has more advantages and a comparatively better efficiency when there is a complex service composition (larger goal model size) with many candidate modular services. The SPL approach is more efficient when there are more variabilities with reasonable numbers (>3) of parameters in each variability in the CVA table.

5.4 Conclusion

This chapter introduces a new approach to customizing functional and non-functional requirements of a web service composition by incorporating software product line engineering techniques into the web service domain. By following the product line engineering procedures, web service compositions in the search space of a commercial service provider can be more readily composed, verified and reused in the presence of users' personal selections of their preferred variations.

This approach creates a two-phase solution for efficiently handling mass-user service customization: a preparation phase in which the composition search space is constructed, and an implementation phase in which the web service composition is

customized. As long as functional requirements remain the same during system evolution, the preparation stage does not need to be repeated. We anticipate that the preparation phase will occur off-line and the customization phase will occur at runtime. Most of the computational overhead of verification is pulled into the preparation stage. The runtime verification for the user's customization requirements (the selection of variations) is thus simplified. A product-line decision model hides background relations, dependency models and verification algorithms from the user. The decision model interacts with the user to maintain a consistent set of customized requirements and to generate a more convenient and efficient experience of service customization.

CHAPTER 6. SIMULATION

This chapter introduces a simulation platform for verifying web service compositions. The purpose of constructing this platform is to evaluate the practicality and the merits of our approaches described in the previous two chapters. Through simulations of QoS property verifications for web service compositions on a randomly generated large data set and under different environmental settings, analyses and comparisons based on simulation results show the advantages and disadvantages of each of our approaches under various circumstances and configurations. These analyses guide us in choosing and configuring our approach process to achieve the best efficiency of web service composition verifications for a service oriented system.

6.1 WSCSimu Platform

The simulation platform for testing web service composition verification approaches is named 'WSCSimu'. To illustrate the design of WSCSimu platform, Figure 35 shows the class diagram excerpt of the WSCSimu project.

We design an **Attribute** class to define QoS properties that can be represented in our system, which consists of their variable types, boundaries and aggregation rules.

The **aggregation rules** can also be redefined in its children classes by taking the advantage of the polymorphism feature of Java. Sample aggregation rules here are the

lowest rules, which always choose the lowest value encountered while aggregating, and the additive rule, which calculates the sum of all encountered property values. More aggregation rules can be defined according to the nature of QoS properties.

The **AttributeValues** object records one or more pairs of attributes and their values.

The **AttributeGenerator** generates different types of attributes and their values randomly for simulation purpose.

The **Constraints** object randomly generates QoS constraints with scoping information and quantitative QoS constraints. When generating QoS constraints, we have considered the size of the scope, so QoS constraints are always appropriate for the scope.

The **Automaton** object consists of a set of the **State** objects and a set of the **Transition** objects which together form the standard automaton. Each of the transitions affixes with a service **Operation** object according to our SOA setting.

The **CompositionGenerator** randomly generates an automaton as the web service goal model. We realize this random generation by specifying three parameters. The first parameter is the number of automaton tiers, i.e., the maximum number of transitions from the start state to the end state in any path. The other two parameters are the maximum number of branches for each state and the possibility of branching. For example, if the number of tiers is 6, the maximum number of branches for each state is 3 and the possibility of branching is 30%, which means the CompositionGenerator generates from a start state. For each state generated, the CompositionGenerator has a 30%

possibility to generate at the most 3 outgoing transitions. At tier 6, all states have out-transitions to an end state.

The **Verifier** is the implementation of our core QoS verification algorithm. It implements the process of calculating the product of the web service composition automaton and the QoS constraint automaton and outputs the verification result. When verifying multiple QoS constraints, this verifier can be used to conduct an independent, sequential or incremental verification process. The other two verifiers, the VerifierBigBang and the VerifierTwoStage, are variations of this verifier. They are designed to perform other processes for multiple constraints verification. Rather than the normal Verifier which can only take one constraint as input, both of them can take a set of constraints as input to verify.

The WSCSimu typically conducts a QoS constraint verification on a web service composition as shown in Figure 36. Simulation processes described in the next sections are very similar to this process, with only small variations. As we can see from Figure 36, first we use CompostionGenerator to generate a composite service goal model. Then we generate a set of QoS attributes and input these attributes into the goal model and the constraints-generator. The constraints-generator takes the goal model as another input and generates the constraints with scoping information and QoS constraints.

We can use the goal model to generate, or functionally compose, one or more web service composition automata. We choose a type of verifier according to the purpose of the simulation and initialize this verifier. The verifier takes the service composition

automata and the constraint automata as inputs, and performs the verification algorithm and process. Finally, it outputs either success or failure.

