Aspect Composition in Core Scenario Models

By

Mohammad AlHaj

A thesis submitted to the Faculty of Graduate Studies in partial fulfillment of the
requirement of the degree of

Master of Applied Science in Electrical Engineering

The Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa

Canada

April 2008

© copyright 2008, Mohammad AlHaj

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-40629-8
Our file Notre référence
ISBN: 978-0-494-40629-8

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Abstract

Aspect-Oriented Modeling (AOM) techniques allow software designers to address separately solutions for crosscutting concerns, and can be applied in different modeling languages. Core Scenario Model (CSM) is a scenario-based modeling language developed to support scenario-based design and performance analysis of software systems. The thesis proposes and implements a method for applying AOM to CSM models, by defining aspect models and composing them with a primary model. The inputs to this composition are the primary CSM model, the generic CSM aspect models, and the point-cut rules. The output is a composed CSM model, which consists of the primary model with the aspects “weaved” in. The thesis proposes three steps to perform the aspect composition. The first step identifies the join-points based on the point-cut rules. The second step instantiates a context-specific aspect model from a generic aspect model, by binding the formal parameters in the generic aspect to actual values from the primary model. The third step composes the context-specific aspect model with the primary model. The thesis describes the design and implementation and verification of CSM aspect composition, and applies it to a case study using security aspects.

Acknowledgements

First and foremost, I would like to thank Allah, the Most Glorious One, for the kind of people who surrounded me, and for everything that I have.

To my dear Mom, who died two years ago. I am sure she is now happy for my achievement.

To my dear father, who has given me the care and the love all these years.

To my dear wife, and my two angels (Aseel and Hanin), who supported me with care and love day and night.

To my brother, and sisters, who provided me with the hope and support.

To my father in law, mother in law, brothers in law, and sisters in law.

To my relatives, and my friends here in Canada, and overseas.

I would like to acknowledge a special thank my supervisor, Dr. Dorina Petriu, for providing an interesting topic, constant direction and valuable advice.

Table of contents

Chapter 1. Introduction	10
1.1. Objective and Motivation	10
1.2. Thesis contributions	15
1.3. Thesis content	15
Chapter 2. Background Literature.....	17
2.1. Aspect Oriented Modeling (AOM).....	17
2.1.1. Extending UCM with the AOM techniques.....	18
2.2. Model transformation: from UML to Performance model	20
2.2.1. Performance from Unified Model Analysis.....	22
2.2.2. Core Scenario Model	23
2.3. Eclipse.....	25
Chapter 3. Operations on CSMs	28
3.3. Algebraic Definition of CSM Operations	28
3.1.1. Definition of Scenario operations	28
3.1.2. Definition of Step operations	31
3.1.3. Definition of PathConnection operations.....	33
3.1.4. Definition of Resource operations	36
3.2. Java implementation of CSM operations	38
3.2.1. Package javax.xml.parsers	38
3.2.2. Package org.w3c.dom	39
3.2.3. Implementation of CSM operations.....	40
Chapter 4. Aspect composition in CSM	44
4.1. Identify join-points through point-cut expressions	44
4.1.1. Defining point-cut for general aspects	44
4.1.2. Defining point-cuts for security aspect	50
4.2. From generic to context-specific aspect models	52
4.3. Composition.....	54
4.3.1. InsertBefore rule	55
4.3.2. InsertAfter rule.....	57
4.3.3. ReplaceStep rule	59
Chapter 5. Aspect Composition Tool.....	61
5.1. UML Design	61
5.1.1. The Use case Diagram	61
5.1.2. The Package diagram.....	62
5.1.3. The Class Diagram.....	67
5.2. GUI Design	87
5.2.1. Design the operations of the GUI	87
Chapter 6. Verification and testing	101
6.1. Simple test cases	103
6.1.1. Testing General Aspect Composition procedure	103
6.1.2. Testing Security Aspect Composition procedure	105
6.2. Case study (TPC-W).....	108
6.2.1. Testing General Aspect Composition procedure	110

6.2.2. Testing Security Aspect Composition procedure	112
Chapter 7. Conclusion.....	115
7.1. Conclusion	115
7.2. Future work.....	116
References.....	117
Appendix A: Algorithms of Composition operations.....	123
Appendix B: The textual description of the usecases	140
Appendix C: Plug-in.xml	152
Appendix D: Test cases map.....	153
Appendix E: Samples of black box template for the general aspect composition procedure of Test cases.....	154
Appendix F: Samples of black box template for the security aspect composition procedure of Test case	156
Appendix G: Samples of black box template for the general aspect composition procedure of Case study.....	157
Appendix H: Samples of black box template for the security aspect composition procedure of Case study.....	160

List of Figures

Figure 2-1 Approach for performance analysis of UML models using AOM	18
Figure 2-2 Basic elements of UCM notation	19
Figure 2-3 Direct UML to LQN Transformation.....	21
Figure 2- 4 Two-step UML transformations.....	22
Figure 2- 5 PUMA architecture	23
Figure 2- 6 The metamodel for the Core Scenario Model	24
Figure 2- 7 Eclipse Platform architecture	26
Figure 3- 1 Scenario-related classes and their attributes	29
Figure 3- 2 Scenario class and its associations	29
Figure 3- 3 Step class and its associations	31
Figure 3- 4 PathConnection class and its associations.....	34
Figure 3- 5 Resource classes and their associations	37
Figure 4- 1 Applying NewHost operation	47
Figure 4- 2 Applying NewComponent operation	48
Figure 4- 3Nested service	49
Figure 4- 4 (a) Primary model: TestCase09, (b) Generic model: generic	53
Figure 4- 5Example of context-specific aspect model.....	54
Figure 4- 6 Insert before rule	56
Figure 4- 7 The options of implementing insertAfter rule.....	57
Figure 4- 8 Insert After rule	59
Figure 4- 9 Replace step rule	60
Figure 5- 1 Use case diagram of Aspect Composition Tool.....	62
Figure 5- 2 Package diagram of the aspect composition tool	63
Figure 5- 3 “actions” package.....	63
Figure 5- 4 “diagram” package	64
Figure 5- 5 “figures” package	64
Figure 5- 6 “Document” package	65
Figure 5- 7 “Element” package.....	65
Figure 5- 8 “Util” package.....	66
Figure 5- 9 Class diagram of Aspect Composition Tool	68
Figure 5- 10 CSMElement class	69
Figure 5- 11 Scenario Class	70
Figure 5- 12 Step Class	71
Figure 5- 13 Connector Class	72
Figure 5- 14 Resource class	74
Figure 5- 15 Refinement class	74
Figure 5- 16 CSMDocument class.....	76
Figure 5- 17 CSMUtil class	78
Figure 5- 18 GeneralPointcut class	79
Figure 5- 19 IDHandler class.....	80
Figure 5- 20 PointcutParser class.....	82
Figure 5- 21 Pointcut class.....	83
Figure 5- 22 SecurityPointcut class	85
Figure 5- 23 BindUtil class.....	86
Figure 5- 24 XMLDocumentWriter.....	86
Figure 5- 25 CSM Viewer menu.....	87

Figure 5- 26 CSM Viewer window.....	87
Figure 5- 27 menu items of the aspect composition tool.....	89
Figure 5- 28 Open Primary File dialog.....	89
Figure 5- 29 Viewing primary model	90
Figure 5- 30 Point-cut definition window.....	90
Figure 5- 31 Aspect Composition window	91
Figure 5- 32 Open Generic File dialog	91
Figure 5- 33 CSM Binding window.....	92
Figure 5- 34 Aspect Composition window	93
Figure 5- 35 The output of the composition	93
Figure 5- 36 Securing levels window	94
Figure 5- 37 Securing Host window	95
Figure 5- 38 Selecting Secured Hosts window	96
Figure 5- 39 Securing Host window	96
Figure 5- 40 Securing Server window	97
Figure 5- 41 Selecting Secured Servers	97
Figure 5- 42 Securing Server window	98
Figure 5- 43 Securing service Window	98
Figure 5- 44 Selecting Secured Services window.....	99
Figure 5- 45 Securing Service window.....	99
Figure 5- 46 Viewing CSM model	100
Figure 5- 47 The output of the expansion operation.....	100
Figure 6- 1GetCustRegPage scenario	109
Figure 6- 2 GetBuyConfirmPage scenario.....	109
Figure 6- 3 Generic SSL aspect model	110

List of Tables

Table3- 1 The algebraic notation of Scenario operations and their signatures.....	40
Table3- 2 The algebraic notation of Step operations and their signatures.....	41
Table3- 3 The algebraic notation of PathConnection operations and their signatures	41
Table3- 4 The algebraic notation of Resource operations and their signatures.....	42
Table3- 5 Other operations used in the CSM composition tool defined in the thesis	43
Table 6- 1 Black box testing template	103
Table 6- 2 Point-cut expressions input.....	104
Table 6- 3 Input section in Black box testing template for general aspect composition procedure.....	105
Table 6- 4 Summary of Testing General Aspect Composition procedure.....	106
Table 6- 5 Input section in Black box testing template for security aspect composition procedure.....	106
Table 6- 6..Summary of Testing Security Aspect Composition procedure	107
Table 6- 7 Point-cut expressions input.....	111
Table 6- 8 put section in Black box testing template for general aspect composition procedure.....	112
Table 6- 9 Summary of Testing General Aspect Composition procedure.....	113
Table 6- 10 Input section in black box testing template for security aspect composition procedure.....	113

Chapter 1. Introduction

1.1. Objective and Motivation

In system modeling, it is important to analyze features that address pervasive concerns separately in the earlier phases of the software development. This can not be achieved using classic decomposition techniques, such as the object-oriented (OO) methods, since classic techniques do not address well the decomposition of the kind of concerns which appear scattered and tangled in several units.

As a result of this situation, Aspect-Oriented Programming (AOP) approach was proposed to solve the problem [Kic97]. AOP provides new techniques that handles the separation of concerns, encapsulates these crosscutting concerns in special modules named aspects, and removes the scattering and the tangling. The Aspect-oriented techniques rely on three elements: a) the “core” or “primary” part which defines the core software design decisions, b) the “aspect” part which define the crosscutting concerns, and c) the associations between them known as “weaving” or “composition”.

Aspect-Oriented principles have been applied early at the implementation phase of the software development life cycle, through Aspect Oriented Programming (AOP). AOP is a paradigm that complements the object-oriented technology by adopting aspect-oriented techniques. A matured AOP is observed from the various powerful languages and implementation tools. One of the best known is AspectJ, which is an aspect-oriented extension to Java that supports programming cross-cutting code as separate aspect programs [Asp08].

The success achieved by using AOP in the programming phase made the Aspect-Oriented Software Development community (ASOD) realizes the importance of applying aspect-

oriented techniques to all phases of the software development life cycle (SDLC). In particular, Aspect-Oriented Modeling applies aspect-oriented concept to software models [Fra04]. Aspect-Oriented Modeling (AOM) handles crosscutting concerns at the model rather than program level. It allows software designers to describe features that address pervasive concerns separately, as aspect models, and to systematically incorporate them into a primary design model by using model composition techniques. AOM composition operates on the primary model (also known as base model) which reflects the core design decisions, and a set of aspect models, where each one of them describe system feature that crosscuts the primary model.

An aspect model defines a crosscutting feature as a pattern, called generic aspect model, independent of any primary model it may be composed with. Generic aspect models may be thought of as a template with formal parameters. For each insertion in the primary model the template is instantiated and its formal parameters are bound to values from the primary model using defined binding rules to produce a context-specific aspect model [Fra04], [Woo07]. The insertions, called the join-points, identify the location where the compositions occur.

A primary or aspect model may require different diagram types; for example, an aspect model can consist of both a class diagram and a sequence diagram. In this case, the model composition takes place by independently composing diagrams of the same type, which raises the need to ensure that information is represented consistently across the different types of diagrams in the composed model.

A lot of AOM research focuses on applying aspect-orientation to UML, by using the UML capabilities of supporting extensions mechanisms, such as stereotypes, tagged values, and

constraints, to tailor profiles that fit the need of specific domains. A number of UML profiles have been proposed in literature. For example, [Ald01] presented an initial discussion on a UML profile for Aspect-oriented modeling, while [Ald03] proposed a UML Profile to tailor UML for AOSD, providing conventions and guidelines for applying and specializing standard UML to the graphical notation for AOSD. The researchers in [Fue06] proposed some generic guidelines on how to construct a UML Profile for an AOD proposal. [Eve07] moves even further and uses the UML extension mechanism to propose a complete meta-model of the AspectJ in UML. The proposed meta-model is supported by UML2.0 compliant tools and allows all aspect-related concepts to be specified in meta-model terms.

Some researches introduced a profile that fits the needs of two phases of SDLC, as in [Mos07] which proposes a so-called Aspect-UML Profile to introduce a high-level modeling approach. The approach carries out the analysis and design of aspect-oriented software, as well as performs a formal verification of aspects composing such systems. Some researchers focus on AOM techniques by using certain UML diagrams as in [Kle04] which proposes a technique to statically weave behavioral aspects into sequence diagrams. [Dai06] and [Coo07] propose a Formal Design Analysis Framework to model and analyze aspect-oriented designs. FDAF supports the automated translation of extended Unified Modeling Language designs into existing formal notations, and provides a repository of predefined, reusable aspects for designer to use.

Even though most of the AOM research is based on UML, some research adopts aspect-oriented techniques for other modeling languages, such as [Mus07a] [Mus07b], and [Mus07c], which extends Use Case Map (UCM), User Requirement Notation, and Goal-

oriented Requirement Language (GRL) with aspect-orientation to produce AoUCM, and AoURN, and AoGRL respectively.

This thesis makes possible to use AOM techniques with another modeling language named Core Scenario Model (CSM) [Pet04], [Pet07a]. Core-Scenario Model (CSM) is an intermediate scenario-based model, which was developed at Carleton University to reduce the gap between software design languages (such as UML and UCM) and different kinds of Performance models. In the Performance from Unified Model Analysis (PUMA) project at Carleton University (described in section 2.2.1), the CSM model is used as an intermediate step in the transformation process from software design models annotated with performance information to performance models: a) the first transformation takes a software design model in a language such as UML and translates it into CSM; b) the second transformation translate the CSM model into a performance model (such as queueing networks, layered queueing networks, stochastic Petri nets, stochastic process algebras).

If software designers would like to use AOM techniques, they can do it even in the software design model (in UML or UCM) or at the CSM level.

AOM techniques applied to UML have to deal with primary and aspect models expressed by multiple types of diagrams, such as class, component, deployment, interaction, statecharts, and activity diagrams. The composition at the UML level is producing some complexity, especially when the aspect or primary model is modeled with different types of diagrams. In this case, the composition is performed by independently composing the diagrams of the same type. This operation requires the need to check for consistency across different diagram types in a composed model.

However, the complexity of AOM composition at the UML level may be avoided if the composition is performed at the Core Scenario Model (CSM). So, a possibility is to express the primary and aspect models in UML, convert them to CSM, and then compose at the CSM level. Performing the composition at the CSM level counts the following advantages:

- Requires a single composition algorithm, as opposed to the UML composition, which requires one composition for each diagram type.
- The composition process is simpler and more robust, since the CSM metamodel is much smaller than the UML metamodel.
- CSM has all the information required for the composition, since it combines behavior, resource, components, and performance attributes.

This thesis designs and implements a tool that supports the application of AOM at the CSM level. The tool implements operations needed to perform the aspect composition, which were introduced in [Woo07]. These operations identify the insertions (join-points) on the primary model, instantiate the context-specific aspect model from the generic one, and compose each context-specific aspect model in the desired insertion points in the primary model.

One of the requirements is that the tool produced in the thesis must be able to interoperate with other transformation tools built in the PUMA project. For instance, the tool should accept CSM input models generated from UML or should produce composed CSM output models that can be further converted to performance models. This means that the tool had to keep the existing CSM metamodel unchanged, even though some metamodel extensions could have helped the addition of AOM support to CSM.

1.2. Thesis contributions

The thesis designs, implements and tests a tool running on top of Eclipse v 3.1 allowing for the first time to apply Aspect-Oriented Modeling to CSM. The tool supports the following AOM steps:

- Defining the point-cut rules for general aspects and security aspects
- Identifying the join-points.
- Instantiating the context-specific aspect model, by binding the parametric values of the generic aspect to concrete ones extracted from the primary model
- Performing three types of aspect composition: insert before, insert after, and replace.

A set of aspect-related operations defined in algebraic form in [Woo07] extended with new operations are supported by the tool. The design of their algorithm, as well as their Java implementation is one of the thesis contributions. Another extension to [Woo07] proposed in the thesis is a general expression for point-cuts defined by a BNF grammar.

1.3. Thesis content

The thesis consists of seven chapters described as follows:

Chapter 1 presents the thesis motivation and contributions.

Chapter 2 presents an overview of the background literature. It discusses AOM, model transformation, PUMA, CSM, and Eclipse as a development tool.

Chapter 3 presents the algebraic definition, and the implementation of the CSM operations used in the CSM aspect composition.

Chapter 4 discusses the design and implementation of the three steps from [Woo07] used in CSM aspect composition: identifying the join-points, transforming generic aspect models into context-specific aspect models, and finally the composition of the context-specific aspects with the primary model.

Chapter 5 presents the UML design, and the GUI design of the aspect composition tool. The UML design includes the package diagram, the class diagrams, and the use case diagram. The GUI design includes the structure and the operations of the GUI.

Chapter 6 discusses the verification of the composition tool. Test cases were developed for the primary model and generic model to test the operations of the tool. In addition to that, a TPC-W case study was used to test the operations of the tool for a larger example.

Chapter 7 presents the conclusion and the future work.

Chapter 2. Background Literature

This chapter contains three sections related to the CSM aspect composition. The first section discusses Aspect Oriented Modeling (AOM), the second section discusses the Performance by Unified Model Analysis (PUMA) project and the Core Scenario Model (CSM). The third section presents Eclipse, which is the IDE tool that is used to develop the aspect composition tool.

2.1. Aspect Oriented Modeling (AOM)

Aspect Oriented modeling (AOM) is a modeling approach that allows software developers to address separate solutions for crosscutting objectives or concerns. These concerns can be either “concrete concern”, which can be realized by some elements in the system model, or a “qualitative concern”, which can be realized by the quality parameters of the system. Security aspect is considered a concrete concern, while performance aspect is considered a qualitative concern [Fra04], [Pet07a].

When applying AOM, two types of models are used:

- **Primary model:** It is the base architecture which describes how core functional requirements are addressed.
- **Aspect model:** It describes how a single concern is addressed in the core design. Aspect model can be either generic, when the annotations of aspect model are parametric, or context-specific, when these parametric annotations of the generic becomes concrete, related to the context of the primary model.

AOM techniques are used in [Pet07b] to assess the performance effects of different aspect realizations early in the development cycle. Figure 2-1 illustrates this approach. The primary model, and one or more of generic model are produced using an external UML tool. Then using defined Binding rules, the generic aspect model is instantiated to produce context-specific aspects. Binding rules define how to transform the parametric annotations of a generic aspect into concrete ones, related to the context in which the aspect will be used. Next using a set of composition directives, the context-specific aspect model is composed with the primary model. Composition directives are used to insure an accurate composition between the two models [Fra04], by specifying if a particular primary model element must be removed or added during the composition, and determining the order, if two or more context-specific models are composed with the primary model.

Figure 2-1 Approach for performance analysis of UML models using AOM

2.1.1. Extending UCM with the AOM techniques

AOM can be applied to other modeling languages than UML. An example is Use Case Maps (UCM), a visual scenario notation that is being standardized by the International

Telecommunication Union (ITU) as part of the User Requirements Notation (URN). Within URN, UCMs are complemented by a language for goal modeling and the description of nonfunctional requirements GRL [Mus07a]. UCMs describe the functional requirements of the system, and focus on the interaction between architectural entities without moving into the details of exchanging message and communication infrastructure. Figure 2-2 is showing the basic elements of UCM, which are used to visualize different kinds of systems such as service-oriented, concurrent, distributed, and reactive systems.

Figure 2-2 Basic elements of UCM notation

In [Mus07a] and [Mus07b], Aspect-oriented Use case Maps (AoUCMs) are introduced as a result of applying aspect-oriented techniques within UCM. Aspect-oriented Use Case Maps (AoUCM) are a modeling technique for early aspects and model concerns expressed as use cases or scenarios at an abstraction level suitable for requirements models. AoUCM are an extension of the Use Case Maps (UCMs) modeling language. Aspect-oriented Use Case Maps (AoUCM) extend UCMs with the concepts of aspect, advice map, pointcut stub, and pointcut map in order to enable the modeling of concerns. These concepts, however, do not require new notational elements as they are just specializations of already

existing concepts. The concrete syntax of UCMs does not need to be altered, allowing the requirements engineer to continue using UCMs as before.

2.2. Model transformation: from UML to Performance model

In software engineering, performance engineering is gaining more attention, as many software projects do not meet their performance requirements. Performance testing is in many cases one of the last activities before the applications are released. However, the cost for doing performance analysis only at the end of the development cycle can be considerable and could cause severe performance disasters in software projects. It has been recognized that performance analysis needs to be integrated throughout the software life-cycle [Smi90]. To achieve this, earlier performance analysis requires the use of models for insight into the sources of problems.

To bridge the gap between software analysis and performance analysis, OMG introduced the UML Profile for Schedulability, Performance, and Time Specification [OMG05], and its successor, the UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) for UML2 [OMG07]. The profiles add light-weight extensions to the UML metamodel to express the basic abstractions used in performance analysis. Adding quantitative performance annotations to UML models enables their transformation into performance models, which can be analyzed with existing performance analysis tools. The performance analysis results can be fed back to the software engineers at an earlier stage of the development process, allowing them to improve the products as necessary.

There are two approaches used in model transformation from software design to performance models. The first approach performs a direct transformation of UML+SPT

model to performance model [Pet04]. In this approach, the performance model structure is generated from UML model of the high-level software architecture, and the UML deployment diagram, while the performance model parameters generated from UML models of key scenarios with performance annotations, such as interaction diagrams, activity diagrams, and sequence diagrams [Pet04].

Figure 2.3 describes UML model transformations and performance model solutions, where the relevant information in the UML design U is scattered in behavior and deployment sub-models, and possibly in other sub-models as well [Pet07a].

Figure 2-3 Direct UML to LQN Transformation

The second approach transforms the UML model to an intermediate scenario-based form, which latterly is transformed to a performance model. The core scenario model, described in the next sections, is an intermediate form which collects the UML design into a form that is convenient for generating the performance model in a consistent form.

Figure 2.4 describes the two-step UML transformation, where the transformation U2C extracts the scenario model, and C2P derives a performance model. Alternative C2P transformations may support different performance formalisms for P [Woo05]. This approach is adapted by PUMA which will be described in the next section.

Figure 2- 4 Two-step UML transformations

2.2.1. Performance from Unified Model Analysis

Performance is a non-functional property, which can be annotated during the UML design using SPT [OMG05] or MARTE [OMG07] profile. Transforming the annotated UML diagrams to various performance models requires adding an intermediate support. This support was developed by developing a unified approach to build a performance model from scenario-based software specification.

Performance from Unified Model Analysis (PUMA) is an ongoing research project in Department of System and Computer Engineering at Carleton University. The goal of the project is to provide a unified interface between different kinds of design information, and different kinds of performance models [Woo06]. The importance of the project came from the complexity of making performance analysis using UML diagrams annotated by SPT or MARTE annotations, and transforming these diagrams to various types of performance models.

Core Scenario model was developed to act as an intermediate model between the annotated UML diagrams, and performance models. Figure 2-5 describes PUMA architecture.

The approach of the project consists of the following steps:

1. A UML tool produces SPT annotated UML model
2. The annotated model is first transformed to CSM model.

3. The CSM model is then transformed to Performance model of any kind.
4. The performance model is analyzed, and the results and recommendation is fed back to the tool.

Figure 2- 5 PUMA architecture

2.2.2. Core Scenario Model

The Core Scenario Model (CSM) was developed as part of PUMA. It provides a meta-model for an intermediate form which correlates multiple UML diagrams, extracts the behavior elements with the performance annotations, attaches important resource information that is obtained from the UML, and supports the creation of many different kinds of performance models [Pet04], [Pet07a]. CSM meta-model captures the performance entities in the SPT profile. Figure 2-6 describes the metamodel of the CSM.

Each CSM instance model is a collection of scenarios, and resources. A scenario is described as a flow of steps linked together by connectors, and a resource is an element which executes operations. Each scenario instance has a collection of steps, and pathConnections. A step is an action within a scenario, and a pathConnection is an

operator which is linked to at least one step. The CSM metamodel defines two types of steps: a primitive step, and a refined step. The primitive step operates a single action, while the refined step operates multiple actions and can be scattered to smaller actions.

Figure 2- 6 The metamodel for the Core Scenario Model

There are also two special types of steps related to acquiring and releasing resources. The resource acquire describes the acquired component, and the resource release describes the released resource.

The CSM metamodel contains seven types of Pathconnections. The Start is linked to one target step. The End is linked to one source step. The Sequence is linked to a collection of target steps, and source steps. The Branch describes an alternative path, with one source step, and two or more target steps. The Merge describes alternative paths, with two or more source steps, and one target step. The Fork describes a parallel path, with one source step, and two or more target steps. The Join describes a parallel path, with two or more source steps, and one target step [Pet07a].

CSM metamodel also, defines a various classes for resources. The super class GeneralResource has two subclasses ActiveResource class and PassiveResource class. Active resources include processing resources and external services, while passive resources include components.

A scenario has a workload, which defines the capacity of usage. CSM structure contains two types of workloads; a close workload, and an open workload. An open workload defines an unlimited number of requests, while a close workload defines a fixed number of requests.

A message can be associated to any type of path connections. It defines the message size sent between system nodes. There are three kinds of messages: async, sync and reply.

2.3. Eclipse

Eclipse is an open-source Integrated development environment (IDE) written primarily in Java. It was first an IBM project developed by Object Technology International (OTI) as a

replacement for VisualAge. Now it is under Eclipse foundation. The Eclipse Platform architecture, described in figure 2-7, consists of Workbench subsystem, Workspace subsystem, Help subsystem, and Team subsystem. There is also a plug-in development Environment.

Figure 2- 7 Eclipse Platform architecture

1. Workbench subsystem is the desktop development environment. It achieves seamless tool integration, and provides the developer with essential functions for the creation, management, and navigation of workspace resources.

A workbench window may contain more than one perspective. Each perspective contains views, editors, and menu, and tool bar controls.

Workbench API composed of two toolkits, the Standard Widget Toolkit (SWT), and JFace. SWT is a well-designed native UI library for Java, which provides the foundation for the entire eclipse. SWT is limited to using simple data types, such as strings, numbers, and images. JFace is a UI toolkit with classes for handling many

common UI programming tasks. JFace is window-system-independent in both its API and implementation, and is designed to work with SWT without hiding it. JFace includes the usual UI toolkit components of image and font registries, text, dialog, preference and wizard frameworks, and progress reporting for long running operations [Ecl08].

2. Workspace subsystem is a directory hierarchy containing both user files such as projects, source code,...etc, and plug-in state information such as preferences
Workspace is supported with a history mechanism to keep track of the previous changes. It also provides a marker mechanism for annotating resources.
3. Help subsystem supports browsing, searching, book-marking, and printing help documentation. It also supports text searching capability for finding the information by searching the phrase or keyword. The documentation is organized into sets of information that are analogous to books.
4. Team subsystem handles managing and versioning resources using Concurrent Versions System (CVS).
5. Plug-in development Environment provides a rich set of tools to create, develop, test, debug and deploy Eclipse plug-ins. A plug-in is a program in eclipse which can be developed and delivered separately. Each plug-in has a plug-in manifest, which defines the structure and the behavior of the plug-in, and how this plug-in relates to the others in the system.

Chapter 3. Operations on CSMs

This chapter consists of two sections; the first section describes the algebraic definition of CSM properties and operations from [Woo07], which are used in the composition of the aspects models with the primary model. The section presents only the operations that were introduced in [Woo07]. The second section describes the implementation of these operations using Java.

3.3. Algebraic Definition of CSM Operations

[Woo07] introduced a set of CSM operations, which covers the basic concepts supporting aspects in CSM. The operations cover the major requirements of CSM aspect composition, starting from defining the point-cut rules for the primary model, until composing the context-specific aspect model with the primary. The paper [Woo07] introduced the algebraic definitions of these operations, which help the developer to understand the logic behind the operations, and makes it easy for the developer to code them. The introduced CSM operations are related to different classes of the CSM metamodel (which is kept unchanged from [Pet07a] for interoperability reasons). The operations belong to the Scenario class, Step class, PathConnection class, Component class, and GeneralResource class.

3.1.1. Definition of Scenario operations

In CSM, a scenario is an ordered sequence of steps, each of which can be further refined by a sub-scenario. The fragment of the CSM metamodel describing the Scenario and its associations with other classes shown in figure 3-1 and figure 3-2 are used by the scenario

operations which navigate CSM metamodel elements that are conform to the CSM metamodel.

Figure 3- 1 Scenario-related classes and their attributes

Figure 3- 2 Scenario class and its associations

Scenario class is an instance of CSM class. It has a Step class, PathConnection class, and Classifier class, with a multiplicity of (1...*), (0...*), and (1...1) respectively. Also, It has an association with Refinement class, with a multiplicity of (0...*).

CSM profile provides a various properties for the Scenario class:

1. id: A unique identity for a scenario.

2. name: It can be any name; however it is preferred to be a name that is related to the scenario function.
3. description: Describes the scenario operation. This property can provides any information which is considered valuable.
4. traceabilityLink: Define a copy of UML identifier of the Scenario.
5. probability: The probability that this scenario will be executed
6. transaction: Define whether this scenario is a transaction or not.

In addition to the previous properties, [Woo07] are defining the following operations in algebraic form:

1. $SC = (S, C, R)$: A scenario SC is defined as a set of Steps S , a set of Connectors C , and a set of Resources R .
2. $ReplaceStep(SC, StepA, SZ)$: Defines an operation in which $StepA$ is replaced by a sub-scenario SZ in a scenario SC .
3. $C' = ReplaceConn(C, ConnA, StepA, StepZ)$: Defines an operation which returns a set of connectors C' , where connector $ConnA$ has been modified by replacing references to $StepA$ by references to $StepZ$.
4. $InsertBefore (SC, StepA, SZ)$: Defines an operation which inserts a sub-scenario SZ before $StepA$.
5. $InsertAfter (SC, StepA, SZ)$: Defines an operation which inserts a sub-scenario SZ after $StepA$.
6. $Expand (SC, StepA, SZ)$: Defines an operation in which a composite step $StepA$ in scenario SC which references a sub-scenario SZ has been flattened by inserting the steps of the sub-scenario in place of $StepA$.

The implementation of these operations is described in section 3.2.3

3.1.2. Definition of Step operations

In CSM, Step is defined as an action within a scenario. A step can be a single step, or it can be composed of a set of smaller steps, which can be represented as a sub-scenario. The class diagram of step and its associations with other classes are described in figure 3-3

Figure 3- 3 Step class and its associations

Step class is an instance of Scenario class. It has associations with PathConnection class, and Component class, with a multiplicity of (1..1), and (0..1) respectively. It has an ExternalDemand class, with multiplicity of (0..*) . Also it is a generalization of ResourceAcquire class and ResourceRelease class.

CSM profile provides a various properties for the Step class. These properties and some additional properties which will be used in developing the CSM operations are described below.

1. id: A unique identity for a step within a scenario.
2. name: A description of the step action.

3. description: Describes the step operation. This property can provides any information which is considered valuable.
4. traceabilityLink: Define a copy of UML identifier for the Step.
5. hostDemand: The total demand of the step on its processing resource.
6. probability: The probability that this step will be executed.
7. repCount: The times of repetition of the step.
8. Step.predecessor: An association role name which identify the preceding connector that the step is linked with. The predecessor holds the id of the preceding connector.
9. Step.successor: An association role name which identify the succeeding connector that the step is linked with. The successor holds the id of the succeeding connector.
10. Step.type: { call | reply | async }: The type of message that triggers the step. The default value is async.
11. Step.component: The component of the resource that the step was most recently acquired.
12. Step.component.host: The processing resource of this step.
13. Step.ResContext: The set of resources that have been acquired and not yet released.
14. Step.subscenario: A scenario which is a refinement of the step.

In addition to the previous properties, [Woo07] are defining the following operations in algebraic form:

1. $S = \text{Steps}(\text{SC})$: Defines an operation which returns a set of steps in scenario SC.
2. $s = \text{First}(\text{SC})$: Defines an operation which returns the first step in the scenario SC.
3. $s = \text{Last}(\text{SC})$: Defines an operation which returns the last step in the scenario SC.
4. $\bullet\text{StepA}$: Defines an operation which returns the preceding connector to StepA within its scenario.

5. {••StepA}: Defines an operation which returns a set of steps that is preceded StepA within its scenario.
6. {•!StepA}: Defines an operation which returns the set of all preceding steps and connectors that is preceded StepA within its scenario.
7. StepA• : Defines an operation which returns the succeeding connector to StepA within its scenario.
8. {StepA••}: Defines an operation which returns a set of steps that is succeeding StepA within its scenario.
9. {StepA!•}: Defines an operation which returns the set of all succeeding steps and connectors that succeeding StepA within its scenario

3.1.3. Definition of PathConnection operations

In CSM, PathConnection is defined as an operator which is linked to at least one element.

An element could be a step, an acquired resource, or a released resource.

CSM metamodel defines seven types of PathConnection described below.

1. Start: Indicates the beginning of a scenario. Each scenario has a unique Start connector, which must begin with.
2. End: Indicates the end of a scenario. Indicates the end of a scenario. Each scenario has one End connector, and any number of EndNoSync connectors. EndNoSync is an End connector with an attribute “noSync” which indicate an end point that does not connect to any succeeding element, when the scenario is a refined step.
3. Sequence: Links two elements. A scenario may have more than one sequence.
4. Branch: An OR-Fork, a branch describes an alternative path, with one source step, and two or more target steps.

5. Fork: An AND-Fork, a fork describes a parallel path, with one source step, and two or more target steps.
6. Merge: An OR-Join, a merge describes alternative paths, with two or more source steps, and one target step.
7. Join: An AND-Join, a join describes a parallel path, with two or more source steps, and one target step [Pet07a].

The class diagram of PathConnection and its associations with other classes are described in figure 3-4

Figure 3- 4 PathConnection class and its associations

PathConnection class is an instance of Scenario class. It has associations with Step class, and Classifier class, with multiplicity of (0..*) and (0..1). It is also of Start, End, Sequence, Fork, Join, branch, and Merge classes.

CSM profile provides a various properties for the PathConnection class. These properties and some additional properties which will be used in developing the CSM operations are described below.

1. id: A unique identity for a connector within a scenario.
2. description: Describes the connector . This property can provides any information which is considered valuable.
3. traceabilityLink: Define a copy of UML identifier for the connector.
4. Connector.source: An association role name which Identify the preceding element that the connector is linked with. The source holds the id of the preceding element. The element can be a step, an acquired resource, or a released resource.
5. Connector.target: An association role name which Identify the succeeding element that the connector is linked with. The target holds the id of the succeeding element. The element can be a step, an acquired resource, or a released resource.
6. Connector.classifier: An association role name which identify the kind of the sent message.

In addition to the previous properties, [Woo07] are defining the following operations in algebraic form:

1. $C = \text{Connectors}(SC)$: Defines an operation which returns a set of connectors in scenario SC.
2. $c = \text{Start}(SC)$: Defines an operation which returns a unique start connector of the scenario SC
3. $c = \text{End}(SC)$: Defines an operation which returns the end connector of the scenario SC.

4. $c = \text{Seq}(\text{StepA}, \text{StepB})$: Defines an operation which returns the sequence connector between StepA and StepB.
5. $c = \text{Branch}(\text{StepA}, \text{SetOfSteps})$: Defines an operation which returns the branch connector between StepA and SetOfSteps.
6. $c = \text{Fork}(\text{StepA}, \text{SetOfSteps})$: Defines an operation which returns the fork connector between StepA and SetOfSteps.
7. $c = \text{Join}(\text{SetOfSteps}, \text{StepA})$: Defines an operation which returns the join connector between SetOfSteps and StepB.
8. $c = \text{Merge}(\text{SetOfSteps}, \text{StepA})$: Defines an operation which returns the Merge connector between SetOfSteps and StepB.

3.1.4. Definition of Resource operations

In CSM, Resource is defined as an element which executes operations. An element could be an active resource such as a device or an external operation, or a passive resource such as a thread.

GeneralResource class, shown in figure 3-5, is an instance of CSM class. It is a generalization of PassiveResource class, and ActiveResource class. PassiveResource is a generalization of Component class. ActiveResource is a generalization of a ProcessingResource class and ExternalOperation class. There is also ResourceAcquire class and ResourceRelease class to describe the acquisition and releasing of the resource. These two classes are a specialization of Step class. GeneralResource has an association with ResourceAcquire class and ResourceRelease class, with multiplicity of (0..*) for both.

Figure 3- 5 Resource classes and their associations

CSM provides various properties for the GeneralResource class, which are described below together with some additional properties used in developing the CSM operations:

1. id: A unique identity for a resource within a CSM.
2. name: A name of the resource.
3. description: Describes the resource. This property can provides any information which is considered valuable.
4. traceabilityLink: Defines a copy of UML identifier for the Resource.
5. multiplicity: Number of copies that exist of a resource.
6. schedPolicy: Policy by which resource schedules workloads
7. ResourceAcquire.acquire: The component's id that is acquired.
8. ResourceRelease.release: The component's id that is released.

9. `Component.host`: The processingResource's id of the component

In addition to the previous properties, in [Woo07] is defined the following operation in algebraic form:

`R = Resources (SC)`: Returns a set of resources in scenario SC.

3.2. Java implementation of CSM operations

This section describes the implementation of the CSM operations, mentioned in section 3.1, which were introduced in algebraic format. The section also describes the packages used in implementing these operations, including the XML parser and DOM API.