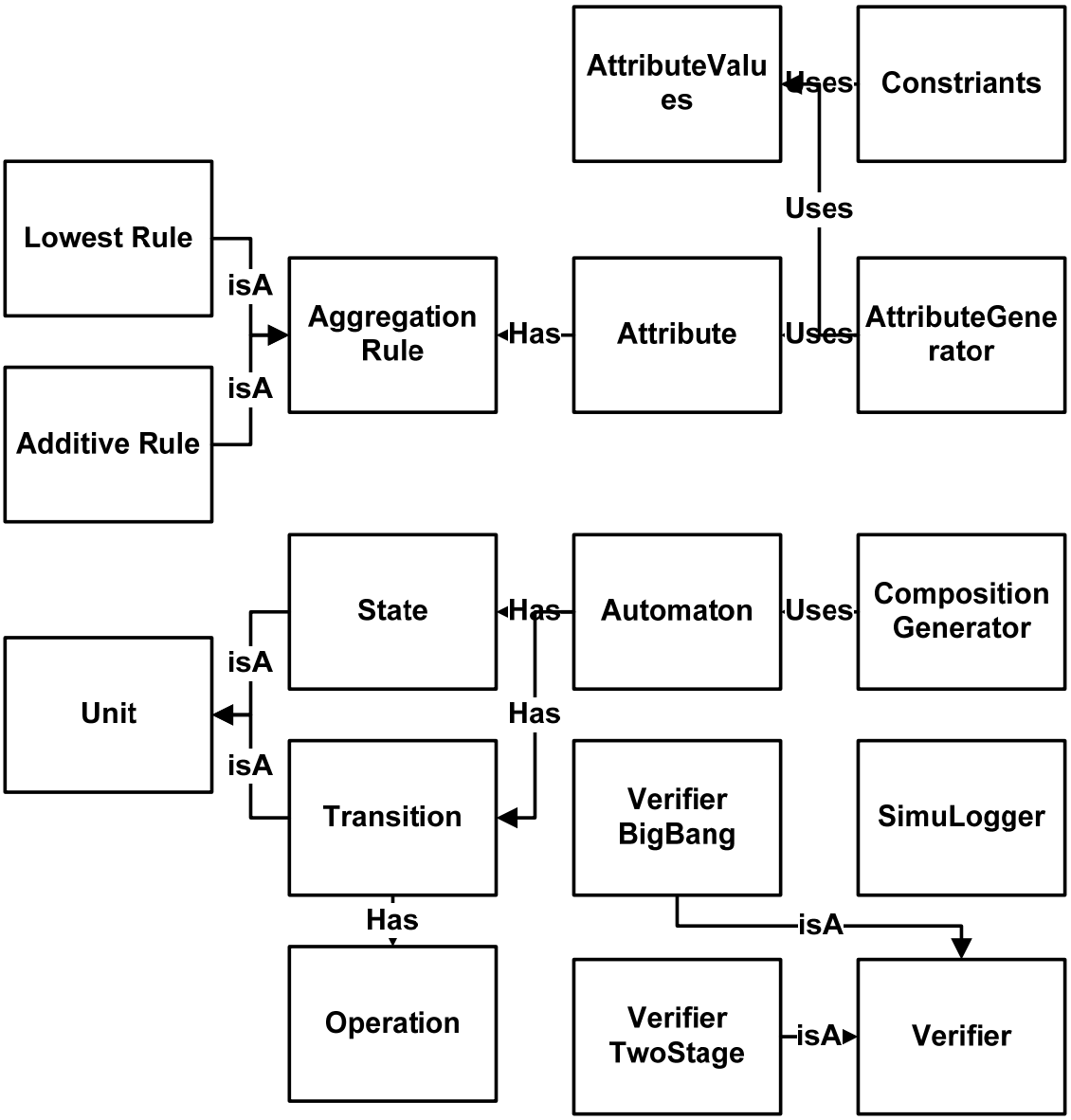

Figure 35. Class diagram excerpt for WSCSimu project

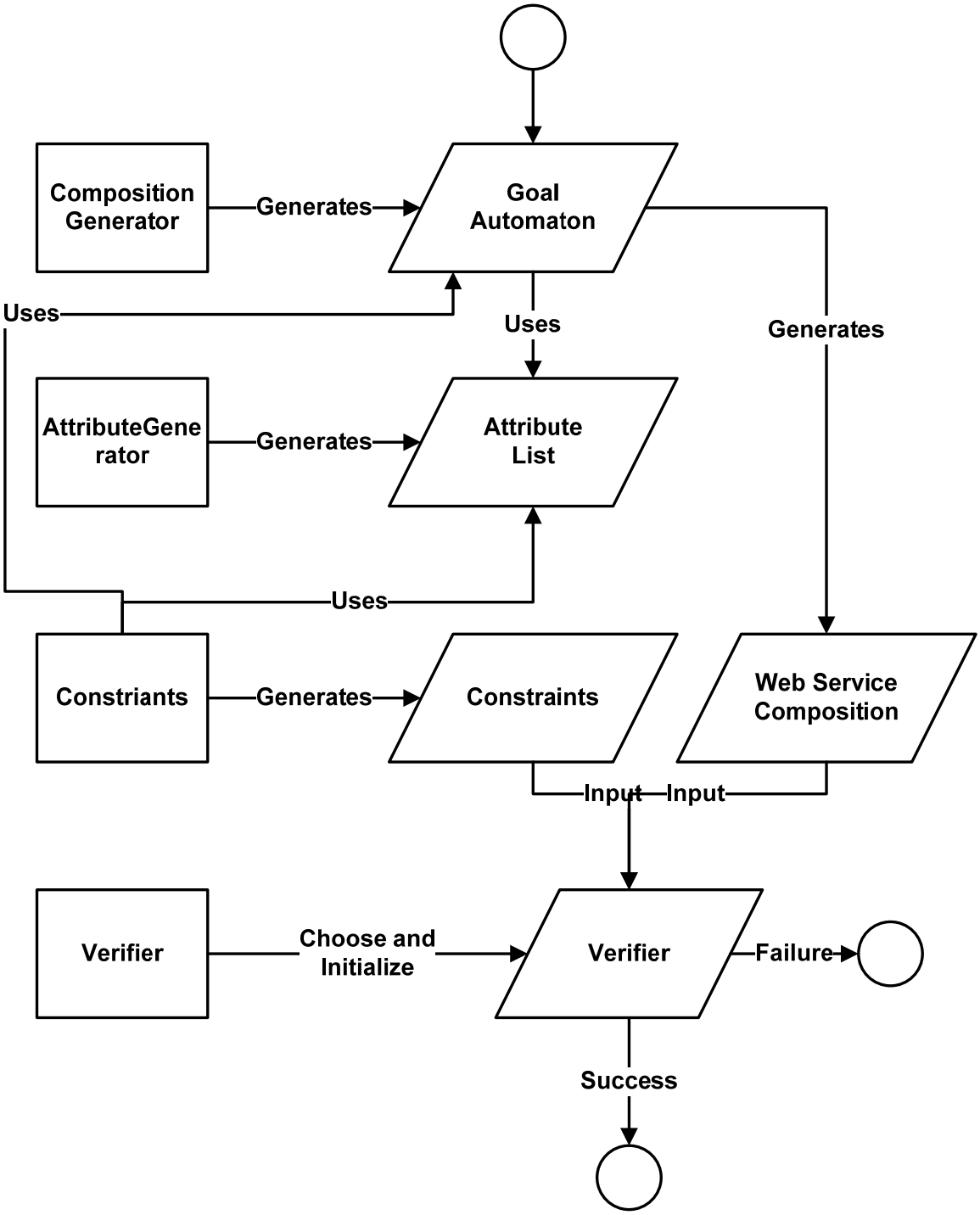

Figure 36.QoS verification process for web service composition in WSCSimu

6.2 Randomization

In the WSCSimu platform, the randomization follows a uniform distribution pattern in the valid value range for different generators. In order to make the simulation results comparable with each other, we use a fixed initialization seed for the Java random object (java.util.Random). In this way, the set of candidate web service compositions are the same in the different verification processes.

There are four random generation features in WSCSimu: Goal model generation, Web service composition generation, Constraint generation and QoS attribute generation.

In the goal model generation process, the fundamental concept is to randomly generate the automaton. There are four input parameters in this randomized automaton generation process: the number of tiers, the number of branches, the seed for the random function and the probability of branching. The number of tiers means the maximum number of states in the longest path starting from the entrance of the automaton to any end of the automaton. The number of branches means the maximum number of parallel sub-paths that can be generated starting at a state. For example, if the number of branches at StateA is 3, the number of the outgoing transitions for this StateA is 3 at maximum. The probability of branching means the probability to generate a set of parallel out-transitions for a state.

In the web service composition generation process, the goal model automaton and a set of QoS attributes are the inputs. The web service composition automata copy the workflow of the goal model and associate the QoS attributes with each state and

transition in the derived automata. A QoS value is uniformly generated for each QoS attribute in this process.

In the constraint generation process, the goal model automaton and the QoS attributes are the inputs. There are two special randomization parameters for this process besides the random seed: the probability to start building a scope and the probability to terminate building a scope. The generation process traverses the goal model automaton from the entrance point. For every state, there is a probability to start building the scope of the constraint. If the scope building is already started, there is a probability to terminate the scope building process for each state. When the scope is built, the QoS constraint is uniformly randomly generated based on the size of the scope.

In the QoS attribute generation process, the user needs to specify the number of attributes, the minimum valid value, the maximum valid value, the aggregation rule and a random seed. A QoS attribute value is uniformly generated in the valid value range.

6.3 Simulations

6.3.1 Simulation 1

Simulation goal: find a reasonable size of the web service composition search space to do further simulation.

Simulation design: we change the size of the web service composition search space while fixing the constraint scope size and goal model size. Table 7 shows the environment parameter setting for the simulation.

Table 7: Environment setting for Simulation 1		
Object	Parameter	
Global	Random seed	111
Goal model	Num of tiers	7
Goal model	Num of branches	2
Goal model	Probability of branching	50%
Attribute	Num of attributes	1
Attribute	Low bound	1
Attribute	High bound	10
Constraint	Probability to start	50%
Constraint	Probability to end	20%
Constraint	Number of constraints	1
Composition	Num of instances	200000
Verifier	Type	Normal

We add a loop at the web service composition generation part in the verification process (Figure 36). We create a set to load all web service compositions and call this set the WSC search space set. We log the time consumed by the verifier to verify the whole WSC search space set and divide this time by the size of the WSC search space set to get the verification time for each WSC. Then we repeat this process by changing the size of the WSC search space set from 1 to 131072 with an exponentially increasing rate. We compare verification time for each WSC in different search space sets. Figure 37 gives the results of this simulation, which shows how the search space size affects WSC verification time.

The results in Figure 37 tell us that when the WSC search space size is small (under 128), the installment overhead of Java object initialization is very obvious, which can result in a biased set of simulation data. From Figure 37, we can see that verification time is stable when the search space size is over 500. Thus, to have a more convincing

simulation data set and analyses in later simulations, we set a minimum limit of 500 for the size of the WSC search space.

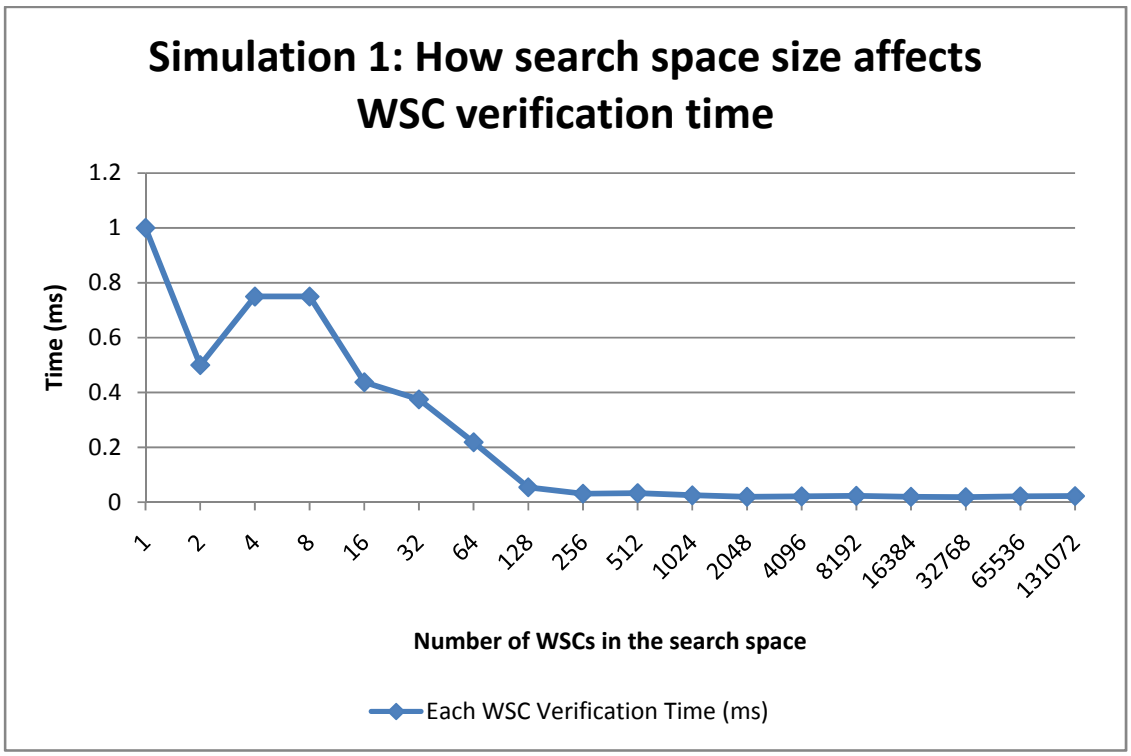

Figure 37. Simulation 1: How search space size affects WSC verification time

While recording the verification time in Simulation 1, we also record the success rate when verifying the WSC search space set. Because we generate the web service compositions, attributes and constraints with the fixed random seed for every round, the expected verification success rate should be fixed. However, when the WSC search space set is small (under 128), the success rate is not stable enough to represent the real success rate. From the results shown in Figure 38, we can derive the same results as Figure 37: a minimum of 500 WSCs in the search space set is a must to get stable results for later simulations.

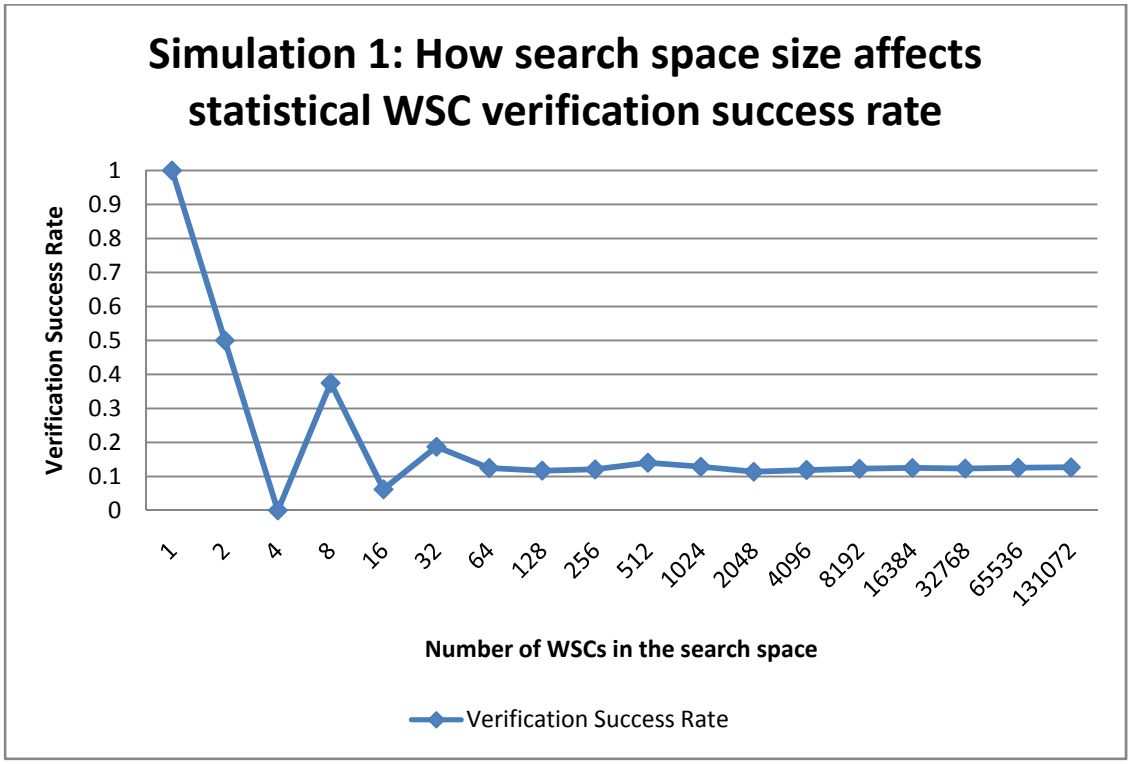

Figure 38. Simulation 1: How search space size affects statistical WSC verification success rate

6.3.2 Simulation 2

Simulation goal: find the relation between QoS constraint scope size and QoS constraint verification time.

Simulation design: we fix the size of the WSC search space and the size of the goal model, while changing the size of the scope in QoS constraint. Table 8 shows the environment parameter setting for the simulation.

Table 8: Environment setting for Simulation 2		
Object	Parameter	
Global	Random seed	111
Goal model	Num of tiers	12
Goal model	Num of branches	2
Goal model	Probability of branching	50%
Attribute	Num of attributes	1
Attribute	Low bound	1
Attribute	High bound	10
Constraint	Probability to start	100%
Constraint	Probability to end	0%
Constraint	Number of constraints	1
Constraint	Scope size limit	1~65
Composition	Num of instances	500
Verifier	Type	Normal

We design the simulation process based on the basic process shown in Figure 36. We create a loop for generating a search space set with 500 web service compositions. Then we create another loop to generate QoS constraints with different scope sizes from 1 to 65. Experiments show that verification time spent on a scope size beyond 65 is too long for conducting simulation on large WSC search space sets. For each scope size, we record and calculate average QoS constraint verification time for each WSC in this simulation. Note that the scope size of a QoS constraint means how many states and transitions in the scope of the QoS constraint are to be verified.

Figure 39 gives an idea of how scope size can affect the verification time of each QoS constraint. As expected, the time overhead of verifying a QoS constraint with a larger scope costs more time than smaller scopes. The relationship between scope size and constraint verification time is similar to a linear relationship.

Results from this simulation show that affixing scope information on QoS constraint, especially a small scope, can significantly reduce verification time of a QoS constraint. In other NFRs models for web service compositions, the NFRs are always defined and verified as global constraints, which is not as efficient and precise as our model.

During Simulation 2, we also recorded and calculated verification time for each state or transition in the constraint automaton. From the results in Figure 40 we can see that as scope size increases, verification time for each state or transition slowly decreases, reaching a stable point at scope size 37. The explanation for this phenomenon is that when the scope size is small (under 37), the overhead of initializing the verifier plays an important role in the whole verification process. When the scope is larger (over 37), this overhead is not as obvious as before since it is divided evenly over verification time of each state or transition.

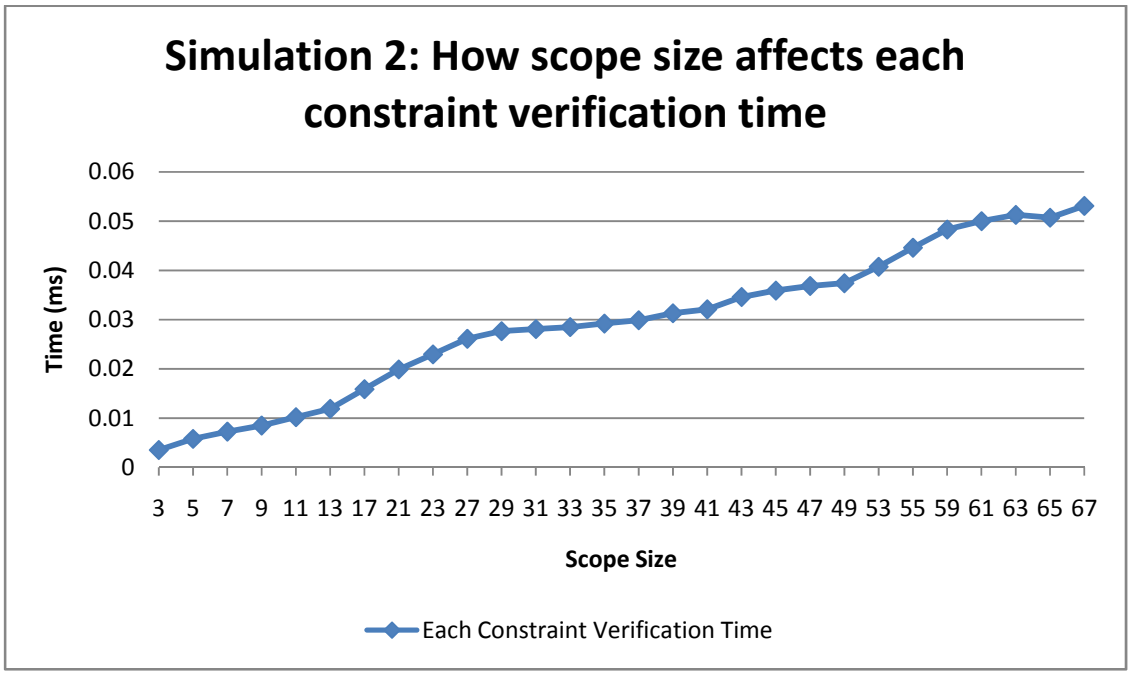

Figure 39. Simulation 2: How scope size affects each constraint verification time

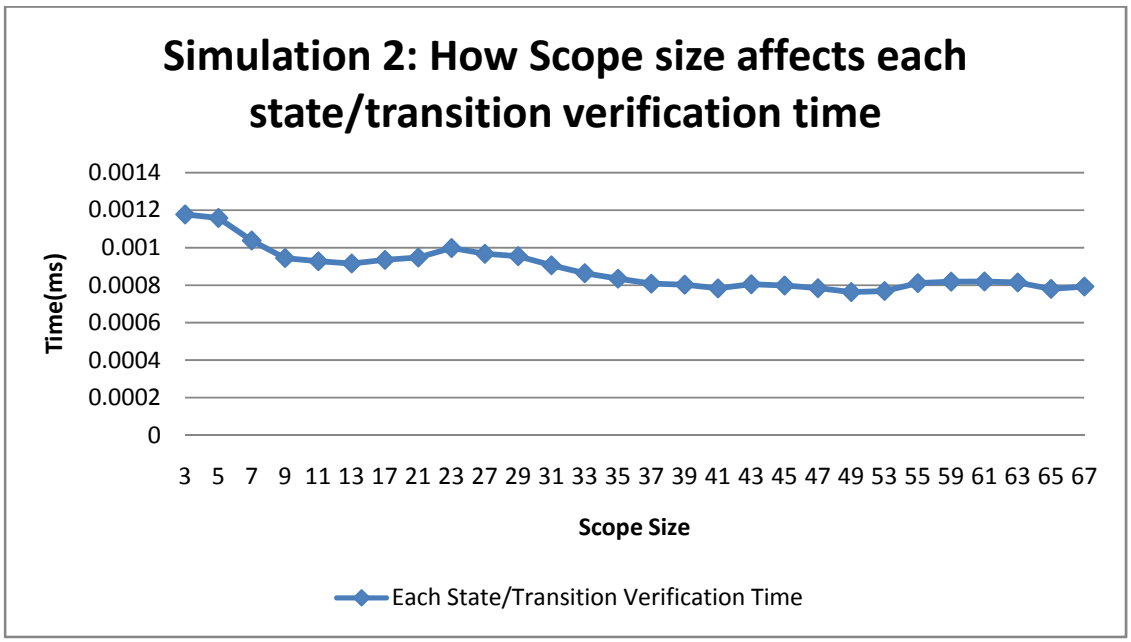

Figure 40. Simulation 2: How Scope size affects each state/transition verification time

6.3.3 Simulation 3

Simulation goal: compare the performance of the three strategies of verifying multiple QoS constraints as introduced in Chapter 4. These three strategies are independent verification strategy, big-bang verification strategy and two-stage verification strategy.