3.2.1. Package `javax.xml.parsers`

This package provides classes which allow processing of XML documents. Such parser is needed because the CSM input models are given in XML format. Two types of parsers are supported, the first one is Simple API for XML (SAX) which is used when the operation in the application is sequential. SAX parser is fast and requires a few memory. The second one is DOM parser which is used in applications where the document needs to be used repeatedly or unsequentially. DOM parser was used in the CSM aspect composition implementation because it provides efficient document navigation [Par08]. During the implementation, the following classes were used:

1. `DocumentBuilder` class: Defines the API to obtain DOM Document instances from an XML document.
2. `DocumentBuilderFactory` class: Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents

3.2.2. Package org.w3c.dom

This package provides the interfaces for the Document Object Model (DOM)

DOM is a platform and language –neutral interface which allows program and scripts to dynamically access and update the content, structure and style of documents. Based on that, the document can be processed, and the results can be incorporated back into the presented page.[W3c08].

DOM was developed by World Wide Web Consortium (W3C) in 1998. It supports features that allow navigation in any direction, and allows various manipulations and modifications for the dynamic page. The page first must be buffered and parsed before DOM can be used.

DOM is a component of the Java API for XML Processing. This API allows programs to dynamically access and update the content and structure of documents [Api08]. During the implementation, the following interfaces were used:

1. Document interface: Represents the entire XML document.
2. NodeList interface: Provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented.
3. Node interface: It is the primary datatype for the entire Document Object Model.
4. Element interface: Represents an element in XML
5. NamedNodeMap interface: Used to represent collections of nodes that can be accessed by name.
6. DOMErrorHandler interface: It is a callback interface that the DOM implementation can call when reporting errors that happens while processing XML data, or when doing some other processing.

3.2.3. Implementation of CSM operations

The algebraic CSM operations introduced in section (3.1) were implemented as methods of the following classes: Scenario, Step, PathConnection, and Resource.

Table 3-1 lists the Scenario operations, Table 3-2 the Step operations, Table 3-3 the PathConnection operations, and Table 3-4 the Resource Operations. Table 3-5 is describing the implementation of new operations used in CSM composition tool

Algebraic notation of the operation	Signature of the operation
SC = (S, C, R)	+getScenario(steps:NodeList,connector:NodeList[], resource: NodeList): Element
ReplaceStep(SC,StepA,SZ)	+replaceStep(primary:Document,stepA:Element, context: Document): void
C'=ReplaceConn(C,ConnA,StepA,StepZ)	+replaceConn(c:NodeList,connA:Element, stepA:Element,stepZ: Element) :Element[]
InsertBefore (SC, StepA, SZ)	+insertBefore(primary:Document,stepA:Element, context: Document): void
InsertAfter (SC, StepA, SZ)	+insetAfter(primary: Document, stepA: Element, context: Document): void
Expand (SC, StepA, SZ)	+expand(document:Document,stepA:Element, SZ: Element): void

Table3- 1 The algebraic notation of Scenario operations and their signatures

Algebraic notation of the operation	Signature of the operation
S = Steps(SC)	+steps(scenario: Element):NodeList
s = First(SC)	+first(scenario: Element): Element
s = Last(SC)	+last(scenario: Element): Element

•StepA	+dotStep(document: Document, stepA: Element): Element
{••StepA}	+doubleDotStep(document: Document, stepA: Element): Element[]
{•!StepA}	+dotClosureStep(document: Document, stepA: Element): Element[]
StepA•	+stepDot(document: Document, stepA: Element): Element
{StepA••}	+stepDoubleDot(document: Document, stepA: Element): Element[]
{StepA!•}	+stepDotClosure(scenario: Document, stepA: Element): Element[]

Table3- 2 The algebraic notation of Step operations and their signatures

Algebraic notation of the operation	Signature of the operation
C = Connectors (SC)	+connectors(scenario: Element): NodeList
c = Start(SC)	+start(scenario: Element): Element
c = End(SC)	+end(scenario: Element): Element
c = Seq(StepA, StepB)	+seq(stepA: Element, stepB: Element): Element
c = Branch(StepA, SetOfSteps)	+branch(stepA: Element, setOfSteps: NodeList): Element
c = Fork(StepA, SetOfSteps)	+fork(stepA:Element,setOfSteps:NodeList): Element
c = Join(SetOfSteps, StepA)	+join(setOfSteps:NodeList,stepA:Element): Element
c = Merge(SetOfSteps, StepA)	+merge(setOfSteps:NodeList,stepA:Element): Element

Table3- 3 The algebraic notation of PathConnection operations and their signatures

Algebraic notation of the operation	Signature of the operation
R = Resources (SC)	+resources(scenario: Element): NodeList

Table3- 4 The algebraic notation of Resource operations and their signatures

Operation	Description
+createSequence(scenario: Document): void	Create new sequence
+getElement(scenario: Document, nType:String, nName:String): Element	Returns the object representing an element for a given node name and node type. If the name does not exist, it will return null
+createRefinement(doc:Document): void	Create new refinement
+createStep(scenario: Document): void	Create new step
+stepComponent(scenario: Document, stepA: Element): Element	Returns the most recently acquired component for StepA. This function can be used when Step.component is undefined
-importScenario(document: Document): void	Import the context-specific aspect XML file, and append it to the primary XML file
+getGenericName():String	Returns the parametric values of the generic aspect
+createNewDocument(): void	Create new document
+openFile(filename: String): void	Opens a file for a given file name
+setParsedDocument(): void	Initial function used to parse the document
+mergeDocument(document: Document): void	Append an imported document to another document
+output(out: PrintWriter): void	Writes a document into XML file
+getRandomId(): String	Gets a random id of string type
-getRedundantElement(document: Document, id:	Returns the id of the redundant elements, if

String): Element	the id parameter is not redundant, it returns null
-getElementId(document: Document): String[]	Return a set of ids of all elements in the document
+setDocumentUniqueld(document: Document): void	Makes all the elements' id of the document unique
-makeRandom(): int	Return a random number

Table3- 5 Other operations used in the CSM composition tool defined in the thesis

Chapter 4. Aspect composition in CSM

This chapter consists of three sections which describes the steps required to perform the aspect composition in CSM. These steps are:

1. Identifying the join-points of the composition by defining a set of rules called the point-cut rules.
2. Generating the context-specific aspect model by instantiating the generic aspect model using Binding rules. The rule defines how the parametric values of the generic model should be transformed into concrete one.
3. Performing the composition between the primary model and the context-specific aspect model.

4.1. Identify join-points through point-cut expressions

The point-cut is defined as a set of rules or conditions, which applied on a flattened primary model scenario to identify the join-point steps, which in turn are the binding locations for the generic aspect [Woo07]. In this thesis, two approaches were developed related to point-cut expressions. The first approach deals with general point-cut expressions, which handles general aspect composition. The second approach deal with “shortcut” point-cut expressions that handles security aspect.

4.1.1. Defining point-cut for general aspects

Point-cut expression consists of a series of point-cut rules. Each rule is a constraint related to a step, which means that if a rule is applied on a step, the return will be either true of

false. Theoretically, point-cut rule can be defined using any property, operation, or element in CSM schema. However, this is not practical since not all the properties, operations, and elements are related to the step element. Also, because there are many of these properties are not user-friendly, and their values can not be used because they changes from one model to another.

Based on that, a modified set of point-cut rules is suggested. It has been defined as below.

Definition: The point-cut rule is defined in BNF form as follows:

```

<point-cut> ::= <expression> | <expression> || <expression> | <expression> & <expression>
<expression> ::= "step." <step_property> <operator> <value>
                | "step." <step_operation> <operator> <value>
                | "step." <step_operation> "." <element1_property> <operator> <value>
                | "step." <step_operation> "." <element1_operation> <operator> <value>
                |      "step."      <step_operation>      "."      <element1_operation>      "."
                <element2_property> <operator> <value>
                | "step." <step_operation> "." <csm_element2> "." <element2_property>
                <operator> <value>
                | "step." <csm_element1> "." <element1_property> <operator> <value>
                | "step." <csm_element1> "." <element1_operation> <element2_property>
                <operator> <value>
                | "step." <csm_element1> "." <csm_element2> "." <element2_property>
                <operator> <value>
                | "step." <csm_element1> "." <csm_element2> "." <element2_operation>
                <operator> <value>

<step_property> ::= "name" | "description" | "probability" | "hostDemand"
                | "repCount"

<step_operation> ::= "stepComponent" | "stepHost" | "NewService"

```

$\langle \text{csm_element1} \rangle ::= \text{"component"} \mid \text{"precondition"} \mid \text{"postcondition"}$
 $\langle \text{element1_property} \rangle ::= \text{"name"} \mid \text{"description"} \mid \text{"multiplicity"} \mid \text{"parent"} \mid \text{"schedPolicy"} \mid$
 "expression"
 $\langle \text{element1_operation} \rangle ::= \text{"componentHost"} \mid \text{"NewComponent"}$
 $\langle \text{csm_element2} \rangle ::= \text{"host"}$
 $\langle \text{element2_property} \rangle ::= \text{"name"} \mid \text{"description"} \mid \text{"multiplicity"} \mid \text{"opTime"} \mid \text{"schedPolicy"}$
 $\langle \text{element2_operation} \rangle ::= \text{"NewHost"}$
 $\langle \text{operator} \rangle ::= \text{"="} \mid \text{"!="} \mid \text{">"}$ $\mid \text{"<"}$ $\mid \text{">="}$ $\mid \text{"<="}$
 $\langle \text{value} \rangle ::= \langle \text{string} \rangle$
 $\langle \text{string} \rangle ::= \{ \langle \text{char} \rangle \}_0^n$

For example, the following are point-cut rules:

$\text{step.name} \neq \text{getData} \parallel \text{step, stepComponent.host.multiplicity} \geq 3.0$
 $\text{step.component.componentHost.name} = \text{ADVProcessor} \ \& \ \text{step.NewHost} = \text{true}$

Point-cut definition includes element operations. Some of these operations are another way of representing elements. As an example `stepComponent` operation returns the most recently acquired component for a step. The operation can be used when `Step.component` is undefined. Same thing applies to `stepHost` and `step.component.host`, also, `componentHost` and `component.host`.

In addition to the previous operations, there are `NewHost`, `NewComponent`, and `NewService` operations. These three operations are used to identify transitions between three levels of system elements. The first level is the Host level. The operation `NewHost`, shown in figure 4.1, identifies join-points when there are transitions from one host to another.

The algebraic form of the operation is:

$$\text{NewHost}(\text{Step}\beta) = ((\text{Step}\beta \in \text{SC}^*) \text{ AND } (\text{for all Step}\alpha \in \{\bullet \bullet \text{Step}\beta\}, \\ \text{Component}(\text{Step}\alpha).\text{host} \neq \text{Component}(\text{Step}\beta).\text{host}))$$

The implementation of NewHost is performed as follows:

For *NewHost(stepB)*:

1. Find $\{\bullet \bullet \text{Step}B\}$, which is a set of steps that is preceded *StepB*
2. For each step in the set , find its Host
3. Find the Host of *StepB*
4. if the Host of *StepB* is not equal to the Host of step, the operation output is true,
otherwise it is false

Figure 4- 1 Applying NewHost operation

The second level is the component level. The operation NewComponent, shown in figure 4.2, identifies join-points when there are transitions from one component to another. The algebraic form of the rule is:

$$\text{NewComponent}(\text{Step}\beta) = ((\text{Step}\beta \in \text{SC}^*) \text{ AND } (\text{for all Step}\alpha \in \{\bullet \bullet \text{Step}\beta\}, \\ \text{Component}(\text{Step}\alpha) \neq \text{Component}(\text{Step}\beta)))$$

The implementation of NewComponent is performed as follows:

For *NewComponent(stepB)*:

1. Find $\{\bullet\bullet StepB\}$, which is a set of steps that is preceded *StepB*
2. For each step in the set , find its Component
3. Find the Component of *StepB*
4. if the Component of *StepB* is not equal to the Component of step, the operation output is true, otherwise it is false

Figure 4- 2 Applying NewComponent operation

The third level is the service level. Service is an interface of a component, and it is associated to an operation called from the outside. In UML services are represented as interfaces, however in CSM this is not possible since interface element is not defined. In CSM, a service can be one step or a set of sequencing steps within one component. So, in order to differentiate between a service and a regular step, it has to be encapsulated within a Scenario element.

Any Scenario element encapsulating one service or more is defined using the attribute “description” of its first step. The definition is performed in order to indicate that this Scenario provides these services. The “description” attribute of the first step in the Scenario is defined in a form:

$\langle \text{description} \rangle ::= \langle \text{property_name} \rangle = \langle \text{property_value} \rangle ;$

$\langle \text{property_name} \rangle ::= \text{"service"}$

$\langle \text{property_value} \rangle ::= \langle a-z \rangle \mid \langle A-Z \rangle \mid \langle 0-9 \rangle$

As an example, the XML format for a scenario provides a checkout service:

```
<Scenario id="id2" name="Checkout">
  <Step id="id88" name="send request" description="service=checkout"
    predecessor="id87" successor="id90"/>
  ...
</Scenario/>
```

A service can be a single one, or it can be a sequence of nested services. Figure 4-3 describes nested services, where webserver component provides service1 to eb component. The webserver component will get service2 from database in order to provide service1.

Figure 4- 3Nested service

At service level, the point-cut rule identifies join-points when there are transitions from a service. The algebraic form of the operation is:

$\text{NewService}(\text{Step}\beta) = ((\text{Step}\beta \in \text{SC}^*) \text{ AND } (\text{Step}\beta.\text{description} \text{ contains "service"}))$

The implementation of NewService is performed as follows:

For *NewService(stepB)*:

1. If the description property of *StepB* contains a word “service”, the operation output is true otherwise it is null;

4.1.2. Defining point-cuts for security aspect

This section focuses on simplifying the way in which the point-cuts for a specific kind of aspects - namely security aspects, can be expressed by the designers. Security aspect is vital, and it plays an important role in different applications. TPC-W application is an example, where it is important to identify which host, component, or service requires security features. Some web pages in TPC-W are defined in the specification as secure, while others are not [Woo07].

Defining point-cuts for security aspects requires defining additional constraints on system elements. The new constraints must indicate that the system element is secured. So, if the element is a host, the new constraint must indicate that this host is a secured host. Same thing if it is a component or a service.

In order to define point-cuts for secure aspects to the host, component, and service elements, *NewHost*, *NewComponent*, and *NewService* operations will be used, as these operations identify transitions between host to host, component to component, and service to service respectively.

So at the host level, to identify join-point that assures secure messages to a secure host, the following point-cut rule is defined:

$$J = \text{any step}\alpha \mid (\text{NewHost}(\text{Step}\alpha) \text{ AND } (\text{Component}(\text{Step}\alpha).\text{Host} \in \text{SecureHosts}))$$

A join-point *J* can be identified to handle secure message for all communications involving hosts in the set *SecureHosts* defined by the user [Woo07].

The implementation of the rule is performed as follows:

For a *secureMessage2SecureHost(stepa, secure host)*:

1. If the *stepa* has a new host, find the host of *stepa*.
2. If the host of *stepa* is within the set of secured host, the operation output *stepa*, otherwise it is null

At the component level, to identify join-point that assures secure messages to a secure component, the following point-cut rule is defined:

$J = \text{any } stepa \mid (\text{NewComponent}(Step\alpha) \text{ AND } (\text{Component}(Step\alpha) \in \text{SecureServers}))$

A join-point J can be identified to handle all communications involving server components in the set SecureServers defined by the user [Woo07].

The implementation of the rule is performed as follows:

For *secureMessage2SecureServer(stepa, secureServer)*:

1. If the *stepa* has a new component, find the component of *stepa*.
2. If the component of *stepa* is within the set of secured component, the operation output *stepa*, otherwise it is null

At the service level, to identify join-point that assures secure messages to or from a secure service, the following point-cut rule is defined:

$J = \text{any } stepa \mid (\text{NewService}(Step\alpha) \text{ AND } (Step\alpha \in \text{SecureServices}))$

A join-point J can be identified to handle security for all communications involve services in the set of SceureServices.

The implementation of the rule is performed as follows

For *secureMessage2OrFromSecureService(stepa, secureService)*:

1. If *stepa* is a service, and
2. If *stepa* is within the set of the secured services, the operation output *stepa.parent*, otherwise it is null.

4.2. From generic to context-specific aspect models

Translating generic aspect model to context-specific aspect model is the second step in the CSM composition. A generic aspect model describes the general structure and behavior of the aspect model. It provides the solution which is proposed by the aspect in a general way, not related to any specific primary model. Generic aspect model can be instantiated multiple times to produce multiple context-specific aspect models based on different binding rules [Fra04], [Pet07b].

Instantiating generic aspect model to context-specific aspect model depends on the join-points, which are the points where the aspect will be inserted in the primary model.

Generic aspect model can be described in a form of list of element's attribute values. These values can be either described as a template form “|value” or as an expression.

$GenericA_i(Roles, parms)$

$Roles = \{|value_1, |value_2, ..., |value_n\}, \{exp_1, exp_2, ..., exp_n\}$

Primary model describes a list of Role bindings $RoleBinding(JS)$, which is defined when the join-point in the primary model is selected

Applying the role bindings defined through the primary model on the generic aspect, produces the context-specific aspect model.

$specificA_i(parms) = GenericA_i(RoleBindin gs(JS), parms)$

There are two types of generic aspect roles, dynamic and static. Dynamic are the roles which bind with the primary's element attribute values, and their values depends on the join-point. Static roles are independent on the join-point, and are defined by the user.

As an example, figure 4-4a, 4-4b describes the steps of transforming the generic aspect into the context-specific aspect.

generic ({|sender, |receiver, |senThread}, parms).

RoleBindings = (client, server, extOperation)

Figure 4- 4 (a) Primary model: TestCase09, (b) Generic model: generic

To insert a context-specific aspect “call” at the join-point, |sender is bound to client, and |receiver to server. This is because client sends the call, and server receives the call. The role |senThread is a static role, and the user defines the binding role. The context-specific aspect will be in the form of:

call = *generic* ({client, server, extOperation}, parms)

Fig 4-5 shows the context-specific aspect.

Figure 4- 5Example of context-specific aspect model

4.3. Composition

The composition is the last step to be carried out, after identifying the join-point and creating its context-specific model. The composition step requires inserting the context-specific model in the primary model at the join-point location. CSM composition tool defines three types of insertion rules. These rules are:

1. Inserting before the join-point

InsertBefore(SC, JS, SpecificA_{ij})

2. Inserting after the join-point

InsertAfter(SC, JS, SpecificA_{ij})

3. Replacing the join-point by the context specific model:

Re placeStep(SC, JS, SpecificA_{ij})

where SC is the scenario of the primary model, JS is the join-point step, and SpecificA is the advice or the sub-scenario of context-specific.

It is important to mention that the tool keeps in its data structure two version of the initial primary model: a structured one (containing nested sub-scenarios), and a flattened one used for navigation. The join points are identified on the flattened scenario, and the insertion of aspects is done on the structured version. This approach was taken to simplify the navigation and at the same time to produce a structured composed model.