Simulation design: create two fixed sets of candidate web service compositions and a fixed set of constraints for verification. One set of candidate WSCs should have a low expected verification success rate and the other a high expected verification success rate. Conduct QoS verifications with the three different verifiers, each of which represents different strategies of verifying multiple QoS constraints, as shown in Table 4. Record and compare verification performance of the three strategies under different environments (different candidate WSCs sets). Table 9 shows the environment parameter setting for the simulation.

Table 9: Environment setting for Simulation 3

Object	Parameter	
Global	Random seed	111
Goal model	Num of tiers	7
Goal model	Num of branches	2
Goal model	Probability of branching	50%
Attribute	Num of attributes	1
Attribute	Low bound	1
Attribute	High bound	10
Constraint	Probability to start	50%
Constraint	Probability to end	20%
Constraint	Number of constraints	3
Composition	Num of instances	1000
Verifier	Type	Normal, TwoStage, BigBang

The simulation process is modified from the basic WSCSimu verification process shown in Figure 36. We use a modified web service composition generator to construct two sets of candidate web service compositions, each of which has 1000 candidates. We also use a modified constraint generator to generate a set of 3 QoS constraints. Through a guided generation process, we make one set of candidate WSCs to have a low expected verification success rate (5%) and the other having a high expected verification success rate (95%). Three strategies for verifying multiple constraints are used to verify all three QoS constraints for candidate WSC sets. Their verification times (their performance) are recorded by SimuLogger. We repeat this experiment 10 times to avoid biased simulation data and compare the derived results in Figures 41 and 42.

Figure 41 shows the performance comparison for the three strategies at a 5% expected verification success rate. We can see from the simulation results in this setting that the big-bang verification strategy gives the best verification performance, which is around 4 times faster than the independent verification strategy and 8 times faster than the two-stage verification strategy. The independent verification strategy gives an average performance. The two-stage verification strategy gives the worst performance, which is two times slower than the independent verification strategy.

We then analyze these simulation results to get a reasonable explanation.

When success rate is very low, the independent algorithm can terminate the whole verification process when it meets the first invalid constraint. However, the two-stage algorithm has an installment overhead of combining all constraints and the combined constraint has a more complex verification process.

The consequence of this overhead is that the independent verification strategy performs better than the two-stage verification strategy when the expected success rate is low, which means the independent verification strategy mostly terminates its verification process after it has verified the first constraint.

The big-bang verification strategy gives the best verification performance. Because we try a low expected success rate before simulation, we set the constraint threshold to be very small, which means attributes in WSCs can easily aggregate to exceed the constraint limit and violate the QoS constraint. In this case, because the Big-Bang verification strategy verifies all QoS constraints simultaneously, it can find any QoS constraint violation and terminate the verification process at a very early stage.

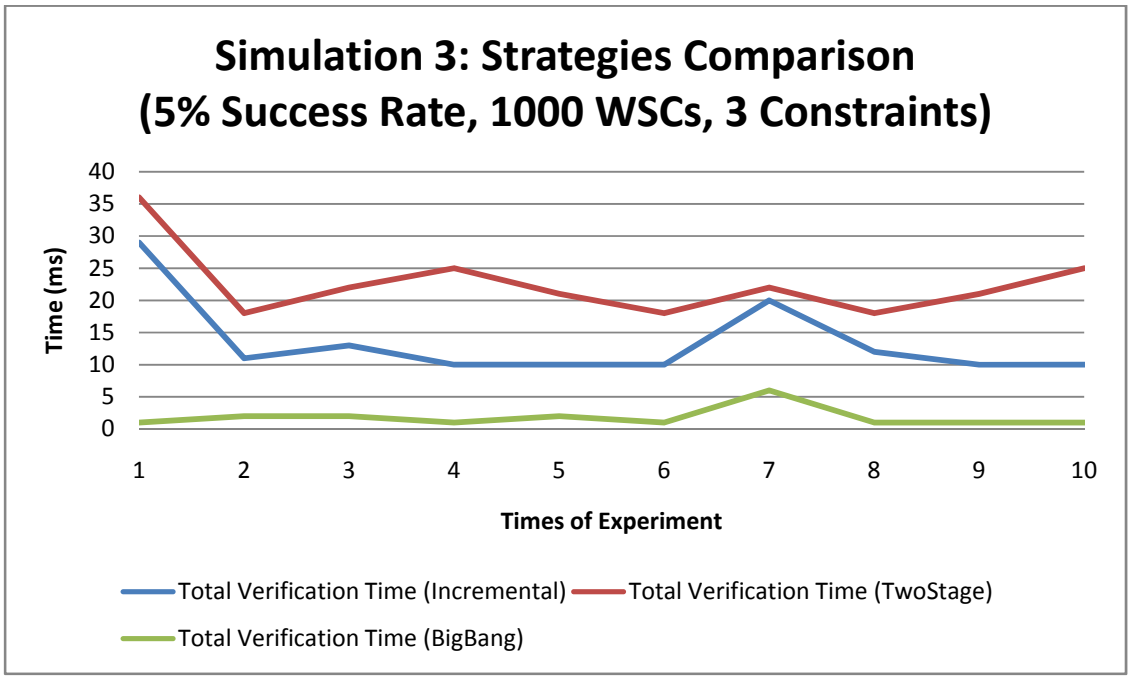

Figure 41. Simulation 3: Strategies Comparison (5% Success Rate, 1000 WSCs, 3 Constraints)

From the simulation data and the analyses, we can conclude that when QoS verification of WSCs has a low success rate expectation, we should choose the big-bang verification strategy to verify multiple QoS constraints. A low verification success rate expectation can usually be caused by composing a composite web service from an untrusted or an open modular service registry.

Figure 42 shows the performance comparison for the three strategies at a 95% expected verification success rate. From the figure we can see that the two-stage verification strategy gives the best performance, which is slightly faster than the independent verification strategy and 3 to 4 times faster than the big-bang verification strategy.

By comparing Figure 42 with Figure 41, we notice that the performance of the two-stage verification strategy does not change too much in either of simulation settings, while verification time of the independent verification strategy increases up to three times and of the big-bang verification strategy increases up to more than 20 times.

We analyze this phenomenon for an explanation. When the expected success rate is high, the independent algorithm needs to verify almost all candidate constraints (95% chance), which increases verification time nearly three times compared to verification at 5% success rate.

The time required by the two-stage verification strategy, on the other hand, does not change much because verifying a combined constraint is a little more efficient than

verifying them independently and the installment overhead of combining the constraints can be ignored when this combined constraint is reused 1000 times. The verification process and length do not change at all in any of the settings. This results in a stable performance for the two-stage verification strategy. However, the overhead of combining the constraints may be obvious when the candidate web service composition set is small. In this case, the performance of the two-stage verification strategy is still stable, but it may be not as efficient as the independent verification strategy.

The big-bang verification strategy, in this setting, needs to simultaneously verify three QoS constraints (at 95% success rate). Because implementation of the big-bang verification algorithm requires more intermediate variables and multiple threads, if it does not terminate at an early verification stage, its performance in terms of time consumed would be worse than verifying multiple constraints independently and incrementally.

We can conclude from this analysis that when the expected verification success rate is high and the candidate WSC set is large, the two-stage verification strategy is a good choice to maximize reuse of the combined constraint and achieve best performance. However, if expectation of the verification success rate is high and the candidate WSC set is small, the independent verification strategy is a better choice.

Web service composition from an enterprise internal modular service registry or a trusted service registry usually has a high verification success rate expectation.

When the expected verification success rate cannot be determined, the independent strategy or the two-stage strategy shall be a stable strategy.

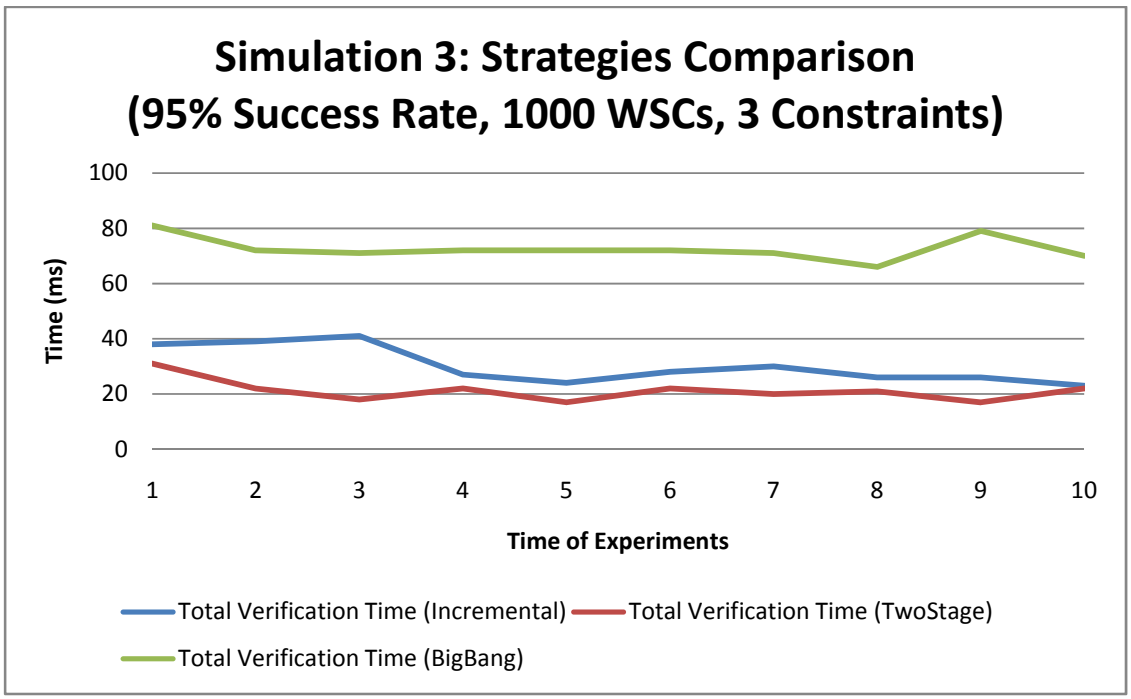

Figure 42. Simulation 3: Strategies Comparison (95% Success Rate, 1000 WSCs, 3 Constraints)

6.3.4 Simulation 4A and Simulation 4B

Simulation goal: To experimentally try the SPL based QoS constraint verification approach. Compare two ways of constructing indexed WSC search space and their respective performances.

Simulation 4A design: Simulation 4A incrementally constructs the indexed search space, which mimics on-demand verifications when a web based system is running. There is no preprocess required for constructing indexed WSC space before the web

system is deployed. For each verification encountered after the web system's deployment, we first lookup in the indexed WSC search space to see if this updated constraint has been verified and stored in the indexed WSC search space. If so, we return the stored verification results. Otherwise, we initiate the verification process on this updated constraint. The derived verification result is returned and stored.

Simulation 4B design: Simulation 4B builds the indexed WSC search space at once before the web system deployment and makes it ready for later verification lookups. In this process, construction of the indexed WSC search space is a preprocess for the verification system. The overhead for this preprocess is an extra overhead for the overall system verifications. The stored and pre-verified QoS constraints are not guaranteed to be used after the web system is deployed. Although the pre-construction of this indexed WSC search space can give a better performance afterwards, if QoS verification results are not frequently and adequately reused after system deployment, the overall average performance for each verification would not be very efficient.

Table 10 shows the environment parameter setting for these two simulations.

We have designed the simulation process based on the basic process shown in Figure 36. We created an indexed WSC search space, which is a hash table in our implementation, to store service compositions and their verification results mappings.

For Simulation 4A, we randomly generated 1000 web service compositions and QoS constraints to verify. For each verification, we randomly picked a web service composition from the 1000 composition instances and randomly picked a QoS constraint

to mimic customer choice. Then we query this composition-constraint pair in the indexed WSC search space. If it has been verified and stored in the hash table, we retrieve and return the verification results. Otherwise, we perform verification of this QoS constraint on this service composition, and store and output the verification results.

Table 10: Environment setting for Simulation 4		
Object	Parameter	
Global	Random seed	111
Goal model	Num of tiers	7
Goal model	Num of branches	2
Goal model	Probability of branching	50%
Attribute	Num of attributes	1
Attribute	Low bound	1
Attribute	High bound	10
Constraint	Probability to start	50%
Constraint	Probability to end	20%
Constraint	Number of constraints	1
Composition	Num of instances	1000
Verifier	Type	Normal
Verifier	Num of verifications	10000

Figure 43 shows the total time spent for 10,000 of QoS verifications when an indexed search space of 1000 WSCs is incrementally built. The indexed search space is not fully built (covering all 1000 WSCs) until the number of random verifications exceeds 5400. We notice that the verification time is stable after 1600. By analyzing the simulation data, we find that the indexed WSC search space is 80% built at the 1600th verification and 90% at the 2400th verification. We can conclude that the web based system using an incrementally built indexed WSC search space can reach a stable verification performance when the indexed WSC search space is 80% built. Taking the

number of possible service compositions as n, 80% to 90% build of the indexed WSC search space requires around 2n times of verifications.

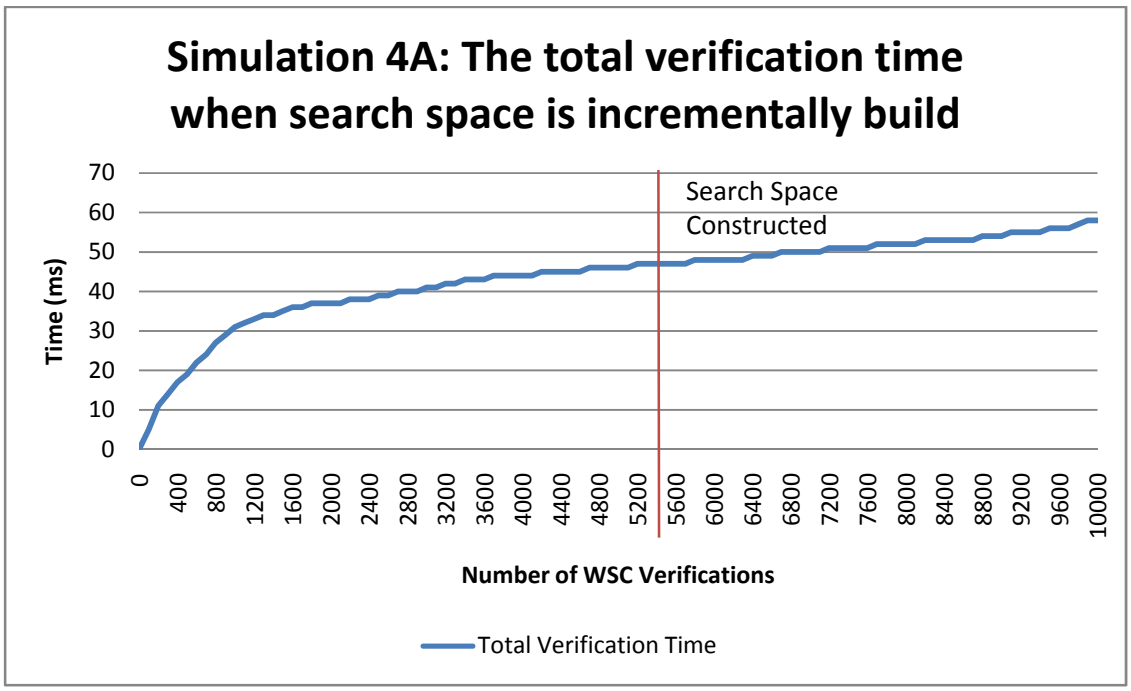

Figure 43. Simulation 4A: The total verification time when search space is incrementally build (SPL Approach)

For Simulation 4B, we randomly generated 1000 web service compositions and QoS constraints for verification. First we verified all 1000 web service compositions on all QoS constraints independently. We stored the composition/constraint pairs and their verification results mappings into indexed WSC search space, which is a hash table in the implementation. Then for any subsequent verifications after the web system's deployment, we only need to query the indexed WSC search space to retrieve the verification results.

Figure 44 shows the results of Simulation 4B. We can see the first 1000 verifications are merely used for constructing the indexed WSC search space, which are not real customers' requests. After the first 1000 verifications, the later verifications are only lookups in the indexed WSC search space, whose performance is stable and linear with the number of verifications.

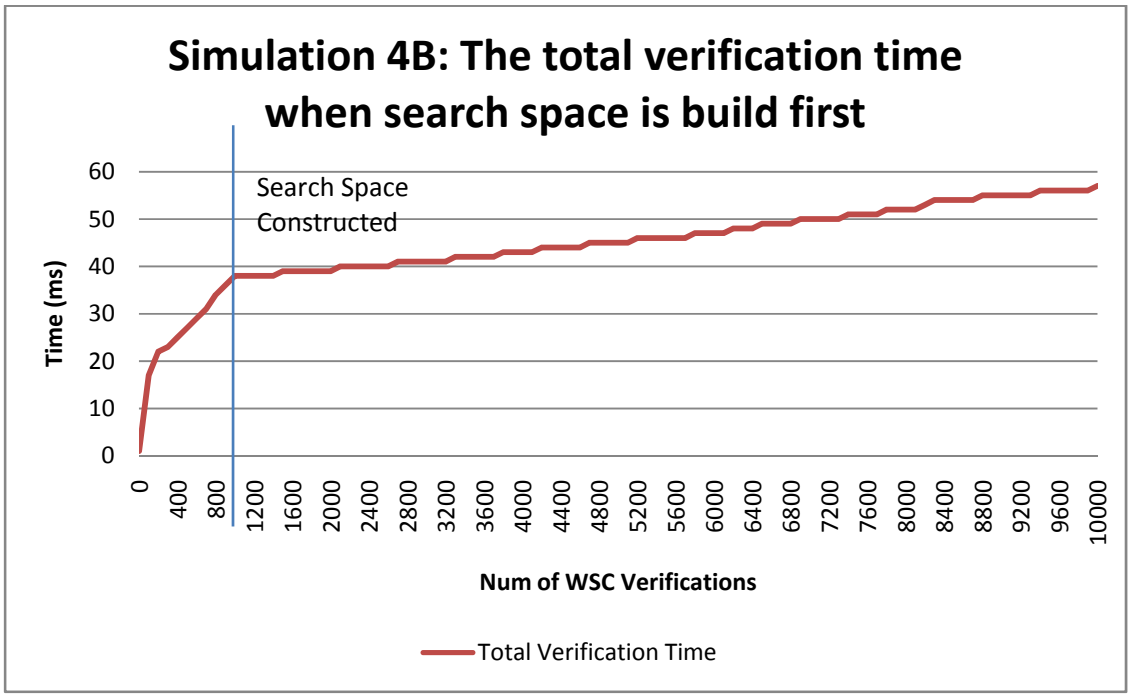

Figure 44. Simulation 4B: The total verification time when search space is build first (SPL Approach)

By comparing Figures 43 and 44, we can see that at the 10,000th QoS verification, the overall verification time in both simulations are similar (near 58ms). This is because there are only 1000 times of real constraint verification process conducted by the verifier. The remaining 9000 verifications were simply indexed search space lookups. However, in Simulation 4B, the first 1000 verifications were not requests from customers, which

means there is an advantage in using the incremental building process of the indexed WSC search space in real applications, especially when the overall number of possible QoS verifications after system deployment is not very large. Building the indexed WSC search space at one time and before the system deployment is thus more suitable for performance critical web based systems, where customers want the verification process to be completed as soon as possible.