4.3.1. InsertBefore rule

InsertBefore rule is selected when the context-specific aspect is inserted before the join-point step JS. The implementation of InsetBefore rule, shown in figure 4.6, is performed as follows:

1. Create random variables.
2. Check the uniqueness of id attribute of every element in the primary model and the context-specific model. If there are redundant ids, eliminate the redundancy by changing the redundant values with the random variables.
3. Identify the join-point step (JPStep), and find its predecessor, and successor.
4. Create a new refined step element (newStep, and newRefinement child), with its successor sequence (newSeq).
5. Insert the new elements before JPStep, and modify the following attributes:

newStep.id = JPStep.id

newStep.predecessor = JPStep.predecessor

newStep.successor = firstRandomNumber

newRefinement.parent = newStep.id

```

newSeq.id = firstRandomNumber

newSeq.source = JPStep.id

newSeq.target = secondRandomNumber

JPStep.id = secondRandomNumber

JPStep.predecessor = firstRandomNumber

(JPStep•).source = secondRandomNumber

```

6. Append the scenario of the context-specific in the primary document.

Figure 4- 6 Insert before rule

4.3.2. InsertAfter rule

InsertAfter rule is selected when the context-specific aspect is inserted after the join-point step JS. Implementing InsertAfter rule requires indirect procedure, since Java API does not provide an insertAfter method. This is not the case with insertBefore rule, where insertBefore method is available. So, in order to implement insertAfter Rule, the insertion is done before the next element after the join-point. The next element could be a step, a resource acquire, or a resource release. Figure 4.7 describe the three cases.

Figure 4- 7 The options of implementing insertAfter rule

Based on these options, the implementation of insertAfter rule is performed as follows:

1. If StepB is succeeding JPStep, insertBefore(StepB)
2. if ResourceAcquire is succeeding JPStep, insertBeforeResource(ResourceAceuire)
3. if ResourceRelease is succeeding JPStep, insertBeforeResource(ResourceResource)

The implementation of insertAfter rule requires a new insertBeforeResource rule.

The implementation of the new rule is performed as follows:

1. Create two random ids for the new inserted elements.

2. Check the uniqueness of id attribute of every element in the primary model and the context-specific model. If there are redundant ids, eliminate the redundancy by using random values.
3. Identify the ResourceAcquire/Release and find its successor.
4. Create a new refined step element (newStep, and newRefinement child), with its successor sequence (newSeq).
5. Insert the new elements before ResourceAcquire/Release, and modify the following attributes:

newStep.id = ResourceAcquire/Release.id

newStep.predecessor = ResourceAcquire/Release.predecessor

newStep.successor = firstRandomNumber

newRefinement.parent = newStep.id

newSeq.id = firstRandomNumber

newSeq.source = ResourceAcquire/Release.id

newSeq.target = secondRandomNumber

ResourceAcquire/Release.id = secondRandomNumber

ResourceAcquire/Release.predecessor = firstRandomNumber

ResourceAcquire/Release.seq.source = secondRandomNumber

6. Append the scenario of the context-specific in the primary document.

Fig 4-8 describes the insertion operation.

Figure 4- 8 Insert After rule

4.3.3. ReplaceStep rule

ReplaceStep rule is used when the join-point is replaced by the context-specific aspect model. The implementation of ReplaceStep rule, shown in figure 4.9, is performed as follows:

1. Check the uniqueness of id attribute of every element in the primary model and the context-specific model. If there are redundant ids, eliminate the redundancy by modifying the redundant values with the random variables.
2. Identify the join-point step (JPStep), and find its successor.
3. Create a new refinement (newRefinement) to the JPStep.
4. Modify the attributes of JPStep, such as:

JPStep.name = contextSpecific.scenario.name

newRefinement.parent = JPStep.id

newRefinement.sub = contextSpecific.scenario.id

For undefined attributes, set the values to default.

5. Append the scenario of the context-specific in the primary document.

Figure 4- 9 Replace step rule

Chapter 5. Aspect Composition Tool

The first section of this chapter describes the UML design of the aspect composition plug-in tool. The second section describes the design of the GUI tool, and how the GUI is operating.

5.1. UML Design

This section presents the use case diagram for the tool, as well as its structural view represented as package and class diagrams.

5.1.1. The Use case Diagram

Aspect Composition Tool consists of fourteen use cases, shown in figure 5-1. The user can open a file (representing a generic aspect model or a primary model), compose an aspect (using insert before, insert after, and replace rules), expand a model (i.e., flatten it), and save a model to a file. The “Compose aspect” use case includes “bind generic” (which generates a context-specific aspect model from a generic one), and “identify join point” by applying the corresponding point-cut rules. The “Identify join point” use case includes “Define general point-cuts”, and extends “Define security point-cuts”. The complete textual description of the use cases are given in appendix B

Figure 5- 1 Use case diagram of Aspect Composition Tool

5.1.2. The Package diagram

The aspect composition tool contains three different packages: the viewer package, the CSM package, and the CSMLLogic package. Figure 5-2 shows the three packages and their relations.

5.1.2.1. The Viewer Package

The viewer package contains three packages which handle the GUI properties and operations. These packages were developed by [Isr05] to manage the transformation between the Sequence diagram, and the CSM diagram in GUI. The packages of viewer are:

Figure 5- 2 Package diagram of the aspect composition tool

1. “actions” package (Figure 5-3): contains a class which implements workbench action delegate. The workbench will create an action proxy, which will be shown in the UI. When the user tries to use the action, this delegate will be created and execution will be delegated to it [Cla06].

Figure 5- 3 “actions” package

2. “diagram” package (Figure 5-4): contains thirteen classes that handle eclipse Plug-in’s UI based on Standard Widget Toolkit (SWT). SWT is a well-designed native UI library for Java, which provides the foundation for the entire eclipse UI.

Figure 5- 4 “diagram” package

During the aspect composition design, the diagram package has been modified with new classes in order handle the CSM composition.

3. “figures” package (Figure 5-5): It contains eight classes that handle the configuration of the scenarios, steps, and pathconnections figures. The classes convert the elements in CSM model into graphical shapes, such as rectangles, and arrows.

Figure 5- 5 “figures” package

5.1.2.2. The CSMLLogic package

The CSMLLogic contains three packages. They handle the CSM composition operations. Some of these operations were described in Chapter 3, and 4. The packages of CSMLLogic are:

1. “Document” package (Figure 5-6): It contains one class which handles transferring CSM models from XML document into DOM object tree. Transferring operation requires methods that build the DOM document, initiate the parser, and parse the XML document.

Figure 5- 6 “Document” package

2. “Element” package(Figure 5-7): contains six classes which handle the CSM operations presented in chapters 3, and 4. These operations plus some additional operations were developed to allow CSM composition.

Figure 5- 7 “Element” package

3. “Util” package (Figure 5-8): It contains eight classes, which provide the tool with additional methods. These methods are used in different operations such as handling the complete procedure of CSM operations, handling the uniqueness of some elements’ attributes, handling the rules of point-cuts, and transferring the DOM object tree to XML document.

Figure 5- 8 “Util” package

5.1.2.3. The CSM Package

CSM package contains two packages. They were generated using Eclipse Modeling Framework (EMF) based on CSM schema. EMF is a java framework and code generation facility for building applications based on a structured model. Using CSM schema as input, EMF plug-in tool can generate a java code which represents the CSM structure. This package was developed by [Isr05] to handle the transformation operation.

5.1.3. The Class Diagram

The section describes the classes which were developed in the aspect composition tool. The section will present only the classes of CSMLLogic package, since they are the classes which are used to build the logic of the aspect composition tool. Figure 5-9 describes the class diagram of the aspect composition tool.

Class diagram shows fifteen classes, with their associations. The figure shows that Scenario, Step, Resource, Connector, and Refinement class are the core of the class diagram, and they are highly associated. These classes are defined within the CSM structure.

5.1.3.1. CSMElement class

It is a super class for five classes (Figure 5-10). It has four properties and two methods, described as follows:

1. newStep: It is a protected property of Element type. It defines the new step that will be inserted in the primary model
2. newRefinement: It is a protected property of Element type. It defines the new refinement of the new step.
3. newSequence: It is a protected property of Element type. It defines the new successor sequence that will be linked to the new step.
4. id: It is a private attribute of IDHandler type. It defines a role name of an association between CSMElement class and IDHandler class. It is used to define a random id for the new elements that will be created.
5. CSMElement(): It is a default constructor.

Figure 5- 9 Class diagram of Aspect Composition Tool

6. `getElement(doc:Document, elementType:String, name:String)`: It is a public method. It returns an element object of a specified element type, and element name. This method applied only to elements which have a name attribute such as step, component, and processing resource.

Figure 5- 10 CSMElement class

5.1.3.2. Scenario class

It is a sub-class of CSMElement class (figure 5-11). It has four properties and six methods, described as follows:

1. step: It is a private property of Step type. It defines a role name of an association between Scenario class and Step class.
2. resource: It is a private property of Resource type. It defines a role name of an association between Scenario class and Resource class.
3. refinement: It is a private property of Refinement type. It defines a role name of an association between Scenario class and Refinement class.
4. connect: It is a private property of Connector type. It defines a role name of an association between Scenario class and Connector class.
5. Scenario(): It is a default constructor.
6. createScenario(step:Element, refinement:Element): It is a public method. It create a sub-scenario for a given step, and refinement.

The rest of the methods are described in details in sections 3.1, 3.2, and 4.3

Figure 5- 11 Scenario Class

5.1.3.3. Step class

It is also a sub-class of CSMElement class (figure 5-12). It has six properties, and twelve methods, described as follows:

1. **id:** It is a private property of type String. It defines the id attribute of the step element.
2. **hostDemand:** It is a private property of String type. It defines the hostdemand attribute of the step element.
3. **predecessor:** It is a private property of String type. It defines the predecessor attribute of the step element.
4. **successor:** It is a private property of String type. It defines the successor attribute of the step element.
5. **probability:** It is a private property of String type. It defines the probability attribute of the step element.

6. connect: It is a private property of Connector type. It defines a role name of an association between Scenario class and Connector class.
7. Step(): It is a default constructor.
8. createStep(pDoc:Document, aDoc:Document): It is a public method. It creates a step for a given document.
9. StepComponent(doc:Document, StepA:Element): It is a public method. It returns the last component acquired StepA.

The rest of the methods are described in details in sections 3.1, 3.2.

Figure 5- 12 Step Class

5.1.3.4. Connector class

It is also a sub-class of CSMElement class (figure 5-13). It has four properties, and ten methods, described as follows:

1. id: It is a private property of String type. It defines the id attribute of a connector element.
2. source: It is a private property of String type. It defines the source attribute of a connector element.
3. target: It is a private property of String type. It defines the target attribute of a connector element.
4. connectorType: It is a private array of String type. This array defines the types of connector.
5. Connector(): It is a default constructor.
6. createSequence(pDoc:Document): It is a public method. It creates a sequence for a primary document.

The rest of the methods are described in details in sections 3.1, and 3.2.

Figure 5- 13 Connector Class

5.1.3.5. Resource class

It is also a sub-class of CSMElement class (figure 5-14). It has four properties and four methods, described as follows:

1. scenario: It is a private property of Scenario type. It defines a role name of an association between Resource class and Scenario class.
2. step: It is a private property of Step type. It defines a role name of an association between Resource class and Step class.
3. connect: It is a private property of Connector type. It defines a role name of an association between Resource class and Connector class.
4. refinement: It is a private property of Refinement type. It defines a role name of an association between Resource class and Refinement class.
5. Resource(): It is a default constructor
6. Resources(doc: Document): It is described in sections 3.1.4 and 3.2.3.4
7. InsertBeforeResource(primary: Document, elementName: String, joinPoint: Element, contextDoc: Document): It is a public method. It makes the insertion before a specified ResourceAcquire or ResourceRelease.
8. InsertAfterResource(primary: Document, elementName: String, joinPoint: String, contextDoc: Document): It is a public method. It makes the insertion after a specified ResourceAcquire or ResourceRelease.

Figure 5- 14 Resource class

5.1.3.6. Refinement class

It is the last sub-class of CSMElement class (figure 5-15). It has one property, and two method, described as follows:

1. Parent: It is a private property of String type. It defines the parent attribute of the refinement element.
2. Refinement(): It is a default constructor.
3. createRefinement(pDoc: Document, conDoc: Document): It is a public method. It creates a new refinement for a defined primary, and context-specific documents.

Figure 5- 15 Refinement class

5.1.3.7. CSMDocument class

It has four properties, and eight methods, described as follows (figure 5-16):

1. **factory:** It is a protected property of `DocumentBuilderFactory` type.

`DocumentBuilderFactory` is a class in `javax.xml.parsers` package that defines a factory API which enables applications to obtain a parser that produces DOM object trees from XML documents [Dbf08].
2. **builder:** It is a protected property of `DocumentBuilder` type. `DocumentBuilder` is a class in `javax.xml.parsers` package, which Defines the API to obtain DOM Document instances from an XML document [Dob08].
3. **document:** It is a protected property of `Document` type. `Document` is an interface in `org.w3c.dom` package, which represents the entire XML document.
4. **file:** It is a protected property of `File` type. It is an abstract representation of the CSM file.
5. **CSMDocument():** It is a default constructor. The constructor instantiate a new instance of `DocumentBuilderFactory`, which is used to create a new instance of document builder.
6. **createNewDocument():** It is a public method. It obtains a new instance of a blank document object.
7. **openFile(filename: String):** It is a public method. It obtains a new instance of a `File` abject using a file path name.
8. **setParsedDocumnet():** It is a public method. It parses the content of the CSM file to a new DOM document object using the document builder instance that was created.
9. **mergeDoc(doc: Document):** It is a public method. It imports a certain document, to another document. In aspect composition case, it imports

context-specific aspect document to primary document. Then it rearranges the node based on the XML format, such that it appends the imported scenarios and their children nodes under the existed ones, and appends the imported components under the existed ones, and appends the imported processing resources under the existed ones.

10. `output(out: PrintWriter)`: It is a public method. It instantiates an object of type `XMLDocumentWriter`, then it writes the DOM document object to a text-output stream.
11. `expandDoc()`: It is public method. It flattens CSM document, by removing all the sub-scenarios and refined steps from the top-level scenario, and replacing the refined steps with the children nodes of the deleted sub-scenarios.
12. `getParametricNames()`: It is a public method. It returns all the parametric values in the generic model.

Figure 5- 16 CSMDocument class

5.1.3.8. CSMUtil class

It has six properties, and six methods, described as follows (figure 5-17):

1. csmDoc: It is a protected property of CSMDocument type. It defines a role name of an association between CSMUtil class and CSMDocument class.
2. outfile: It is a protected property of File type. It defines the out put file of the DOM document tree.
3. step: It is a private property of Step type. It defines a role name of an association between CSMUtil class and Step class.
4. sequence: It is a private property of Connector type. It defines a role name of an association between CSMUtil class and Connector class.
5. scenario: It is a private property of Scenario type. It defines a role name of an association between CSMUtil class and Scenario class
6. refinement: It is a private property of Refinement type. It defines a role name of an association between CSMUtil class and Refinement class.
7. CSMUtil(): It is a default constructor.
8. initDocument(file String): It is a public method. The method operates two functions, first it opens a file, and then it sets the parser.
9. createElement(docComposed:CSMDocument, docContext: CSMDocument): It is a public method. The method operates four functions, it create a step, a refinement, a scenario, and a sequence.
10. InsertBefore(primary: CSMDocument, joinPoint:Element, specific: Document): It is a public method. The method operates two functions, first it inserts a step before the join-point element, and then it merges the context-specific model with the primary model.

11. InsertAfter(primary: CSMDocument, joinPoint: Element, specific: Document): It is a public method. The method operates two functions, first it inserts a step after the join-point element, and then it merges the context-specific model with the primary model.
12. ReplaceStep(primary: CSMDocument, joinPoint: Element, specific: Document): It is a public method. The method operates two functions, first it replaces the join point with a refined step, and then it merges the context-specific model with the primary model

Figure 5- 17 CSMUtil class

5.1.3.9. GeneralPointcut class

It handles the methods that analyze the point-cut expressions. The class has two properties, and six methods, described as follows (figure 5-18):

1. hashTree: It is a private property of HashMap type. The property contains the general pointcut rules which are allowed to be typed by the user.
2. step: It is a private property of Step type. It defines a role name of an association between GeneralPointcut class and Step class.

3. `GeneralPointcut()`: It is a default constructor.
4. `selectOperation(pcCondition: String, val: String, operator: String)`: it is a public method. The method checks every pointcut rule within the expression.
5. `compareString(stepCondition: String, value: String, prelim: String )`: It is a private method. The method compares two string values for a given operator.
6. `compareInteger(stepCondition:String, value:String, prelim:String)`: It is a private method. The method compares two Integer values for a given operator.
7. `compareDouble(stepCondition: String, value: String, prelim: String)`: It a private method. The method compares two double values for a given operator.
8. `setConfiguration(doc: Document, step: Element)`: It is a public method. The method set the values of the properties and elements that are related to a step.

GeneralPointcut
- hashTree:HashMap<Integer,String> - step:Step
+ GeneralPointcut() + selectOperation (pcCondition:String,val:String , operator:String):boolean - compareString(stepCondition :String,value:String, prelim:String):boolean - compareInteger(stepCondition :String,value:String, prelim:String):boolean - compareDouble(stepCondition :String,value:String, prelim:String):boolean + setConfiguration (doc:Document, step:Element):void

Figure 5- 18 GeneralPointcut class

5.1.3.10. IDHandler class

It handles the redundancy and uniqueness of id property of all elements in CSM model. “id” redundancy occurs during the creation of a new elements in the model, or during the inserting and merging operations between the primary model and the context-specific

aspect model. This class handles all these matters and guarantees the uniqueness of elements' ids.

The class has one property, and six methods, described as follows (figure 5-19):

1. rd: It is a private property of Integer type. The property is used to assign a unique id for an element.
2. IDHandler(): It is a default constructor. It instantiates makeRandom method.
3. getRandomId(): It is a public method. The method creates a random id in String type.
4. getRedundantElement(doc: Document, out: String): It is a private method. The method gets the elements which have redundant ids.
5. getElementsId(doc: Document): It is a private method. The method returns all ids' attributes in a document.
6. setDocumentUniqueId(pDoc: Document, contextDoc: Document): It is a public method. The method sets a unique id for every element in the primary model and context-specific aspect model.
7. makeRandom(): It is a private method. The method creates a random int number

Figure 5- 19 IDHandler class

5.1.3.11. PointcutParser class

It parses the point-cut expression into sets of string operations, values, and operators. The class has one inner class, two properties, and five methods, described as follows (figure 5-20):

1. Pointcut class: It is a public inner class. The class will be described below.
2. output: It is a private property of HashMap type. The property contains the parsed strings of point-cut expression.
3. orAndPrelim: It is a private property of Vector type. The property contains the 'And', and the 'OR' operators of the expression.
4. PointcutParser(): It is a default constructor.
5. andOrParser(pointcutString: String): It is a public method that parses the AND (&), and the OR (||). It is the first step of parsing operators.
6. operatorParser(prelim: HashMap<Integer,Pointcut>): It is a public method. The method parses “ = ”, “ != ”, “ > ”, “ < ”, “ >= ”, and “ <= ” operators. It is the second step of parsing operators
7. getOrAndPrelim(pointcutString: String): It is a private method. The method arrange the priority of “ & ”, and “ || ” operators.
8. getJoinPoint(output: HashMap<Integer,Pointcut>): It is a public method. The method applies the pointcut expression on a step, and finds it is a join-point or not.