6.3.5 Simulation 5

Simulation goal: show the difference between QoS constraint verification performance obtained by the normal approach (by calculating automata product) and the SPL-based approach (by pre-constructing the indexed WSC search space).

Simulation design: record and compare the performance of a normal QoS constraint verification process using the verifier and the performance of a normal QoS constraint verification process using the SPL approach when the indexed WSC search space is ready. Table 11 shows the environment parameter setting for the simulation.

The process design of this simulation is exactly the same as Simulation 4. The difference is that we record and calculate the verification time for each QoS constraint on a web service composition before the indexed WSC search space is fully constructed and after it has been constructed.

Table 11: Environment setting for Simulation 5		
Object	Parameter	
Global	Random seed	111
Goal model	Num of tiers	7
Goal model	Num of branches	2
Goal model	Probability of branching	50%
Attribute	Num of attributes	1
Attribute	Low bound	1
Attribute	High bound	10
Constraint	Probability to start	50%
Constraint	Probability to end	20%
Constraint	Number of constraints	3
Composition	Num of instances	1000
Verifier	Type	Normal
Verifier	Num of non-SPL verifications	1000
Verifier	Num of SPL verifications	1000

For 1000 times of verification, we get the computation time consumed before the indexed WSC search space is fully constructed and after it has been constructed:

Table 12. Simulation 5: Performance comparison with/without SPL approach

	Total Verification Time (ms)	Each WSC Verification Time (ms)
Before indexed search space constructed	66	0.066
After indexed search space constructed	7	0.007

From the results shown in Table 12 we can see that WSC verification with a fully constructed WSC search space is 9.4 times faster than verification by calculating the automata product.

From this simulation, we can conclude that for a highly customizable web service composition where the customers may request frequent QoS constraint verification, a SPL-based approach has a significant performance advantage compared to the traditional verification process.

6.4 Related Researches

Chandrasekaran et. al. [10] introduced a composite web service simulation tool named Service Composition and Execution Tool (SCET), which functionally composes the component services. SCET can conduct performance simulation and analysis on the composed web service in order to compare different composite web services before deploying them.

Narayanan and McIlraith [39] modeled the web services in Petri-Net formalism and provided verification simulations for semantic web service compositions. They also analyzed the complexity of these processes. Functional properties, such as reachability and liveness, can be verified for the web service compositions to ensure the behavior correctness.

In these two simulation based approaches [10][39] for web service compositions, the focus is on simulating the process of functional web service compositions and verifying the functional properties of the web service compositions. On the other hand, our simulation platform WSCSimu randomly generates the web service compositions based on a functional goal model, which is not a real functional composition process. WSCSimu instead focuses on the QoS constraint verification of these service compositions.

Chang et. al. [11] proposed a simulation platform for web service composition with that considers some QoS attributes including availability, reliability and performance. The simulation is conducted within an example of a travel planning web process. Different aggregated QoS values for the composite web service are calculated with different numbers of users as inputs. However, in this paper, the concept of QoS aggregation rules is not clearly identified. All simulations are conducted in the same manually-constructed web service composition. Thus, simulation results cannot be compared across multiple web service compositions. In our simulation platform WSCSimu, the aggregation rules are separately defined for each type of QoS attributes. The web service compositions and QoS constraints are automatically generated for verification.

From the evaluation point of view, the goal of the existing simulation platforms [10][39][11] is to evaluate the functional properties or non-functional properties for the composed web services and derive a web service composition that satisfies the requirements. On the contrary, WSCSimu targets on the evaluation of the QoS verification techniques for web service compositions, rather than the properties of the service compositions. With the help of the simulations with the WSCSimu, the engineers can compare different QoS verification strategies for web service compositions and choose the most efficient one for their projects.

6.5 Conclusion of the Simulations

Simulations conducted on WSCSimu platform have shown the validity and the advantages of our NFRs verification approaches introduced in Chapter 4 and Chapter 5.

Simulation 1 identifies the minimum size of the candidate web service composition search space in order for later simulations to have reasonable and analyzable results.

Simulation 2 shows that verification time of a QoS constraint has a positive linear correlation with the size of the constraint scope. It shows that a scoped QoS constraint has better verification efficiency than a global QoS constraint.

Simulation 3 compares the three verification strategies for verifying multiple constraints. With a low verification success rate expectation, the big-bang verification strategy gives the best performance. With a high verification success rate expectation and a large candidate web service composition set, the two-stage verification strategy gives the best performance. With a high verification success rate expectation and a small candidate web service composition set, the independent verification strategy gives the best performance. When the expected verification success rate cannot to determined, either the independent strategy or the two-stage strategy shall be selected as a stable verification strategy.

Simulations 4A and 4B compare the two ways of constructing the indexed WSC search space in a SPL-based web service composition NFRs verification approach. One way is the on-demand incremental construction after web system deployment and the other is to pre-construct it before web system deployment. The simulations show that there is an advantage in incrementally building the indexed WSC search space in real applications, especially when the overall number of possible QoS verifications after system deployment is not very large. Building the indexed WSC search space at once and

before system deployment is more suitable for performance critical web based systems where customers desire the verification process to be done as soon as possible.

Simulation 5 shows that for a highly customizable web service composition where customers may request frequent QoS constraint verifications, a SPL-based approach has a significant performance advantage compared to the traditional verification process.

The results of the simulations in this chapter support the approaches proposed in Chapters 4 and 5, showing their advantages across different circumstances.

CHAPTER 7. CONCLUSION AND FUTURE WORK

This chapter summarizes the work we have presented in this dissertation, followed by a summary of contributions and a brief discussion of some future directions.

7.1 Summary

As service oriented architectures have become more widespread in industry, many complex web services are assembled with independently developed modular services from different vendors. Although functionalities of composite web services are ensured during the composition process, non-functional requirements (NFRs) are often ignored in this process.. In Chapter 3 we introduced an Emergency Management System (EMS), which was used as a case study of a critical web based system in this dissertation.

To ensure quality of services in critical web based systems such as EMS, we proposed two effective approaches to verify that a composite web service not only offers the required functionality but also satisfies the desired NFRs. In Chapter 4, we described techniques for ensuring that a composite service meets user-specified NFRs that can be expressed in the form of hard constraints, e.g., "the messages of particular operations must be authenticated". We introduced an automata-based framework for verifying that a composite service satisfies the desired NFRs based on known guarantees regarding non-functional properties of component services. This automata-based model is able to

represent NFRs that are hard, quantitative constraints for composite web services. This model addressed two issues we identified in NFRs modeling and verification for composite web services that had been overlooked by other researchers: scoping of NFRs and consistency checking of multiple NFRs. We then introduced and compared three alternative verification strategies: the independent verification strategy, the two-stage verification strategy and the big-bang verification strategy, in the case where multiple NFRs exist, and analyzed their relative advantages and disadvantages under different scenarios. This approach to verifying NFRs can also support efficient re-verification of composite services as needed when NFRs are updated.

We then focused on multiple highly customizable composed web services where repeated verification of similar sets of NFRs can waste computational resources. We introduced a new approach to reconfigure the existing verification process by extending software product line engineering techniques to web service composition domain. Using a partitioning similar to that between domain engineering and application engineering in the product-line context, this approach first creates a web-service composition search space that satisfies the common requirements and then queries the search space as the user selects values for parameters of variation. It specifies the options that the user can select and constructs the resulting web-service compositions. This drew most of the computation overhead into the first stage in the design process to enable improved runtime efficiency in the second stage. The capability to reuse the composition search space provided a more efficient way to customize web services.. A complexity model showed the advantage of the product-line based verification process. Experiments on the

complexity model showed the relationships among the parameters in this approach and the ways of choosing values for these parameters to take advantage of this SPL-based approach.

A simulation platform WSCSimu was constructed to conduct experiments on the two verification approaches and three strategies we have introduced in this dissertation. The results of these simulation experiments were analyzed to show and compare the advantages of our approaches in various situations and environments over the traditional verification process and over each other.

7.2 Future Work

The work presented in this dissertation is a part of a larger topic that concerns how to model and verify non-functional requirements and properties for web service compositions.

Some interesting directions for extending our work are briefly described below.

Soft constraint modeling and verification for web service compositions

Soft constraints are constraints that cannot be simply judged as satisfied or unsatisfied. For example, the requirement that "the room shall be kept warm" for an automatic heater is a soft constraint because there is no way to clearly distinguish 'warm' and 'not warm'. If this constraint can be assigned an evaluation function to measure the unclear concept 'warm' by a value 'temperature', this soft constraint becomes a quantitative constraint. Such a soft constraint can be satisfied in percentage terms. If there is no such evaluation function for this soft constraint, this constraint is a qualitative

constraint or qualitative preference, e.g. United Airlines is preferred over Delta Airlines. It would be interesting to see if recent work by Santhanam et al. [52] to handle preferences (soft constraints) could be integrated with our approach.

QoS constraint violation monitoring and recovery for deployed web service compositions

Composite web services may fail to satisfy their non-functional requirements after deployment if the requirements or the environment evolve over time. For example, a new security policy may be required for an existing composite web service. In many cases, service users want to retain their current service provider, whom they may trust, to the extent possible. Rather than re-composing modular services from the service registry to meet the new goal, a better solution may be to replace only the components that caused the NFRs to fail. To fix the existing problematic service composition with minimal impact [26], we need to identify the service(s) now causing the NFR(s) to be violated. To achieve this goal, we need to take two steps: 1. to introduce a strategy to identify specific weaknesses of the composite web service that contribute the most to failure of updated non-functional requirements, and 2. to design a comparative evaluation method for the alternative modular services based on how much a replacement can improve failed QoS attributes of the service composition and how many replacements are required to satisfy the failed QoS constraints.

Towards this goal, we plan to explore the use of a product-line fault tree analysis. In previous work, we analyzed reliability and safety properties of a software product line using a product line fault tree [57]. This work modeled the software product line at the

architectural stage, using Architecture Analysis and Description Language (AADL) [57]. Each module associates with an error behavior model. The error behavior models utilize Markov-chains and interact with each other to simulate possible error events. The failure rates and failure scenarios of a member of the product line can then be analyzed by coordinating all associated modular error behavior models with the product line fault tree information. By taking the composition of a web service as a member of a product line, a similar analysis of the service composition appears to be possible.

Van Lamsweerde [60] also suggests the use of fault trees and cut sets for comparing the options in a goal model, similar to our planned use in comparing two possible replacement solutions.

Continuous work on WSCSimu platform

In this dissertation, we have introduced a simulation platform WSCSimu for verifying web service compositions. The purpose of constructing this platform was to demonstrate the practicability and the merits of our approaches. Through simulations of QoS property verifications for web service compositions on a randomly generated large data set and under different environment settings, we can analyze and compare the performance of WSC verification approaches and processes, which is convenient for testing new ideas in further research. The current weakness of this platform is that it lacks a user-friendly interface to set up the simulation environment, parameters and processes.

The next goal in building the WSCSimu platform is to create a user-friendly GUI to perform QoS verifications on web service compositions. An additional goal for the

WSCSimu is for it to output a graphical analysis of simulation results, rather than text-based data as in the current version.

7.3 Conclusion and Contribution

This dissertation addressed the following problems:

1. How to formally model a QoS constraint for a web service composition with scoping information incorporated and consistency checking enabled?
2. How to verify this QoS constraint model on a web service composition?
3. How to efficiently verify multiple constraints for a web service composition?
4. How to efficiently verify multiple constraints for multiple web service compositions?

To solve the first problem, we introduced an automaton-based model to represent a quantitative QoS hard constraint with. The scoping information can be affixed to this QoS constraint model. The contributions include:

- The introduction of a formal model of quantitative QoS for web service compositions.
- The introduction of a scoped QoS constraint model, which allows more efficient verification than for a global constraint.

To solve the second problem, we introduced a verification algorithm for this QoS constraint model that calculated the product of the QoS constraint automaton and the web service composition automaton. The algorithm provided:

- The capability of verifying an automaton-based QoS model for a web service composition
- The capability of consistency checking multiple QoS constraints.

To solve the third problem, we designed three alternative verification strategies: the independent verification strategy, the two-stage verification strategy and the big-bang verification strategy. The contributions of this work include:

- The introduction of three different verification strategies for multiple QoS constraints.
- The analysis of the relative advantages and disadvantages of the three verification strategies under different circumstances.
- The introduction of a strategy for choosing an efficient verification strategy when multiple QoS constraints exist for a web service composition.
- The support for efficient re-verification of composite services as needed when NFRs are updated.

Through investigations using the simulation platform we developed, WSCSimu, we concluded that with a low verification success rate expectation, the big-bang verification strategy gives the best performance. With a high verification success rate expectation and a large candidate web service composition set, the two-stage verification strategy gives the best performance. With a high verification success rate expectation and a small candidate web service composition set, the independent verification strategy gives

the best performance. When the expected verification success rate cannot be determined, a stable verification strategy is either the independent strategy or the two-stage strategy.

To solve the fourth problem, a new approach to customizing the functional and non-functional requirements of a web service composition was proposed which incorporated software product line engineering techniques into the web service domain.

The contributions of this were:

- The creation of a two-phase solution for efficiently handling mass-user service customization: a preparation phase in which the composition search space is constructed, and an implementation phase in which the web service composition is customized.
- The introduction of a highly customizable web service composition where customers may request frequent QoS constraint verification, with significant performance advantage compared to the traditional verification process

By following product line engineering procedures, web service compositions in the search space of a commercial service provider can be more readily composed, verified and reused in the presence of users' personal selections of their preferred variations.

Through the complexity construction for this SPL-based verification approach and its comparison with traditional verification complexity, we identified the impacts of the parameters in the SPL approach on its advantages over the normal approach. An increase of the candidate composition search space size can increase the reuse advantage

of applying our SPL approach. Meanwhile, the number of QoS variabilities has a positive effect on the SPL approach, while the size of the variability parameter set has a negative effect on the SPL approach when it is too large.

Overall, the SPL approach is more efficient when the web based system has a complex service composition (goal model size) with more varieties of candidate modular services. When designing the CVA table for this system, more variabilities with reasonable sizes of parameter sets are preferred.

The simulations in WSCSimu showed that there is an advantage in using the incremental building process of the indexed WSC search space in real applications, especially when the overall number of possible QoS verifications after system deployment is not very large. Building the indexed WSC search space at once and before system deployment is more suitable for performance critical web based systems where customers desire the verification process to be done as soon as possible. We also found that for a highly customizable web service composition where customers may request frequent QoS constraint verification, a SPL-based approach has significant performance advantage compared to the traditional verification process.

BIBLIOGRAPHY

[1] Ambler, S. (2004). The Object Primer: Agile Model Driven Development with UML 2. *Cambridge University Press*.

[2] Ardissono, L., Console, L., Goy, A., Petrone, G., Picardi, C., Segnan, M. and Theseider Dupre, D., (2005). Advanced Fault Analysis in Web Service Composition. *Proc. 14th international conference on World Wide Web*. pp:1090-1091.

[3] Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing*. vol. 1 issue. 1, pp. 11-33.

[4] Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Nadalin, A., Nagaratnam, N., Prafullchandra, H., Riegn, C., Roth, D., Schlimmer, J., Sharp, C., Shewchuk, J., Vedamuthu, A., Yalcinalp, U., Orchard, D., (2006). Web Services Security Policy Language Specification V1.2. *W3C member submission*. pp. 30~34. Available at http://www.w3.org/Submission/WS-Policy/ (Accessed Oct 2010).

[5] Balzerani, L., Di Ruscio, D., Pierantonio, A., De Angelis, G., (2005). A product line architecture for web applications. *Proc. ACM Symposium on Applied Computing (SAC2005)*. pp. 1689–1693.

[6] Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S. and Spoletini, P. (2007). Validation of web service compositions. *Software, IET*. Vol. 1. Issue 6. pp. 219 – 232.

[7] Brady, T.F. (2004). Emergency management: capability analysis of critical incident response. *Proc. Simulation Conference.* Vol. 2. pp. 1863 – 1867.

[8] Bruegge, B. and Dutoit, A.H. (2003). Object-oriented Software Engineering: Using UML, Patterns and Java. *Prentice Hall*. pp. 181-196.

[9] Capilla, R. and Yasemin Topaloglu, N., (2005). Product lines for supporting the composition and evolution of service oriented applications. *Proc. 8th Int'l Workshop on Principles of Software Evolution*. pp. 53–56.

[10] Chandrasekaran, S., Miller, J. A., Silver, G. S., Arpinar, B. and Sheth, A. P. (2003). Performance Analysis and Simulation of Composite Web Services. *Electronic Markets*. Vol. 13, Issue 2. pp. 120 – 132.

[11] Chang, H., Song, H., Kim, W., Lee, K., Park, H., Kwon, S. and Park J. (2005). Simulation-Based Web Service Composition: Framework and Performance Analysis. *Lecture Notes in Computer Science*, Vol. 3398/2005. pp. 352-360.

[12] Cheung, S. and Kramer, J. (1999). Checking Safety Properties Using Compositional Reachability Analysis. *ACM TOSEM*. Vol. 8. No. 1. pp. 49-78.

[13] Chung, L., Nixon, B. A., Yu, E. and Mylopoulos, J. (1999). Non-Functional Requirements in Software Engineering. *Springer*.

[14] Dustdar, S. and Schreiner, W. (2005). A survey on web services composition. *Int'l Journal of Web and Grid Services*, Vol. 1, No.1, pp. 1–30.

[15] Erl, T. (2005). Service-oriented architecture: concepts, technology, and design, *Prentice Hall*.

[16] Feather, M. S., Fickas, S., Van Lamsweerde, A. and Ponsard, C. (1998). Reconciling System Requirements and Runtime Behavior. *Proc. the 9th International Workshop on Software Specification and Design*. pp. 50-59.

[17] Fickas, S. and Feather, M. S. (1995). Requirements Monitoring in Dynamic Environments. *Proc. the Second IEEE International Symposium on Requirements Engineering*. pp. 140.

[18] Foster, H., Uchitel, S., Magee, J. and Kramer, J. (2003). Model-based Verification of Web Service Compositions. *Proc. 18th IEEE/ACM International Conference on Automated Software Engineering*, 2003, pp. 152-163.

[19] Foster, H., Uchitel, S., Magee, J. and Kramer, J. (2006). Model-Based Analysis of Obligations in Web Service Choreography. *Telecommunications*. pp. 149 – 159.

[20] Foster, H. (2008).Tool Support for Safety Analysis of Service Composition and Deployment Models. *Proc. IEEE International Conference on Web Services 08*. pp. 716-723.

[21] Guo, H., Shin, Y. and Lee, W. (2007). Enhanced Compositional Safety Analysis for Distributed Embedded Systems using LTS Equivalence. *Proc. 6th ICACS*. vol. 6. pp. 115-120.

[22] Hall, L. (2009). Police had to fill in forms and wait for David Iredale phone tapes. *The Australian*. Available at: http://www.news.com.au/national/red-tape-email-glitch-delayed-search/story-e6frfkvr-1225704503765 (Accessed Oct 2010).