Figure 5- 20 PointcutParser class

5.1.3.12. Pointcut class

It is an inner class. It contains a set of properties that are used in parsing. The class has six properties, and two methods, described as follows (figure 5-21):

1. tokenString: It is a private property of String type. The property contains the String part of the point-cut rules.
2. valueString: It is a private property of String type. The property contains the value part of the point-cut rules.
3. operator: It is a private property of String type. The property contains the operator part of the point-cut rules.
4. tokenBool: It is a private property of Boolean type . The property contains the boolean result of point-cut rules.
5. orAnd: It is a private property of String type. The property contains the And/OR operator in prioritized way.
6. stTokenizer: It is a private property of StringTokenizer type. The property contains the part of the point-cut expression that will be parsed next.

7. Pointcut(tokenString: String, valueString: String, operator: String, tokenBool: boolean, orAnd: String): It is constructor that assign a defined values to the properties.
8. Pointcut(pc:Pointcut): It is a constructor that copy an object of Pointcut type.

Figure 5- 21 Pointcut class

5.1.3.13. SecurityPointcuts class

It defines the methods used to operate the point-cut rules. The class has two properties, and ten methods, described as follows (figure 5-22):

1. document: It is a protected property of Documents type.
2. step: It is protected property of Step type. It defines a role name of an association between SecurityPointcuts class and Step class.
3. SecurityPointcuts (document: Document): It is a constructor which initiates the document property
4. NewHost(StepB: Element): it is a private method. The method checks if StepB is preceded by another step from a different host.
5. OldHost(StepB: Element): it is a private method. The method checks if StepB is succeeded by another step from a different host.

6. NewComponent(StepB: Element): it is a private method. The method checks if StepB is preceded by another step from a different component.
7. OldComponent(StepB: Element): it is a private method. The method checks if StepB is succeeded by another step from a different component.
8. NewComponent(StepB: Element): it is a private method. The method checks if StepB is preceded by another step from a different component.
9. NewService(StepB: Element): it is a private method. The method checks if StepB is a new service, by checking its description attribute.
10. SecureMessage2SecureHost(StepAlpha: Element, SecureHost: Vector): It is a public method. The method returns StepAlpha if it satisfy NewHost method, and its host belongs to SecureHost list.
11. SecureMessage2SecureServerComponent(StepAlpha: Element, SecureServer: Vector): It is a public method. The method returns StepAlpha if it satisfy NewComponent method, and its host belongs to SecureCompnent list.
12. SecureMessage2OrFromSecureService(StepAlpha: Element, SecureService: Vector): It is a public method. The method returns StepAlpha if it belongs to SecureService and its description is defined as service.
13. isService(step: Element): It is a private method. The method checks if step is a service.

Figure 5- 22 SecurityPointcut class

5.1.3.14. BindUtil class

It handles the binding operation to produce the context-specific aspect model. . The class has three properties, and two methods, described as follows (figure 5-23):

1. step: It is private property of Step type. It defines a role name of an association between BindUtil class and Step class.
2. host: It is a private property of Vector type. The property contains the ProcessingResource elements of the primary model that will binds to the parametric values of the generic aspect model.
3. component: It is a private property of Vector type. The property contains the component elements of the primary model that will binds to the parametric values of the generic aspect model.
4. BindUtil(): It is a default constructor.
5. bindsTo(doc: Document, StepAlpha: Element): It is a public method. The method binds the parametric values of the generic aspect model with the dynamic Role Binding of primary model.

BindUtil
-step:Step -host:Vector<Element> -component:Vector<Element>
+ BindUtil() + bindsTo(doc:Document, StepAlphaElement):Vector<Element>

Figure 5- 23 BindUtil class

5.1.3.15. XMLDocumentWriter class

It outputs the DOM object tree to the output stream. The class has one property, and three methods, described as follows (figure 5-24):

1. out: It is a private property of PrintWriter type. PrintWriter class prints formatted representations of objects to a text-output stream.
2. XMLDocumentWriter(out: PrintWriter): It is a constructor which initiates the out property.
3. close(): It is a public method. The method closes the output stream.
4. write(node: Node, indent: String): It is a public method. The method Output the specified DOM Node object, printing it using the specified indentation string.

XMLDocumentWriter
- out:PrintWriter
+ XMLDocumentWriter(out:PrintWriter) + close(): void + write(Node node:Node, indent:String):void

Figure 5- 24 XMLDocumentWriter

5.2. GUI Design

The tool was designed to operate as a plug-in under eclipse 3.2.1 environment. The tool depends on the following libraries:

1. Eclipse Modeling Framework (EMF2.2)
2. Graphical Editing Framework (GEF 3.2.2)
3. CSM - one of the tool packages.

5.2.1. Design the operations of the GUI

In order to run the aspect composition tool, a new configuration must be created to launch an eclipse application. After running the new configuration, a new eclipse window will be launched. On the menu bar of the new window, a 'CSM Viewer' menu will be enabled.

Figure 5-25 describes the new window.

Figure 5- 25 CSM Viewer menu

Selecting 'CSM Viewer' menu produces a new widow described in figure 5-26

Figure 5- 26 CSM Viewer window

CSM viewer menu bar contains four menus, and one hidden menu.

1. File menu: It contains two Items 'Open', and 'Exit'. The Open item is a submenu for two items 'Primary', and 'Generic'.
2. Bind menu: It contains one item 'Bind'.
3. Compose menu: It contains two items 'General Aspect', and 'Security Aspect'.
4. Expand menu: It contains one item 'Expand'.
5. Save menu: It is a hidden menu. It appears when a model is opened. It contain one item 'Save'.

Within these menus, the tool provides seven menu items, described in figure 5-27.

1. Primary item: It opens the primary file.
2. Generic item: It opens the generic file.
3. Bind item: It allows the binding of the generic model to the context specific aspect model. This item can be activated under two conditions. First the generic file must be opened. Second the parametric values of the generic model must be all assigned by the user. Since at this stage the join-points are not identified, dynamic values can not be defined by the tool. So, if the user did not define the dynamic values, they will be left as parametric values.
4. General Aspect item: It allows the user to perform the aspect composition using general definition of point-cut rules.
5. Security Aspect item: It allows the user to perform security aspect composition using security definition of point-cut rules.
6. Expand item: It flattens the viewed model.
7. Save item: It is a hidden item. It saves the viewed model.

Figure 5- 27 menu items of the aspect composition tool

The aspect composition tool allows the developers to perform four major operations.

These operations are:

1. Compose using General Aspect.
2. Compose using Security aspect.
3. Expand operation.

5.2.1.1. Performing Compose using General aspect

The following steps describe the composition procedure based on general aspect.

1. Select File > Open > Primary command to open the primary file dialog, described in figure 5-28.

Figure 5- 28 Open Primary File dialog

2. Choose the primary file from the dialog, and then confirm selection. The selected file will be viewed. See figure 5-29

Figure 5- 29 Viewing primary model

3. Select Compose > General Aspect command to open Point-cut Definition window, described in figure 5-30. Type the required constraints using the attributes, elements, and methods which are related to the step element. Select Help to find all available constraints. The tool provides a set of step variable names, and a set of operators described in chapter 3, and 4.

Figure 5- 30 Point-cut definition window

5. After typing the point-cut rule, and confirming, a new Aspect composition window will appear. See figure 5-31

Figure 5- 31 Aspect Composition window

6. Select Define Parametric values button to open generic file dialog, described in figure 5-32.

Figure 5- 32 Open Generic File dialog

7. Choose the generic file from the dialog, and then confirm selection. The CSM Binding window appears, as described in figure 5-33.

Figure 5- 33 CSM Binding window

8. CSM Binding window contains three columns, the Parametric values, the Static check boxes, and the 'Binds to' text boxes. Check the boxes for the values which will be defined by the user.

Choose Refresh button to enable to the text for the values which have been checked. Type the concrete values for the checked parametric values. The rest are dynamic values, and they are defined by the tool based on the identified join-points.

9. Confirm the selection of the CSM Binding opens the Aspect composition window, as described in figure 5-34.

Figure 5-34 Aspect Composition window

10. Aspect composition window contains the join-point which complied with the point-cut rule, the type of the composition, and compose check box to give the user the choice to accept the composition of the identified join-point or not. The user selects the composition types, and the compose check boxes.
11. Choose Done bottom to complete the composition operation. The output is viewed in figure 5-35.

Figure 5-35 The output of the composition

5.2.1.2. Performing Compose using Security aspect

This section describes special short-cuts for security aspects, where the user indicates the intention to secure certain hosts, servers, and services rather than using general point-cut expressions. The following steps describe the composition procedure based on security aspect.

1. Select File > Open > Primary command to open the primary file dialog, described in figure 5-28.
2. Choose the primary file from the dialog, and then confirm selection. The selected file will be viewed. See figure 5-29
3. Select Compose > Security Aspect command to open Securing level window, described in figure 5-36. The Securing levels window defines three levels Securing Hosts, securing Servers, and Securing Services. The user chooses the levels he wants to secure.

Figure 5- 36 Securing levels window

4. Select >> to open the Securing Host window as described in figure 5-37

Figure 5- 37 Securing Host window

5. Select Define Parametric values button to open generic file dialog, described in figure 5-32.
6. Choose the generic file from the dialog, and then confirm selection. The CSM Binding window appears, as described in figure 5-33.
7. CSM Binding window contains three columns, the Parametric values, the Static check boxes, and the 'Binds to' text boxes. Check the boxes for the values which will be defined by the user.

Choose Refresh button to enable to the text for the values which have been checked. Type the concrete values for the checked parametric values. The rest are dynamic values, and they are defined by the tool based on the identified join-points.
8. Confirm the selection of the CSM Binding returns back to the Securing Host window, as described in figure 5-37.
9. Select Get Secure Hosts button to open Selecting Secured Hosts window, described in figure 5-38. Select the secured hosts.

Figure 5- 38 Selecting Secured Hosts window

10. Confirm the selection opens Securing Host window as described in figure 5-39. The window contains three columns; 'Host', 'To', and 'From'. The host column defines the selected secured Hosts. The 'To', and 'From' columns are a pair of join-point which defines the incoming, and outgoing message of the host. The composition requires securing all the incoming, and outgoing message of the secured hosts.

Figure 5- 39 Securing Host window

11. Select >> to open the Securing Server window as described in figure 5-40

Figure 5- 40 Securing Server window

12. Repeat steps 5, 6, 7, and 8.
13. Select Get Secure Servers button to open Selecting Secured Servers window, described in figure 5-41. Select the secured Servers.

Figure 5- 41 Selecting Secured Servers

14. Confirm the selection opens Securing Server window as described in figure 5-42.

The window contains three columns; 'Server', 'To', and 'From'. The Server column defines the selected secured Servers. The 'To', and 'From' columns are a pair of join-point which defines the incoming, and outgoing message of the Server. The composition requires securing all the incoming, and outgoing message of the secured Servers.

Figure 5- 42 Securing Server window

15. Select >> to open the Securing Service window as described in figure 5-43

Figure 5- 43 Securing service Window

16. Repeat steps 5, 6, 7, and 8.
17. Select Get Secure Services button to open Selecting Secured Services window, described in figure 5-44. Select the secured Services.

Figure 5- 44 Selecting Secured Services window

18. Confirm the selection opens Securing Service window as described in figure 5-45.

Figure 5- 45 Securing Service window

12. Choose Done button to complete the composition operation.

5.2.1.3. Performing Expand operation

Expand operation is flattening a CSM model; it is performed as an internal operation during the aspect composition operation to simplify the navigation process. It also can be performed as a separate operation to expand any CSM model. The following steps describe the Expand operation.

1. To expand a primary model, Select File > Open > Primary command to open the primary file dialog, described in figure 5-28.
2. Choose the primary file from the dialog, and then confirm selection. The selected file will be viewed. See figure 5-46

Figure 5- 46 Viewing CSM model

3. Select Expand> Expand to perform the expansion. Figure 5-47 describes the output of the expansion

Figure 5- 47 The output of the expansion operation

Chapter 6. Verification and testing

This chapter consists of two sections. The first section uses a set of test cases containing 72 samples of primary model, and a sample of generic model. These samples were developed using JUCMNav [Juc08]. The second section uses a case study which represents the Transactional Processing Web e-Commerce benchmark (TPC-W) [Tpc08], where the CSM model was generated with PUMA transformation tools from UMLmodels annotated with SPT.

Aspect Composition tool operates two main procedures: General aspect composition and Security aspect composition. Testing these procedures will verify that all tool's functions are tested.

The General aspect composition procedure contains the following functions:

1. Expanding primary model.
2. Parsing point-cut expression.
3. Defining point-cut rules.
4. Identifying join-point.
5. Binding Generic model.
 - a. Identifying parametric values.
 - b. Defining static values.
 - c. Binding static values.
 - d. Binding dynamic values.
6. Performing composition.
 - a. Insert before.
 - b. Insert after.
 - c. Replace.

7. Saving composed model.

The Security aspect composition procedure contains the following functions:

1. Expanding primary model.
2. Defining secure point-cut rules
 - a. Secure host
 - b. Secure component
 - c. Secure service
3. Identifying join-point
4. Binding Generic model.
 - a. Identifying parametric values.
 - b. Defining static values.
 - c. Binding static values.
 - d. Binding dynamic values.
5. Performing composition.
 - a. Insert before.
 - b. Insert after.
6. Saving composed model.

The methodology used in testing is the black box testing. In this methodology, input sets of variables and actions are applied to produce expected outputs. The testing will be verified if the output is similar to what is expected. Table 6-1 describes the template used in black box testing.

Variant	Input section		Output section	
	Input variables	Input actions	Expected results	Expected output actions

Table 6- 1 Black box testing template

The approach used to select test cases is based on category-partition method, which provides a systematic way of decomposing a functional specification into test specifications for individual operation [Chi94]. The test cases were selected by considering different alternatives of point-cut rules according to the BNF grammar, described in section 4.1.1, in a way that eliminate the point-cut expression redundancy.

6.1. Simple test cases

The test cases map, described in appendix D, used in this section consists of 72 samples of primary model. These samples were developed using fifteen elements; StepA, StepB, StepC, RefineA, RefineB, RefineC, ResourceA1, ResourceA2, ResourceA3, Comp1, Comp2, Comp3, Host1, Host2, Host3. On each sample, the elements have been rearranged in order to represent a unique primary model. The generic model is represented by one sample, which is enough to be used with all primary model samples. A limitation of this set of test cases is that the primary model is flat, without any sub-scenarios.

6.1.1. Testing General Aspect Composition procedure

The input section of the testing template of the general aspect composition procedure contains three input variables, and one input action.

1. The primary file: It is an input variable. It is selected from the test case samples {tc1-tc72}, described in the test case map.

2. The generic file: It is an input variable. There is one test case sample (tcGen).
3. The point-cut expression: It is an input variable. It is defined from a set of expressions which covers all the branches of point-cut expression tree.

Table 6-2 describes a set of point-cut expressions developed using point-cut expression BNF grammar

Exp #	Point-cut Expression	Operator	Value
1	step.name <operator> <value>	{ = , != }	{StepA, StepB, StepC}
2	step.stepComponent.name <operator> <value>	{ = , != }	{Comp1, Comp2, Comp3}
3	step.stepComponent.componentHost.name <operator> <value>	{ = , != }	{Host1, Host2, Host3}
4	step.stepComponent.host.name <operator> <value>	{ = , != }	{Host1, Host2, Host3}
5	step.component.name <operator> <value>	{ = , != }	{Comp1, Comp2, Comp3}
6	step.component.componentHost.name <operator> <value>	{ = , != }	{Host1, Host2, Host3}
7	step.component.host.name <operator> <value>	{ = , != }	{Host1, Host2, Host3}

Table 6- 2 Point-cut expressions input

The previous expressions can be joined using "&", and "||" operators to produce additional point-cut expressions

4. Selecting the composition type: It is an input action. The aspect composition tool allows selecting three types of compositions {insertBefore, insertAfter, replace

The previous inputs are described in table 6-3

Input section			
Input variables			Input actions
Primary file	Generic file	Point-cut expression	Select composition type
{tc1- tc72}	tcGen	Table 6.2	{insertBefore, insertAfter, replace}

Table 6- 3 Input section in Black box testing template for general aspect composition procedure

Testing plan of the general aspect composition procedure is developed in appendix E. The testing plan contains samples of the tested inputs and their expected outputs. Matching the exact outputs with the expected ones is the final step in the testing, which is done during the running process.

Table 6-4 summarizes the testing of the general aspect composition procedure.

6.1.2. Testing Security Aspect Composition procedure

The input section of the testing template of the security aspect composition procedure contains two input variables, and one input action.

1. The primary file: It is an input variable. It is selected from the test case samples {tc1- tc72}, described in the test case map in appendix F.
2. The generic file: It is an input variable. There is one test case sample (tcGen).
3. Selecting the security level: It is an input action. The aspect composition tool allows selecting three types of security level {securedHost, securedComponent, securedService}

The previous inputs are described in table 6-5

Variant	Expanding primary model	Parsing point-cut expression	Defining point-cut rules	Identifying join-point	Binding Generic model			composing			Saving composed model
					Identifying parametric values	Defining static values	Binding static & dynamic values	Insert before	Insert after	Replace	
1	X	X	X	X	X	X	X	X			X
2	X	X	X	X	X	X	X	X			X
3	X	X	X	X	X	X	X		X		X
4	X	X	X	X	X	X	X		X		X
5	X	X	X	X	X	X	X			X	X
6	X	X	X	X	X	X	X			X	X
7	X	X	X	X	X	X	X	X			X
8	X	X	X	X	X	X	X	X			X
9	X	X	X	X	X	X	X		X		X
10	X	X	X	X	X	X	X		X		X
11	X	X	X								
12	X	X	X	X	X	X	X			X	X
13	X	X	X								
14	X	X	X	X	X	X	X	X			X
15	X	X	X	X	X	X	X		X		X

Table 6- 4 Summary of Testing General Aspect Composition procedure

Input section		
Input variables		Input actions
Primary file	Generic file	Select security level
{tc1- tc72}	tcGen	{securedHost, securedComponent, securedService}

Table 6- 5 Input section in Black box testing template for security aspect composition procedure

Testing plan of the security aspect composition procedure is developed in appendix F. The testing plan contains samples of the tested inputs and their expected outputs. Matching the exact outputs with the expected ones is the final step in the testing, which is done during the running process.

Table 6-6 summarizes the testing of the security aspect composition procedure.

Variant	Expanding primary model	Defining secure point-cut rules			Identifying join-point	Binding Generic model			composing		Saving composed model
		Secure d Host	Secured Component	Secure d Service		Identifying parametric values	Defining static values	Binding static & dynamic values	Insert before	Insert after	
1	X	X			X	X	X	X	X	X	X
2	X		X		X	X	X	X	X	X	X
3	X			X							
4	X	X									
5	X		X		X	X	X	X	X	X	X
6	X			X							
7	X	X									
8	X		X		X	X	X	X	X	X	X
9	X			X	X	X	X	X	X	X	X
10	X	X			X	X	X	X	X	X	X
11	X		X		X	X	X	X	X	X	X
12	X			X	X	X	X	X	X	X	X
13	X	X			X	X	X	X	X	X	X
14	X		X		X	X	X	X	X	X	X
15	X			X	X	X	X	X	X	X	X

Table 6- 6..Summary of Testing Security Aspect Composition procedure

6.2. Case study (TPC-W)

This section is similar to the previous section. The only difference is that the testing sample is a TPC-W case study. TPC is a Transaction Processing Performance Council benchmarks, which are widely used today in evaluating the performance of computer systems. TPC-W models the workload of an online bookstore.

The primary model of the case represents the functionality without the security mechanisms. Through aspect composition, SSL security communication model is added to the primary model [Woo07]. The case study has two primary models, and one generic model.

1. GetCustRegPage scenario: It is a primary model. The scenario starts by the electronic browser EB which issues a request for the customer registration page. Then, the WebServer gets the necessary images form ImgServer. Finally, the WebServer constructs the html customer registration form and returns it to EB. The scenario is described in figure 6-1
2. GetBuyConfirmPage scenario: It is a primary model. The scenario starts by the EB which issues a request to WebServer for “buy confirm page”. The WebServer gets the corresponding shopping cart object. With 5% probability, a shipping address is passed from EB. The WebServer tries to match the shipping address in the corresponding table in the database. If no address record is found, insert a new address record. Then, the Checkout sub-scenario is invoked. Next, WebServer gets necessary images from ImageServer. Finally, WebServer constructs the html code for the buy confirm page and returns it to EB [Woo07]. The scenario is described in figure 6-2

Figure 6- 1GetCustRegPage scenario

Figure 6- 2 GetBuyConfirmPage scenario

3. Generic SSL aspect model: It is the generic model. The model has two phases; a handshake phase, and a data transfer phase. The scenario is described in figure 6-3

Figure 6- 3 Generic SSL aspect model

6.2.1. Testing General Aspect Composition procedure

The input section of the testing template of the general aspect composition procedure contains three input variables, and one input action.

1. The primary file: It is an input variable. There are two primary models {GetCustRegPage, GetBuyConfirmPage}.
2. The generic file: It is an input variable. There is one generic model (genericSSL).
3. The point-cut expression: It is an input variable. It is defined from a set of expressions which covers all the branches of point-cut expression tree.

Table 6-7 describes a set of point-cut expressions developed using point-cut expression tree.

Exp#	Point-cut Expression	Operator
1	step.<property1> <operator> <value>	{=, !=, >, >=, <, <=}
2	step.stepComponent. <property2> <operator> <value>	{=, !=, >, >=, <, <=}
3	step.stepComponent.componentHost. <property3> <operator> <value>	{=, !=, >, >=, <, <=}
4	step.stepComponent.host. <property3> <operator> <value>	{=, !=, >, >=, <, <=}
5	step.component. <property2> <operator> <value>	{=, !=, >, >=, <, <=}
6	step.component.componentHost. <property3> <operator> <value>	{=, !=, >, >=, <, <=}
7	step.component.host. <property3> <operator> <value>	{=, !=, >, >=, <, <=}

Table 6- 7 Point-cut expressions input

Where:

property1= {name, description, probability, hostDemend, repCount}

property2 = {name, description, multiplicity, parent, schedPolicity, Expression}

property3 = {name, description, multiplicity, opTime, schedPolicity}

value: A string defined by the user.

The previous expressions can be joined using “&”, and “||” operators to produce additional point-cut expression inputs.

4. Selecting the composition type: It is an input action. The aspect composition tool allows selecting three types of compositions {insertBefore, insertAfter, replace}

The previous inputs are described in table 6-8

Input section			
Input variables			Input actions
Primary file	Generic file	Point-cut expression	Select composition type
{GetCustRegPage, GetBuyConfirmPage}	(genericSSL)	Table 6.7	{insertBefore, insertAfter, replace}

Table 6- 8 put section in Black box testing template for general aspect composition procedure

Testing plan of the general aspect composition procedure is developed in appendix G. The testing plan contains samples of the tested inputs and their expected outputs. Matching the exact outputs with the expected ones is the final step in the testing, which is done during the running process.

Table 6-9 summarizes the testing of the general aspect composition procedure.

6.2.2. Testing Security Aspect Composition procedure

The input section of the testing template of the security aspect composition procedure contains two input variables, and one input action.

1. The primary file: It is an input variable. There are two primary models {GetCustRegPage, GetBuyConfirmPage}.
2. The generic file: It is an input variable. There is one generic model (genericSSL).
3. Selecting the security level: It is an input action. The aspect composition tool allows selecting three types of security level {securedHost, securedComponent, securedService} see figure 5-44

The previous inputs are described in table 6-10

Variant	Expanding primary model	Parsing point-cut expression	Defining point-cut rules	Identifying join-point	Binding Generic model			composing			Saving composed model
					Identifying parametric values	Defining static values	Binding static & dynamic values	Insert before	Insert after	Replace	
1	X	X	X	X	X	X	X	X			X
2	X	X	X	X	X	X	X	X			X
3	X	X	X	X	X	X	X		X		X
4	X	X	X	X	X	X	X		X		X
5	X	X	X	X	X	X	X			X	X
6	X	X	X	X	X	X	X			X	X
7	X	X	X	X	X	X	X		X		X
8	X	X	X	X	X	X	X			X	X
9	X	X	X								
10	X	X	X	X	X	X	X	X			X
11	X	X	X	X	X	X	X	X			X
12	X	X	X								
13	X	X	X								
14	X	X	X	X	X	X	X			X	X
15	X	X	X	X	X	X	X			X	X

Table 6- 9 Summary of Testing General Aspect Composition procedure

Input section		
Input variables		Input actions
Primary file	Generic file	Select security level
{GetCustRegPage, GetBuyConfirmPage}	genericSSL	{securedHost, securedComponent, securedService}

Table 6- 10 Input section in black box testing template for security aspect composition procedure

The testing plan of the security aspect composition procedure is developed in appendix H.

The testing plan contains samples of the tested inputs and their expected outputs.

Matching the exact outputs with the expected ones is the final step in the testing, which is done during the running process.

Table 6.11 summarizes the testing of the security aspect composition procedure.

Variant	Expanding primary model	Defining secure point-cut rules			Identifying join-point	Binding Generic model			composing		Saving composed model
		Secure d Host	Secured Component	Secure d Service		Identifying parametric values	Defining static values	Binding static & dynamic values	Insert before	Insert after	
1	X	X			X	X	X	X	X	X	X
2	X		X		X	X	X	X	X	X	X
3	X			X							
4	X	X			X	X	X	X	X		X
5	X		X		X	X	X	X	X	X	X
6	X			X	X	X	X	X	X	X	X
7	X	X			X	X	X	X	X	X	X
8	X		X		X	X	X	X	X	X	X
9	X			X	X	X	X	X	X	X	X
10	X	X			X	X	X	X	X	X	X

Table 6.11 Summary of Testing Security Aspect Composition procedure

Chapter 7. Conclusion

7.1. Conclusion

The thesis designed and implemented tool support for CSM aspect composition as part of PUMA project [Woo05]. The inputs are the primary CSM model, generic CSM model, and the point-cut rules. The output is a composed CSM model, obtained by weaving a set of context-specific aspect models into the primary model. Each context-specific aspect model is inserted in the primary model at specific join-point.

The join-points are identified by point-cut rules defined by the users. Usually these rules are related to a certain aspect. In this thesis, two kind of point-cut expressions were implemented: a) general expressions for any kind of aspects, and b) short-cut expressions for security aspects.

The context-specific aspect is an instantiation of a generic aspect, by binding the parameters of the generic aspect to concrete values from the primary model at the join-point.

The thesis implemented many CSM operations that are necessary for supporting AOM. These operations are used in the steps of the CSM aspect composition. Some of the major operations are related to the flattening of a CSM model that contains sub-scenarios. There are also composition operations that are rather complex (insert before, insert after, and replace).

The tool implemented in the thesis has the following limitations:

1. The aspect model has a single input and a single output. More research is required to extend the composition to aspect with multiple inputs and outputs.

2. If an aspect is inserted in a sub-scenario, it will be executed every time the sub-scenario is invoked.

This work is another step in a larger research project, and aims to perform aspect composition at the intermediate CSM level. This will reduce the gap between UML system design, and the performance models.

7.2. Future work

The work done in the thesis is a small slice in a larger project. There are many directions for future research that can be done in this field in the future. Below are some examples:

1. In this thesis, the composition is performed at the CSM level. It would be interesting if the composition is performed at the UML model level. As discussed, dealing with aspects at the UML level is more challenging due to the complexity of the UML metamodel.
2. If the CSM schema will be updated in the future to meet the needs of the MARTE profile, then the implemented tool would also need to be updated.
3. There are new emerging model transformation languages (such as MOF-QVT recently standardized by OMG). Future implementation of aspect-composition could be implemented in such specialized languages rather than in Java.

References

- [Ald01] Omar Aldawud, Tzilla Elrad, Atef Bader, "A UML Profile for Aspect Oriented Modeling", In Proceedings of OOPSLA 2001 Workshop on Aspect Oriented Programming, 2001.
- [Ald03] Omar Aldawud, Tzilla Elrad, Atef Bader, "UML Profile for Aspect-Oriented Software Development", In Proceedings of Third International Workshop on Aspect-Oriented Modeling, March 2003.
- [Api08] <http://java.sun.com/j2se/1.5.0/docs/api/> , last visited 1/3/2008
- [Asp08] AspectJ, <http://www.eclipse.org/aspectj/>, last visited 1/3/2008
- [Chi94] John Joseph Chilenski and Steven P. Miller; "Applicability of modified condition /decision coverage to software testing", Software Engineering Journal, Volume 9, Issue 5, pp. 193 - 200, September 1994.
- [Cla01] Siobhán Clarke, Robert J. Walker, "Composition Patterns: An Approach to Designing Reusable Aspects", In Proc. of 23rd Int. Conf. on Software Engineering (ICSE), Toronto, Canada, 2001..
- [Cla02] Siobhán Clarke, Robert J. Walker, "Towards a Standard Design Language for AOSD", Proceedings of the 1st international conference on Aspect-oriented software development, pp113-119, Enschede, Netherlands 2002.
- [Cla06] Eric Clayberg, Dan Rubel, "Eclipse: Building Commercial-Quality Plugins", Addison Wesley, Second Edition, 2006.
- [Coo07] Kendra Cooper, Lirong Dai, YiDeng, "Performance Modeling and Analysis of Software Architectures: An Aspect-Oriented UML Based Approach", Science of Computer Programming, Vol, 57, Issue 1, pp 89-108, July 2005.

- [Dai06] Lirong Daia, Kendra Cooper, "Modeling and performance analysis for security aspects", Science of Computer Programming, Volume 61, Issue 1, pp. 58-71 June 2006.
- [Dbf08] <http://java.sun.com/j2se/1.5.0/docs/api/javax/xml/parsers/DocumentBuilderFactory.html> , last visited 25/3/2008
- [Dob08] <http://java.sun.com/j2se/1.5.0/docs/api/javax/xml/parsers/DocumentBuilder.html> , last visited 22/3/2008
- [Ecl08] <http://wiki.eclipse.org/index.php/JFace> , last visited 22/2/2008
- [Eve07] Joerg Evermann,"A Meta-Level Specification and Profile for AspectJ in UML", Proceedings of the 10th international workshop on Aspect-oriented modeling,Vancouver, Canada, pp. 21 - 27, 2007.
- [Fra04] France, R., Ray, I., Georg, G. and Ghosh, S., "An Aspect-Oriented Approach to Early Design Modeling," IEEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4):173-185, August 2004.
- [Fue06] Lidia F, Pablo Sánchez, "Elaborating UML 2.0 Profiles for AO Design", Proc. of the 8th Workshop on Aspect-Oriented Modelling (AOM), 5th Int. Conference on Aspect-Oriented Software Development (AOSD), Bonn (Germany), March 2006.
- [Geo02] Georg G., France, R. and Ray, I. "An Aspect -Based Approach to Modeling Security Concerns". In Proceedings of the Workshop on Critical Systems Development with UML, Dresden, Germany, 2002.

- [Geo03] Georg G. R. France and Indrakshi Ray “Composing aspect models”, The 4th Aspect Oriented Software Development Modeling With UML Workshop, in conjunction with UML 2003 Conference, October 2003.
- [Isr05] Toqeer A Israr, “Transformation of UML 2.0 Models with Performance Annotations to Core Scenario Models” Department of System and Computer Engineering, Carleton University, Ottawa, Canada.
- [Jac06] Andrew Jackson, Siobhan Clarke, “Towards a Generic Aspect Oriented Design Process”, Satellite Events at the MoDELS 2005 Conference, Lecture Notes in Computer Science Vol. 3844, pp 110-119, 2006.
- [Juc08] Use Case Map Navigator (JUCMNav),
<http://jucmnav.softwareengineering.ca/twiki/bin/view/UCM/UcmNav>, last visited 15/2/2008
- [Kic97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin, “Aspect-Oriented Programming”. In Proc. of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997.
- [Kle07] Jacques Klein, Franck Fleurey, Jean-Marx Jezequel, “Weaving Multiple Aspects in Sequence Diagrams”, Lecture Notes in Computer Science, Vol. 4620, pp.167--199, 2007.
- [Meh07] Katharina Mehner_ Gabriele Taentzer, “Supporting Aspect-Oriented Modeling with Graph Transformations”, In Proc. of Workshop Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design, Chicago, IL, 2005.
- [Mos07] Farida Mostefaoui, Julie Vachon, “Formalization of an Aspect-Oriented

- Modeling Approach”, Formal Methods 2006, McMaster University, Hamilton, ON, Canada, 2006.
- [Mus06] Gunter Mussbacher, Daniel Amyot, Michael Weiss, “Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps”, *International Workshop on Requirements Engineering Visualization (REV 2006)*, Minneapolis, USA, September 11, 2006.
- [Mus07a] Gunter Mussbacher, Daniel Amyot, Jon Whittle, and Michael Weiss, “Flexible and Expressive Composition Rules with Aspect-oriented Use Case Maps (AoUCM)”, 10th International Workshop on Early Aspects (EA 2007), Vancouver, Canada, March 13, 2007. Early Aspects: Current Challenges and Future Directions. Moreira, A. and Grundy J. (Eds.). Springer, LNCS 4765:19–38.
- [Mus07b] Gunter Mussbacher, “Aspect-Oriented User Requirements Notation”, SITE, Doctoral Symposium at the 15th IEEE International Requirements Engineering Conference (RE'07), New Delhi, India, October 15, 2007.
- [Mus07c] Gunter Mussbacher, Daniel Amyot, João Araújo, Ana Moreira, Michael Weiss, “Visualizing Aspect-Oriented Goal Models with AoGRL”, Second International Workshop on Requirements Engineering Visualization (REV'07), New Delhi, India, October 15, 2007.
- [OMG05] Object Management Group, “UML Profile for Schedulability, Performance, and Time” (SPT), version 1.1, see <http://www.omg.org/cgi-bin/doc?formal/2005-01-02>

- [OMG07] Object Management Group, “UML Profile for Modeling and Analysis of Real-Time and Embedded Systems” (MARTE), <http://www.omg.org/docs/ptc/07-08-04.pdf>, August 2007
- [Par08] <http://java.sun.com/j2se/1.4.2/docs/api/javax/xml/parsers/package-summary.html>, last visited 6/3/2008
- [Pet04] Dorin B. Petriu, Murray Woodside, “A Metamodel for Generating Performance Models from UML Designs”, Proc. UML 2004 - Modelling Languages and Applications, 7th Int. Conference, Lisbon, Portugal, LNCS 3273, pp. 41-53, Springer 2004.
- [Pet07a] Dorin B. Petriu, Murray Woodside, “An Intermediate metamodel with scenarios and resources for generating performance models from UML designs”, Software and Systems Modeling, Vol.6, Nb. 2, pp. 163-184, 2007.
- [Pet07b] D.C. Petriu, C.M. Woodside, D.B. Petriu, J. Xu, T. Israr, Geri Georg, R. France, J.M. Bieman, S. H. Houmb, “Performance Analysis of Security Aspects in UML Models”, Proc. 6th Int. Workshop on Software and Performance WOSP’2007, Buenos Aires, Argentina, 2007.
- [She05] Hui Shen and Dorina C. Petriu, “Performance Analysis of UML Models Using Aspect-Oriented Modeling Techniques”, Model Driven Engineering Languages and Systems, (L. Briand and C. Williams, Eds). LNCS Vol. 3713, pp.156-170, Springer, 2005.
- [Smi90] Connie U. Smith, “Performance Engineering of Software Systems”, Reading Mass., Addison Wesley, 1990.
- [Tpc08] <http://www.tpc.org/tpcw/default.asp> , last visited 22/3/2008

- [W3c08] <http://www.w3.org/DOM/> , last visited 13/3/2008
- [Woo05] Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Hui Shen, Toqeer Israr, Jose Merseguer, “Performance by Unified Model Analysis (PUMA)”, Proc.of The 5th ACM Workshop on Software and Performance, pp. 1-12, Palma, Spain, July 11-14, 2005.
- [Woo06] Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Hui Shen, Tauseef Israr, Jose Merseguer, “Performance by Unified Model Analysis (PUMA)”, extended version of [Woo05], submitted for publication, 2006.
- [Woo07] Murray Woodside , Dorina C. Petriu, Dorin B. Petriu, Jing Xu, Tauseef Israr, Geri Georg, Robert France, James M. Bieman, Siv Hilde Houmb, Jan Jürjens, “Performance Analysis of Security Aspects by Weaving Scenarios from UML Models”, (extended version of [Pet07b], accepted for publication in Journal of Systems & Software, 2008.