[23] Huth, M., and Ryan, M., (2004). Logic in Computer Science. *Cambridge University Press*.

[24] Jaeger, M. C., Rojec-Goldmann, G. and Muhl, G. (2004). QoS aggregation for Web service composition using workflow patterns. *Proceedings of the Enterprise Distributed Object Computing Conference*. pp. 149–159.

[25] Jaeger, M. C., Rojec-Goldmann, G. and Muhl, G. (2005). QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms. *Lecture Notes in Computer Science*. Vol. 3760/2005. pp. 646-661.

[26] Kammer, P., Bolcer, Taylor, G. R., Bergman, M. (2000). Techniques for Supporting Dynamic and Adaptive Workflow. *Computer Supported Cooperative Work*. Vol. 9. Issue 3-4. pp. 269-292.

[27] Karam, M., Dascalu, S., Safa, H., Santina, R. and Koteich, Z. (2008). A product-line architecture for web service-based visual composition of web applications. *Journal of Systems and Software,* Vol. 81. Issue 6. pp. 855-867.

[28] Koehler, J. and Srivastava, B. (2003). Web service composition: Current solutions and open problems. *ICAPS Workshop on Planning for Web Services*, pp. 28-35.

[29] Krafzig, D., Banke, K. and Slama, D. (2005). Enterprise SOA. *Prentice Hall*.

[30] Lee, K., Kang, K. C. Kim, M. and Park, S. (2006). Combining feature-oriented analysis and aspect-oriented programming for product line asset development. *Proc. 10th International Software Product Line Conference (SPLC)*. pp. 102-112.

[31] Liu, J., Dehlinger, J., Sun, H. and Lutz, R. (2007). State-Based Modeling to Support the Evolution and Maintenance of Safety-Critical Software Product Lines. *5th Workshop on Model-Based Development for Computer-Based Systems: Domain-Specific Approaches to Model-Based Development (MBD'07)*. Tucson, AZ.

[32] Martin, R. (2004). Will Reliability Kill the Web Service Composition?. Dept of Computer Science. *Rutgers University*.

[33] Masters, S. (1997). Emergency room: London's Ambulances won't crash again. *Personal Computer World (Pcw.co.uk)*.

[34] Medjahed, B. and Atif, Y. (2007). Context-based matching for Web service composition. *Distributed and Parallel Databases* vol. 21, No. 1. pp. 5-3.

[35] Milanovic, N. and Malek, M. (2008). Adaptive Search and Learning-Based Approaches for Automatic Web Service Composition. *Web Service Research and Practices Chapter 6,* CyberTech Publishing. pp. 135-188.

[36] Mikalsen, T., Rouvellou, I. and Tai, S. (2001). Reliability of Composed Web Services--From Object Transactions to Web Transaction,. *Workshop on Object-Oriented Web Services, OOPSLA*. Tampa, Florida.

[37] Muehlen, M., Nickerson, J. V. and Swenson, K. D. (2005). Developing web services choreography standards—the case of REST vs. SOAP. *Decision Support Systems*. Vol. 40. Issue 1. pp: 9-29.

[38] Nadkarni, D. (2008). An iterative approach towards web service composition using feedback from analysis of composition failures. *MS Thesis*, Iowa State University.

[39] Narayanan, S. and McIlraith, S. A. (2003). Analysis and simulation of Web services. *Computer Networks: The International Journal of Computer and Telecommunications Networking - Special issue: The Semantic Web: an evolution for a revolution*. Vol. 42 Issue 5.

[40] Neumann, P. (1987~2009). The Risk Digest: Forum On Risks To The Public In Computers And Related Systems. *ACM Committee on Computers and Public*

Policy. Vol. 4. Issue 52 and Vol. 25. Issue 67. Available at: http://catless.ncl.ac.uk/Risks/ (Accessed Oct 2010).

[41] Newcomer, E., Lomow, G. (2004). Understanding SOA with Web Services (Independent Technology Guides). Addison-Wesley Professional.

[42] O'Neill, B. (2008). A Model Assessment Tool for the Incident Command System: A Case Study of the San Antonio Fire Department, Applied Research Projects. pp. 270.

[43] Oster , Z . (2009). Extending Substitutability in Composite Services by Allowing Asynchronous Communication. *MS thesis*, Iowa State University, USA.

[44] Pathak, J., Basu, S. Lutz, R. and Honavar, V. (2006). Parallel Web Service Composition in MoSCoE: A Choreography-based Approach. *Proc. 4th IEEE European Conference on Web Services*. pp. 3-12.

[45] Pathak, J., Basu, S. and Honavar, V. (2006). Modeling Web Services by Iterative Reformulation of Functional and Non-Functional Requirements. *Proc. 4th International Conference on Service Oriented Computing*. pp. 314-326.

[46] Pistore, M., Barbon, F., Bertoli, P., Shaparau, D. and Traverso, P. (2004). Planning and Monitoring Web Service Composition. *Artificial Intelligence: Methodology, Systems, and Applications*, Springer Berlin / Heidelberg. pp. 106-115.

[47] Rao, J. Kungas, P. and Matskin, M. (2004). Logic-based Web services composition: from service description to process model. *Proc. 2004 IEEE International Conference on Web Services*, pp. 446-453.

[48] Raya, M. Papadimitratos, P. and Hubaux, J., (2006). Securing Vehicular Communications. *Wireless Communications, IEEE Publication*. Vol. 13, Issue: 5. pp: 8-15.

[49] Robinson, W. (2003). Monitoring Web Service Requirements. *Proc. the 11th IEEE International Conference on Requirements Engineering*. pp.65.

[50] Robinson, W. (2009). Specifying and Monitoring Interactions and Commitments in Open Business Processes. *IEEE Software*. Vol. 26, Issue 2. pp.72-79.

[51] Samtani, G. and Sadhwani, D., (2002). Web Services Monitoring and Performance Management: Optimize the design and runtime performance of Web Services. *Web Services Journal*.

[52] Santhanam, G., Basu, S., and Honavar, V., (2008). A TCP-net based Algorithm for Efficient Composition of Web Services Based on Qualitative Preferences. *International Conference on Service Oriented Computing (ICSOC) 08*.

[53] Sohrabi, S., Prokoshyna, N. and McIlraith, S. (2006). Web Service Composition via Generic Procedures and Customizing User Preferences. *5th International Semantic Web Conference*. pp. 597-611.

[54] Sohrabi, S., Baier, J. and McIlraith, S. (2008). HTN Planning with Quantitative Preferences via Heuristic Search. *Oversubscribed Planning and Scheduling Workshop of ICAPS*, Australia.

[55] Stallings, W. (2007). Network Security Essentials: Applications and Standards, *Prentice Hall, 3rd Edition*.

[56] Sun, H., and Lutz, Robyn. (2006). Identifying Safety-Critical Requirement Defects Using a Tool-Based, Iterative Process, *Proc of the 17th IEEE International Symposium on Software Reliability Engineering (ISSRE'06)*. Raleigh, NC, USA.

[57] Sun, H., Hauptman, M. and Lutz, R. (2007). Integrating Product-Line Fault Tree Analysis into AADL Models. *10th IEEE International Symposium on High Assurance System Engineering (HASE)*, Dallas, TX.

[58] Sun, H., Lutz, R., and Basu, S. (2009). Product-Line-Based Requirements Customization for Web Service Compositions. *13th International Software Product Line Conference (SPLC '09)*.

[59] Sun, H., Basu, S., Lutz, R. and Honavar, V. (2010). Automata-Based Verification of Security Requirements of Composite Web Services. *21th IEEE International Symposium on Software Reliability Engineering (ISSRE'10)*, San Jose, CA, USA.

[60] Van Lamsweerde, A. (2004). Reasoning about Partial Goal Satisfaction or Requirements and Design Engineering. *ACM SIGSOFT Software Engineering Notes*. Vol. 29 Issue 6. pp.53-62.

[61] Van Lamsweerde, A. (2009). Reasoning About Alternative Requirements Options. *Lecture Notes in Computer Science. Springer Berlin / Heidelberg.* Vol. 5600, 2009. pp. 380-397.

[62] Van Lamsweerde, A. (2009). Requirements Engineering: From System Goals to UML Models to Software Specifications. *Wiley*. pp.24.

[63] W3C organization. (2007). Web Service Policy 1.5. Available at http://www.w3.org/Submission/WS-Policy/ (Accessed Oct 2010).

[64] Weiss, D. M. and Lai, C. T. R., (1999). Software Product Line Engineering. *Addison Wesley Longman*.

[65] Weiss, D. M., Li, J.J., Slye, H. Dinh-Trong T. and Sun, H. (2008). Decision-Model-Based Code Generation for SPLE. *Proc. 12th International Software Product Line Conference (SPLC)*. pp. 129-138.

[66] Weiss, M. and Mouratidis, H. (2008). Selecting Security pattern that Fulfill Security Requirements. Proc. Requirements Engineering (RE'08). pp: 169-172.

[67] Wu, J. and Yang, F. (2007). QoS Prediction for Composite Web Services with Transactions. *LNCS, Springer Berlin/Heidelberg*. Vol. 4652.pp: 86-94.

[68] Zachos, K. and Maiden, N. (2008). Inventing Requirements from Software: An Empirical Investigation with Web Services. *Proc. 16th IEEE International Requirements Engineering Conference.* pp. 145-154.

[69] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J. and Sheng, Q.Z. (2003). Quality driven web services composition. *Proc. 12th Int'l Conference on World Wide Web*. pp. 411–421.

[70] Zhang, D. Qi, Z. and Xu, X. (2008). Reliability Prediction and Sensitivity Analysis of Web Services Composition. *Petri Net, Theory and Applications*, I-Tech Education and Publishing. pp. 20-31.