Appendix A: Algorithms of Composition operations

```
getElement(doc, elementType, name) {  
    For each element in doc which got an elementType tagName{  
        If ( element.getAttribute("name") = name ) return element  
        else return null  
    }  
}  
  
createScenario(step, refinement) {  
    step.appendChild(refinement)  
}  
  
ReplaceConn(C, connA, stepA, stepZ) {  
    If ( stepA.getAttribute("id") = connA.getAttribute("source")){  
        For each element in C {  
            If (conn.getAttribute("id") = connA.getAttribute("id")) {  
                conn.setAttribute("source", stepZ.getAttribute("id"));  
            }  
        }  
    }else if ( stepA.getAttribute("id") = connA.getAttribute("target")){  
        For each element in C {  
            if ( conn.getAttribute("id") = connA.getAttribute("id")) {  
                conn.setAttribute("target", stepZ.getAttribute("id"));  
            }  
        }  
    }  
    return C  
}
```

```
StepComponent(doc, StepA) {  
    flag1 = true  
    child = StepA  
    chld = StepA  
    while(flag1 = true & StepA does not have child){  
        parent = chld.getParentNode()  
        next = parent.getChildNodes().item(1)  
        flag3 = false  
        nxt = next  
        prnt = parent.getLastChild().getPreviousSibling()  
        while( nxt.getAttribute("id") != prnt.getAttribute("id") & flag3 = false){  
            if(next.getNodeType() = 1){  
                nxt= (Element)next  
                if(nxt.getTagName() = Step & chld.getAttribute("predecessor")  
                = (nxt.getAttribute("successor")) & flag3 = false) {  
                    chld = nxt  
                    flag3= true  
                }  
            }  
        }  
    }  
}
```

```

    } else if (nxt.getTagName() = ResourceRelease &
chld.getAttribute("predecessor")= nxt.getAttribute("successor") &
flag3=false ){
        chld = nxt
        flag3 = true
        rr.add(chld)
    } else if(nxt.getTagName() = ResourceAcquire &
chld.getAttribute("predecessor")= nxt.getAttribute("successor") &
flag3 = false ){
        if (rr is not empty){
            g = true
            For each element in rr {
                if(nxt.getAttribute("acquire") =
element.getAttribute("release")){
                    chld = nxt
                    rr.remove(i)
                    g = false
                }
            }
            if (g = true){
                child = nxt
                flag3 = true
            }
        } else {
            child = nxt
            flag3 = true
        }
        flag3= true
    }
}
next= next.getNextSibling()
}
if(child.getTagName() = ResourceAcquire){
    For each component in doc {
        if(component.getAttribute("id") = child.getAttribute("acquire")){
            child= component
        }
    }
    flag1= false
}
}
return child
}

createStep(doc) {
    return doc.createElement( "Step" )
}

```

```

First(scenario) {
    child = scenario.getChildNodes()
    For each element in child {
        if (child.getClassName() = Step) {
            out = child
            break if statement
        }
    }
    return out
}

Last(scenario) {
    Child = scenario.getChildNodes()
    For each element in child(from last to first) {
        if (child.getClassName() = Step) {
            out = child
            break if statement
        }
    }
    return out
}

dotStep(doc, stp) {
    For each connector in doc {
        if (connector.getAttribute("id") = stp.getAttribute("predecessor")) {
            topConn = connector
        }
    }
    return topConn
}

doubleDotStep(doc, stp) {
    flag1= true
    child= stp
    while(flag1 =true & stp does not have child){
        parent = child.getParentNode()
        next= parent.getChildNodes().item(1)
        flag3= false
        nxt= next
        prnt= parent.getLastChild().getPreviousSibling()
        while (nxt.getAttribute("id") != prnt.getAttribute("id") & flag3=false){
            if(next.getNodeType() = 1){
                nxt= next
                if(nxt.getTagName() = Step OR nxt.getTagName() = ResourceAcquire
                OR nxt.getTagName() = ResourceRelease){
                    if(child.getAttribute("predecessor") = nxt.getAttribute("successor")
                    & flag3=false){
                        child= nxt
                    }
                }
            }
        }
    }
}

```

```

        flag3= true
    }
}
next = next.getNextSibling()
}
if( child.getNodeName() != Step & flag3 = false){
    childParent= child.getParentNode()
    For each refinement in doc {
        if( refinement.getAttribute("sub") = childParent.getAttribute("id")){
            subStep= (Element)refinement.getParentNode()
        }
    }
    subSc = subStep.getParentNode()
    first = subSc.getChildNodes().item(1)
    last = subSc.getLastChild().getPreviousSibling()
    frst = first
    while( frst.getAttribute("id") != last.getAttribute("id") & flag3= false){
        if(first.getNodeType() = 1){
            frst= (Element) first
            if(frst.getTagName() = Step OR frst.getTagName() =
ResourceAcquire OR frst.getTagName() = ResourceRelease){
                if(subStep.getAttribute("predecessor") =
frst.getAttribute("successor") & flag3 = false){
                    child= frst
                    flag3= true
                }
            }
        }
        first= first.getNextSibling()
    }
}
if(child.getTagName() = Step){
    if( child does not have child ){
        out.add(child)
        flag1= false
    }
    else if(child has child){
        chld= (Element) child.getChildNodes().item(1)
        For each scenario in doc {
            if(chld.getAttribute("sub") = scenario.getAttribute("id")){
                subStep = scenario.getChildNodes().item(1)
                last = secnario.getLastChild().getPreviousSibling()
                frst= subStep
                while( frst.getAttribute("id") != last.getAttribute("id")){
                    if(subStep.getNodeName() = Step){
                        flag1= false

```

```

        child = subStep
        out.add(child)
    }
    subStep= subStep.getNextSibling();
}
}
}
}
}
}
}
}
}

return out
}

DotClosureStep(doc, stp) {
    parent = stp.getParent()
    child = parent.getChildNodes()
    For each element in child {
        if ( element != stp) elements[] = element
        else break if statement
    }
    return element[]
}

stepDot(doc, stp) {
    For each connector in doc {
        if (connector.getAttribute("id") = stp.getAttribute("successor")) {
            bottomConn = connector
        }
    }
    return bottomConn
}

stepDoubleDot(doc:Document, stp:Element) {
    flag1= true
    child= stp
    while(flag1 = true & stp does not have child ){
        parent = child.getParentNode()
        next = parent.getChildNodes().item(1)
        flag3 = false
        nxt = next
        prnt = parent.getLastChild().getPreviousSibling()
        while( nxt.getAttribute("id") != prnt.getAttribute("id") & flag3 = false){
            if(next.getNodeType() = 1){
                nx= next
                if(nxt.getTagName() = Step OR nxt.getTagName() = ResourceAcquire
                OR nxt.getTagName() = ResourceRelease){

```

```

        if(child.getAttribute("predecessor") = nxt.getAttribute("successor")
        & flag3=false){
            child= nxt
            flag3= true
        }
    }
    next= next.getNextSibling()
}
if( child.getNodeName() != Step & flag3 = false){
    childParent = child.getParentNode()
    For each refinement in doc{
        if(refinement.getAttribute("sub") = childParent.getAttribute("id")){
            subStep = refinement.getParentNode()
        }
    }
    subSc = subStep.getParentNode();
    first = subSc.getChildNodes().item(1)
    last = subSc.getLastChild().getPreviousSibling()
    frst = first
    while( frst.getAttribute("id") != last.getAttribute("id") & flag3 = false){
        if(first.getNodeType() = 1){
            frst = first
            if(frst.getTagName() = Step OR frst.getTagName() =
            ResourceAcquire OR frst.getTagName() = ResourceRelease ){
                if(subStep.getAttribute("predecessor") =
                frst.getAttribute("successor") & flag3 = false){
                    child= frst
                    flag3= true
                }
            }
        }
        first= first.getNextSibling();
    }
}
if(child.getTagName() = Step ){
    if( child does not have child ){
        out.add(child)
        flag1= false
    }
    else if (child has child){
        chld= child.getChildNodes().item(1)
        For each scenario in doc {
            if(chld.getAttribute("sub") = scenario.getAttribute("id")){
                subStep = scenario.getChildNodes().item(1)
                last = secnario.getLastChild().getPreviousSibling()
                frst = subStep
                while(frst.getAttribute("id") != last.getAttribute("id")){

```



```

        if(subStep.getNodeName() = Step ){
            flag1= false
            child = subStep
            out.add(child)
        }
        subStep= subStep.getNextSibling()
    }
}
}
}
}
}
return out
}
}

stepDotClosure(doc, stp) {
    parent = stp.getParent()
    child = parent.getChildNodes()
    For each element in child {
        if ( element = stp) trigger =true
        if (trigger) elements[] = element
    }
    return element[]
}

Steps(doc) {
    return doc.getElementByTagName("Step")
}

createSequence(doc) {
    return doc.createElement("Sequence")
}

Connectors(doc) {
    return doc.getElementByTagName("Sequence")
}

Seq(stepA, stepB) {
    if(stepA.getAttribute("successor") = stepB.getAttribute("predecessor"))
        return stepA.getAttribute("successor")
    else return null;
}

Start(scenario){
    childScenario = scenario.getChildNodes()
    For each element in childScenario{
        if(child.getClass().getName() = Start ){
            out= child
        }
    }
}

```

```

    }
  }
  return out
}

```

End(scenario){

```

    childScenario = scenario.getChildNodes()
    For each element in childScenario{
        if(child.getClass().getName() = End ){
            out= child
        }
    }
    return out
}

```

Resources(doc){

```

    return doc.getElementsByTagName("Component")
}

```

InsertBeforeResource(primary, elementName, joinPoint, contextDoc){

```

//Define two random variables
randomOne= id.getRandomId();
randomTwo= id.getRandomId();
//Check the uniqueness id of the primary and the context
id.setDocumentUniqueId(primary, contextDoc)
//Find the successor sequence
For each sequence in primary{
    if(sequence.getAttribute("id") = joinPoint.getAttribute("successor")){
        bottomConnector= sequence
    }
}
// create new sub-scenario with the successor connector
step.createStep(primary, contextDoc)
refinement.createRefinement(primary, contextDoc)
scenario.createScenario(step.getNewStep(), refinement.getNewRefinement())
connect.createSequence(primary)
// Set the new attributes values
step.getNewStep().setAttribute("predecessor",
joinPoint.getAttribute("predecessor"))
step.getNewStep().setAttribute("successor", randomOne)
refinement.getNewRefinement().setAttribute("parent",
joinPoint.getAttribute("id"))
connect.getNewSequence().setAttribute("id", randomOne)
connect.getNewSequence().setAttribute("source", joinPoint.getAttribute("id"))
connect.getNewSequence().setAttribute("target", randomTwo)
joinPoint.setAttribute("id", randomTwo)
joinPoint.setAttribute("predecessor", randomOne)

```

```

bottomConnector.setAttribute("source", randomTwo)
//Insert the new step and sequence
joinPoint.getParentNode().insertBefore(step.getNewStep(), joinPoint)
bottomConnector.getParentNode().insertBefore(connect.getNewSequence(),
bottomConnector)
}

InsertAfterResource(primary, elementName, joinPoint, contextDoc){

    //Get the Resource type from the component name
    cutPointComponent= getElement(primary,"Component", joinPoint);
    For each resource in primary {
        if(elementName = ResourceAcquire){
            if(resource.getAttribute("acquire") = cutPointComponent.getAttribute("id")){
                cutPointResource = resource
            }
        } else if(elementName = ResourceRelease ){
            if(resource.getAttribute("release") =
            cutPointComponent.getAttribute("id")){
                cutPointResource = resource
            }
        }
    }

    For each sequence in primary {
        if(sequence.getAttribute("id") =
        cutPointResource.getAttribute("successor")){
            bottomConnector = sequence
        }
    }

    // Insert after: If Resource followed by a Step
    For each step in primary {
        if(step.getAttribute("id") = bottomConnector.getAttribute("target")){
            scenario.InsertBeforeStep(primary,step,contextDoc)
        }
    }

    //Insert after: If Resource followed by a ResourceAcquire
    For each resource in primary {
        if(resource.getAttribute("id") = bottomConnector.getAttribute("target")){
            For each component in primary {
                if(component.getAttribute("id") =
                resource.getAttribute("acquire")){
                    InsertBeforeResource(primary,
                    "ResourceAcquire",resource,contextDoc)
                }
            }
        }
    }
}

```

```

    }
  }
  // Insert after: If Resource followed by a ResourceRelease
  For each resource in primary {
    if(resource.getAttribute("id") = bottomConnector.getAttribute("target")){
      For each component in primary {
        if(component.getAttribute("id") = resource.getAttribute("release")){
          InsertBeforeResource(primary,
            "ResourceRelease",resource,contextDoc);
        }
      }
    }
  }
}

createRefinement(doc){
  return doc.createElement( "Refinement" )
}

createNewDocument(){
  document= builder.newDocument()
}

setParsedDocumnet(){
  document= builder.parse(file)
}

mergeDoc(Document doc) {

  For each scenario in doc {
    dup = document.importNode(scenario, true)
    document.getDocumentElement().getElementsByTagName("Scenario")
    .item(primaryList.getLength()-1).getParentNode().appendChild(dup)
  }

  For each component in doc {
    dup = document.importNode(component, true)

    document.getDocumentElement().getElementsByTagName("Scenario")
    .item(primaryList.getLength()-1).getParentNode().appendChild(dup)
  }

  For each resource in doc {
    dup = this.document.importNode(resource, true)
    document.getDocumentElement().getElementsByTagName("Scenario")
    .item(primaryList.getLength()-1).getParentNode().appendChild(dup);
  }
  document.normalizeDocument()
}

```

```

expandDoc() {

    // Find the Start(target) and the End(Source),and get the subscenario attributes
    (Parent step)
    For each scenario in document {
        For each refinement in document {
            if(refinement.getAttribute("sub") = scenario.getAttribute("id")){
                parentStep= refinement.getParentNode()

            }

        }

    }

    //The first loop is used to get the Start(target) and the End(source)
    //The two attributes will be used to link the subscenario with the top one.

    For each childNode in scenario {
        if(childNode.getNodeName() = Start){
            target= childNode.getAttribute("target")
        }
        if(childNode.getNodeName() = End &
            childNode.getAttribute("noSync") = false){
            source= childNode.getAttribute("source")
        }
    }
}

//Find the first element after the Start, and the last element before the End
For each childNode in scenario {
    if(childNode.getAttribute("id").equals(target)){
        firstChild = scenario.getChildNodes().item(1)
    }
    if(childNode.getAttribute("id").equals(source)){
        lastChil= scenario.getChildNodes().item(1)
    }
}

//Find the first sequence after the first child
For each childNode in scenario {
    if(childNode.getAttribute("id") =
        firstChild.getAttribute("successor")){
        firstSequence = scenario.getChildNodes().item(1)
    }
}

For each sequence in document {
    if(lastSequence.getAttribute("id") =
        parentStep.getAttribute("successor")){
        lastSequence.setAttribute("source",lastChild.getAttribute("id"))
    }
}

```

```

        For each end in document {
            if(lastSequence.getAttribute("id") =
            parentStep.getAttribute("successor")){
                lastSequence.setAttribute("source", lastChild.getAttribute("id"))
            }
        }

//Set the new values
    firstChild.setAttribute("predecessor",
    parentStep.getAttribute("predecessor"))
    firstChild.setAttribute("id",parentStep.getAttribute("id"))
    firstSequence.setAttribute("source", parentStep.getAttribute("id"))
    lastChild.setAttribute("successor", parentStep.getAttribute("successor"))
}
}
}
//import the scenarios into the top one
For each scenario in document {
    For each childNode in scenario {
        if(childNode.getNodeName() != Start){
            if(childNodes.getNodeName() != End OR
            childNode.getAttribute("noSync").equals("true") ){
                moveNode = document.importNode(childNode, true)

                document.getDocumentElement().getElementsByTagName(
                "Scenario").item(0).appendChild(moveNode)
            }
        }
    }
}
}
}

```

```

getParametricNames() {
    nNode= document.getElementsByTagName("CSM:CSMType").item(0)
    cNode= nNode.getChildNodes().item(1)
    x= nNode.getChildNodes().getLength()-1
    while((x--)!= 0){
        cElement = cNode
        if(cElement.hasAttribute("name")){
            if(cElement.getAttribute("name").startsWith("|")){
                genericNames.add(cElement.getAttribute("name"))
            }
        }
        if(cNode.hasChildNodes()){
            gNode= cNode.getChildNodes().item(1)
            y = cNode.getChildNodes().getLength()-1
            while((y--)!=0){
                gElement= gNode

```

```

        if(gElement.hasAttribute("name")){
            if(gElement.getAttribute("name").startsWith("|")){
                genericNames.add(gElement.getAttribute("name"))
            }
        }
        gNode= gNode.getNextSibling()
    }
    cNode=cNode.getNextSibling()
}
return genericNames
}

```

```

initDocument(file) {
    openFile(file)
    setParsedDocumnet()
}

```

```

createElement(docComposed, docContext) {
    createStep(docComposed, docContext)
    createRefinement(docComposed, docContext)
    createScenario(step, refinement)
    createSequence(docComposed)
}

```

```

compareString(stepCondition, value, prelim ){
    if(prelim.equals("=")){
        return (stepCondition = value)
    }
    else if(prelim.equals("!=")){
        return (stepCondition != value)
    }
    else return false
}

```

```

compareInteger(stepCondition, value, prelim){
    if(prelim.equals("="))
        return getInteger(stepCondition) = getInteger(value)
    else if(prelim.equals("!="))
        return getInteger(stepCondition) != getInteger(value)
    else if(prelim.equals(">")){
        if(parseInt(stepCondition) > parseInt(value)) return true
        else return false
    }else if(prelim.equals("<")){
        if(parseInt(stepCondition) < parseInt(value)) return true
        else return false
    }
}

```

```

    }else if(prelim.equals(">=")){
        if(parseInt(stepCondition) >= parseInt(value))return true
        else return false
    }else if(prelim.equals("<=")){
        if(parseInt(stepCondition) <=parseInt(value))return true
        else return false
    }else return false;
}

compareDouble(stepCondition, value, prelim){
    if(prelim.equals("="))
        return valueOf(stepCondition) ==.valueOf(value)
    else if(prelim.equals("!="))
        return valueOf(stepCondition) !=.valueOf(value)
    else if(prelim.equals(">")){
        if(parseDouble(stepCondition) > parseDouble(value)) return true
        else return false
    }else if(prelim.equals("<")){
        if(parseDouble(stepCondition) < parseDouble(value)) return true
        else return false
    }else if(prelim.equals(">=")){
        if(parseDouble(stepCondition) >= parseDouble(value)) return true
        else return false
    }else if(prelim.equals("<=")){
        if(parseDouble(stepCondition) <=parseDouble(value)) return true
        else return false
    }else return false
}

getRandomId() {
    rd= makeRandom()
    uId= "id"+ rd.toString()
    return uId
}

getRedundantElement(doc, out){
    flag= false
    nNode = doc.getElementsByTagName("CSM:CSMType").item(0)
    cNode = nNode.getChildNodes().item(1)
    x = nNode.getChildNodes().getLength()-1
    while((x--) != 0 & flag = false){
        if(cNode.getNodeType() ==1){
            cElement = cNode
            if(cElement.getAttribute("id") = out){
                elementId= cElement
                flag= true
            }
        }
    }
}

```



```

        if(cNode has child){
            gNode = cNode.getChildNodes().item(1)
            y = cNode.getChildNodes().getLength()-1
            while((y--) !=0 & flag = false){
                if(gNode.getNodeType() =1){
                    gElement = gNode
                    if(gElement.getAttribute("id") = out){
                        elementId= gElement
                        flag= true
                    }
                }
                gNode= gNode.getNextSibling()
            }
        }
        cNode=cNode.getNextSibling()
    }
    return elementId;
}

getElementsId(doc) {
    nNode= doc.getElementsByTagName("CSM:CSMType").item(0)
    cNode= nNode.getChildNodes().item(1)
    x= nNode.getChildNodes().getLength()-1
    while((x--) != 0){
        if(cNode.getNodeType() = 1){
            cElement = cNode
            if( cElement.getAttribute("id") != (""))
                elementId.add(cElement.getAttribute("id"))
        }
        if(cNode has child){
            gNode= cNode.getChildNodes().item(1)
            y= cNode.getChildNodes().getLength()-1
            while((y--)!=0){
                if(gNode.getNodeType()=1){
                    gElement= gNode
                    if(gElement.getAttribute("id") != (""))
                        elementId.add(gElement.getAttribute("id"))
                }
                gNode= gNode.getNextSibling()
            }
        }
        cNode=cNode.getNextSibling()
    }
    return elementId;
}

makeRandom(){
    randInt = nextInt()

```

```

    randInt= abs(randInt)
    return randInt
}

andOrParser(pointcutString) {
    getOrAndPrelim(pointcutString)
    instantiate st as a StringTokenizer with a value of pintcutString
    while(st.hasMoreElements()){
        put in hashMap the next token using "&" prelim
    }
    For each element in the hashMap {
        st= prelimAND.get(j).stTokenizer
        if(st.hasMoreElements()){
            while(st.hasMoreElements()){
                put in hashMap the next token using "||" prelim
            }
        }else{
            put in hashMap the previous token of st
        }
    }

    return prelim; // (END STATE) done.
}

getOrAndPrelim(pointcutString){
    v1= 0
    v2=1
    v3=0
    v4=1
    orAndPrelim.clear()
    orAndPrelim.add(" ")
    while(v2 != 0 & v4 != 0){
        v1= pointcutString.indexOf("&", v3)
        v2= pointcutString.indexOf("||", v4)
        if(v1<v2 OR v2=-1){
            orAndPrelim.add("&")
            v3= v1+1
        }
        if(v2<v1OR v1=-1){
            orAndPrelim.add("||")
            v4= v2+1
        }
    }
}

```

```

getJoinPoint(output){
  if(the size of output = 0) return false
  else if(the size of output =1) return the value of tokebBool at position 0
  else{
    token= the value of tokebBool at position 0
    For each position of output {
      if ( the value of orAnd at position x is "&"){
        token = token && the value of tokenBool at posiotn x
      }
      if (the value of orAnd at position x is "||"){
        token = token || the value of tokenBool at posiotn x
      }
    }
    return token
  }
}

isService(step){
return step.getAttribute("description").contains("service")
}

```

Appendix B: The textual description of the usecases

Use case: Open file

Name: Open file

Actors: user

Description: The user browses for a CSM file to open

Trigger: Open file request from the user.

Summary: When the user requests the file opening, a file open dialog appears. The user selects the required CSM file.

Steps:

User actions	Tool responses
1. Choose 'Open file' command	2. File open dialog appears
3. Specify file name	
4. Confirm selection	5. Dialog disappears

Use case: Open primary

Name: Open primary.

Actors: user

Description: The user browses for a primary file to open

Trigger: Open file request from the user.

Summary: When the user requests the file opening, a primary file open dialog appears. The user selects the required primary file.

Steps:

User actions	Tool responses
1. Choose 'Open primary file' command	2. Primary file open dialog appears

3. Specify primary file name

4. Confirm selection

5. Dialog disappears

Use case: Open generic

Name: Open generic.

Actors: user

Description: The user browses for a generic file to open

Trigger: Open file request from the user.

Summary: When the user requests the file opening, a generic file open dialog appears. The user selects the required generic file.

Steps:

User actions	Tool responses
1. Choose 'Open generic file' command	2. Generic file open dialog appears
3. Specify generic file name	
4. Confirm selection	5. Dialog disappears

Use case: Expand model

Name: Expand model.

Actors: user

Description: The user views the opened models in expanded format.

Trigger: Expand request from the user.

Summary: When the user requests the expand command, the non-flattened viewed model will be viewed as a flatten model.

Steps:

User actions	Tool responses
1. Choose 'Expand' command	2. A flattened model appears

Use case: Compose Aspect

Name: Compose Aspect.

Actors: user

Description: The tool composes the context-specific aspect model, which is produced from the generic model, with primary model.

Trigger: Compose request from the user.

Summary: When the user requests the compose command, the primary model will be expanded internally. The join-point will be identified based on point-cuts rules defined by the user. The context-specific aspect will be produced from the generic model based on the join-point location. The context-specific aspect will be composed with the primary based on the join-point location.

Steps:

User actions	Tool responses
1. Choose 'Compose' command	2. Tool expands the primary model internally
	3. Point-cut Rules window appears
4. Type point-cut rules	5. Tool identifies the joint point based on the point-cut rules
	6. Compose aspect window appears
7. Choose the 'Binds generic' command	
8. Define the static parametric values	
9. Confirm the selection.	10. Tool binds parametric values into

concrete ones

11. Based on the binding, tool transfers the generic aspect is into context-specific aspect
12. Confirm the composition
13. Compose the primary with the context-specific aspect
14. Composed model is viewed

Use case: Insert before

Name: Insert before.

Actors: user

Description: The tool composes the context-specific aspect model, which is produced from the generic model, with primary model using an insert before option.

Trigger: Insert before request from the user.

Summary: When the user requests the compose command, the primary model will be expanded internally. The join-point will be identified based on point-cuts rules defined by the user. The context-specific aspect will be produced from the generic model based on the join-point location. The context-specific aspect will be composed with the primary based on the join-point location using an insert before option.

Steps:

User actions

Tool responses

1. Choose 'Compose' command	2. Tool expands the primary model internally
	3. Point-cut Rules window appears
4. Type point-cut rules	5. Tool identifies the joint point based on the point-cut rules

6. Compose aspect window appears
7. Choose the 'Binds generic' command
8. Define the static parametric values.
9. Confirm the selection
10. Tool binds parametric values into concrete ones.
11. Based on the binding, tool transfers the generic aspect is into context-specific aspect
12. Select InsertBefore option
13. Confirm the selection
14. Compose the primary with the context-specific aspect
15. Composed model is viewed

Use case: Insert after

Name: Insert after.

Actors: user

Description: The tool composes the context-specific aspect model, which is produced from the generic model, with primary model using an insert after option.

Trigger: Insert after request from the user.

Summary: When the user requests the compose command, the primary model will be expanded internally. The join-point will be identified based on point-cut rules defined by the user. The context-specific aspect will be produced from the generic model based on the join-point location. The context-specific aspect will be composed with the primary based on the join-point location using an insert after option.

Steps:

User actions	Tool responses
1. Choose 'Compose' command	2. Tool expands the primary model internally
	3. Point-cut Rules window appears
4. Type point-cut rules	5. Tool identifies the joint point based on the point-cut rules
	6. Compose aspect window appears
7. Choose the 'Binds generic' command	
8. Define the static parametric values	
9. Confirm the selection	10. Tool binds parametric values into concrete ones
	11. Based on the binding, tool transfers the generic aspect is into context-specific aspect
12. Select InsertAfter option	
13. Confirm the selection	14. Compose the primary with the context-specific aspect
	15. Composed model is viewed

Use case: Replace

Name: Replace.

Actors: user

Description: The tool composes the context-specific aspect model, which is produced from the generic model, with primary model using a replace option.

Trigger: Replace request from the user.

Summary: When the user requests the compose command, the primary model will be expanded internally. The join-point will be identified based on point-cut rules defined by the user. The context-specific aspect will be produced from the generic model based on the join-point location. The context-specific aspect will be composed with the primary based on the join-point location using a replace option.

Steps:

User actions	Tool responses
1. Choose 'Compose' command	2.Tool expands the primary model internally
	3. Point-cut Rules window appears
4. Type point-cut rules	5. Tool identifies the joint point based on the point-cut rules
	6. Compose aspect window appears
7. Choose the 'Binds generic' command	
8. Define the static parametric values	
9. Confirm the selection	10. Tool binds parametric values into concrete ones
	11. Based on the binding, tool transfers the generic aspect is into context-specific aspect
12. Select replace option	
13. Confirm the selection	14. Compose the primary with the context-specific aspect
	15. Composed model is viewed

Use case: Identify Join-point

Name: Identify Join-point.

Actors: System.

Description: The system searches for the steps in the primary model that are matching the point-cut rule.

Trigger: Confirm the selection of the point-cut rule.

Summary: When the user confirms the selection of the point-cut rule, the system parses the rule into a set of tokens. Then each token is tested by the set of steps of the primary model. The system selects the steps that match all the tokens of the point-cut rule.

Steps:

User actions	Tool responses
1. The user confirms point-cut rule	2. The tool checks the grammar of the rule, then it parses it into a set of tokens
	3. The tool searches for all steps in the primary model
	4. By applying the attributes' values of the steps. The tool tests the compliance of the steps with the tokens. Then it defines a list of JPs

Use case: Define point-cuts

Name: Define point-cuts

Actors: user

Description: The user types or select the point-cut rules based on specified rules.

Trigger: Defined aspect request from the user.

Summary: When the user requests a defined aspect, a point-cut window appears. The user defines the point-cut rules.

Steps:

User actions	Tool responses
1. Select the level of point-cut	2. Point-cut window appears
3. Define the point-cut rules	
4. Confirm selection	5. Window disappears

Use case: Define general point-cuts

Name: Define general point-cuts

Actors: user

Description: The user types the point-cut rules based on specified rules.

Trigger: General aspect request from the user.

Summary: When the user requests a general aspect, a point-cut text box appears. The user types the rules.

Steps:

User actions	Tool responses
1. Choose 'General aspect' command	2. Text box appears
3. Type the point-cut rule	
4. Confirm selection	5. Window disappears

Use case: Define security point-cuts

Name: Define security point-cuts

Actors: user

Description: The user selects the point-cut rules based on a certain security level.

Trigger: Security aspect request from the user.

Summary: When the user requests a security aspect, a security level window appears. The tool provides three levels host, server, and service. The user selects the security levels. Then under each selected level, the user selects the hosts, servers, services he wants to secure.

Steps:

User actions	Tool responses
1. Choose 'Security aspect' command	2. Security aspect window appears
3. Check the security level	
4. Confirm selection	5. Based on the selected security levels, Securing Host/Server/Service window appears
6. Choose Get secure hosts/ servers/ services button	7. Selecting secure hosts/servers/ services window appears
8. Select the secured hosts/ servers/ services	
9. Confirm selection	10. Window disappears

Use case: Bind Generic

Name: Bind Generic.

Actors: System

Description: The system binds the parametric values in the generic model to concrete ones. Based on the join-point, the concrete values are extracted. Or they are defined by the user.

Trigger: Define parametric values request from the user.

Summary: When the user requests Defining parametric values, a file open dialog appears.

The user selects the required generic file. Then form the CSM binding window, the user defines some of the concrete values, and leaves the rest to the system to be defined based on the join-point location.

Steps:

User actions	Tool responses
1. Select 'Define parametric values' button	2. Open generic file dialog appears
3. Specify file name	
4. Confirm selection	5. Dialog disappears
	6. CSM Binding window appears
7. Check the static values	
8. Select 'Refresh' button	
9. Type the concrete value	
10. Confirm selection	11. Window disappears
	12. The tool defines the dynamic values, then based on the join-point, and point-cut rules, it defines the concrete values

Use case: Save file

Name: Save file.

Actors: user

Description: The user saves a CSM file.

Trigger: Save file request from the user.

Summary: When the user requests the file saving, a file save dialog appears. The user selects the required CSM file.

Steps:

User actions	Tool responses
1. Choose 'Save file' command	2. File save dialog appears
3. Specify file name	
4. Confirm selection	5. Dialog disappears

Appendix C: Plug-in.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
  id="ca.carleton.sce.puma.csm.viewer"
  name="Viewer Plug-in"
  version="0.8.8"
  provider-name=""
  class="ca.carleton.sce.puma.csm.viewer.ViewerPlugin">

  <runtime>
    <library name="viewer.jar">
      <export name="*" />
    </library>
  </runtime>

  <requires>
    <import plugin="org.eclipse.ui" />
    <import plugin="org.eclipse.core.runtime" />
    <import plugin="CSM" />
    <import plugin="org.eclipse.gef" />
    <import plugin="org.eclipse.core.resources" />
    <import plugin="org.eclipse.ui.editors" />
    <import plugin="org.eclipse.ui.ide" />
  </requires>
  <extension
    point="org.eclipse.ui.actionSets">
    <actionSet
      label="CSM Viewer"
      visible="true"
      id="ca.carleton.sce.puma.csm.viewer.actionSet">
      <menu
        label="CSM & Viewer"
        id="sampleMenu">
        <separator name="sampleGroup" />
      </menu>
      <action
        toolbarPath="CSM Viewer"
        label="CSM & Viewer"

class="ca.carleton.sce.puma.csm.viewer.actions.OpenCSMViewerAction"
        tooltip="CSM Viewer"
        icon="icons/sample.gif"
        menubarPath="sampleMenu/sampleGroup"

id="ca.carleton.sce.puma.csm.viewer.actions.OpenCSMViewerAction" />
      </actionSet>
    </extension>
    <extension
      point="org.eclipse.ui.actionSets">
    </extension>

</plugin>
```


Appendix D: Test cases map

TC#	StepA	StepB	StepC	RefineA	RefineB	RefineC	ResourceA1n	ResourceA2n	ResourceA3n	Comp1	Comp2	Comp3	Host1	Host2	Host3
1	X	X	X				X			X			X		
2	X	X	X				X			X	X		X		
3	X	X	X				X			X	X	X	X		
4	X	X	X				X			X	X		X	X	
5	X	X	X				X			X	X	X	X	X	
6	X	X	X				X			X	X	X	X	X	X
7	X	X	X	X			X			X			X		
8	X	X	X	X			X			X	X		X		
9	X	X	X	X			X			X	X	X	X		
10	X	X	X	X			X			X	X		X	X	
11	X	X	X	X			X			X	X	X	X	X	
12	X	X	X	X			X			X	X	X	X	X	X
13	X	X	X	X	X		X			X			X		
14	X	X	X	X	X		X			X	X		X		
15	X	X	X	X	X		X			X	X	X	X		
16	X	X	X	X	X		X			X	X		X	X	
17	X	X	X	X	X		X			X	X	X	X	X	
18	X	X	X	X	X		X			X	X	X	X	X	X
19	X	X	X	X	X	X	X			X			X		
20	X	X	X	X	X	X	X			X	X		X		
21	X	X	X	X	X	X	X			X	X	X	X		
22	X	X	X	X	X	X	X			X	X		X	X	
23	X	X	X	X	X	X	X			X	X	X	X	X	
24	X	X	X	X	X	X	X			X	X	X	X	X	X
25	X	X	X				X	X		X			X		
26	X	X	X				X	X		X	X		X		
27	X	X	X				X	X		X	X	X	X		
28	X	X	X				X	X		X	X		X	X	
29	X	X	X				X	X		X	X	X	X	X	
30	X	X	X				X	X		X	X	X	X	X	X
31	X	X	X	X			X	X		X			X		
32	X	X	X	X			X	X		X	X		X		
33	X	X	X	X			X	X		X	X	X	X		
34	X	X	X	X			X	X		X	X		X	X	
35	X	X	X	X			X	X		X	X	X	X	X	
36	X	X	X	X			X	X		X	X	X	X	X	X
37	X	X	X	X	X		X	X		X			X		
38	X	X	X	X	X		X	X		X	X		X		
39	X	X	X	X	X		X	X		X	X	X	X		
40	X	X	X	X	X		X	X		X	X		X	X	
41	X	X	X	X	X		X	X		X	X	X	X	X	
42	X	X	X	X	X		X	X		X	X	X	X	X	X
43	X	X	X	X	X	X	X	X		X			X		
44	X	X	X	X	X	X	X	X		X	X		X		
45	X	X	X	X	X	X	X	X		X	X	X	X		
46	X	X	X	X	X	X	X	X		X	X		X	X	
47	X	X	X	X	X	X	X	X		X	X	X	X	X	
48	X	X	X	X	X	X	X	X		X	X	X	X	X	X
49	X	X	X				X	X	X	X			X		
50	X	X	X				X	X	X	X	X		X		
51	X	X	X				X	X	X	X	X	X	X		
52	X	X	X				X	X	X	X	X		X	X	
53	X	X	X				X	X	X	X	X	X	X	X	
54	X	X	X				X	X	X	X	X	X	X	X	X
55	X	X	X	X			X	X	X	X			X		
56	X	X	X	X			X	X	X	X	X		X		
57	X	X	X	X			X	X	X	X	X	X	X		
58	X	X	X	X			X	X	X	X	X		X	X	
59	X	X	X	X			X	X	X	X	X	X	X	X	
60	X	X	X	X			X	X	X	X	X	X	X	X	X
61	X	X	X	X	X		X	X	X	X			X		
62	X	X	X	X	X		X	X	X	X	X		X		
63	X	X	X	X	X		X	X	X	X	X	X	X		
64	X	X	X	X	X		X	X	X	X	X		X	X	
65	X	X	X	X	X		X	X	X	X	X	X	X	X	
66	X	X	X	X	X		X	X	X	X	X	X	X	X	X
67	X	X	X	X	X	X	X	X	X	X			X		
68	X	X	X	X	X	X	X	X	X	X	X		X		
69	X	X	X	X	X	X	X	X	X	X	X	X	X		
70	X	X	X	X	X	X	X	X	X	X	X		X	X	
71	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
72	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

**Appendix E: Samples of black box template for the general aspect composition
procedure of Test cases**

Variant	Input section			Output section	
	Input variables			Input actions	Expected results
	Primary file	Generic file	Point-cut expression	Select composition type	
1	tc1	tcGe	step.name = StepA	insertBefore	Context-specific inserted before StepA
2	tc3	tcGen	step.stepComponent.name = Component1	insertBefore	Context-specific inserted before StepA
3	tc5	tcGen	step.stepComponent.componentHost.name = Host1	insertAfter	Context-specific inserted After StepA
4	tc7	tcGen	step.stepComponent.host.name != Host2	insertAfter	Context-specific inserted after { StepB, StepC, StepA1, StepB1, StepC1}
5	tc9	tcGen	step.component.name != Component2	replace	Context-specific replaces { StepC, StapA1, StepB1, StepC1}
6	tc10	tcGen	step.component.componentHost.name != Host3	replace	Context-specific replaces { StepB, StepC, StapA1, StepB1, StepC1}
7	tc12	tcGen	step.component.host.name = Host3	insertBefore	Context-specific inserted before StepC
8	tc14	tcGen	step.name != StepB	insertBefore	Context-specific inserted before {StepC, StepA1, StepB1, StepC1}
9	tc16	tcGen	step.stepComponent.name != Component1	insertAfter	Context-specific inserted after {StepC, StepA2, StepB2, StepC2}
10	tc18	tcGen	step.stepComponent.componentHost.name != Host1	insertAfter	Context-specific inserted after {StepC, StepA2, StepB2, StepC2}

11	tc20	tcGen	step.stepComponent.host.name = Host2	replace	No match
12	tc21	tcGen	step.component.name = Component2	replace	Context-specific replaces {StepA2, StepB2, StepC2}
13	tc23	tcGen	step.component.componentHo st.name = Host3	insertBefore	No match
14	tc25	tcGen	step.component.host.name != Host3	insertBefore	Context-specific inserted before{StepA, StepB, StepC}
15	tc27	tcGen	step.name = StepC	insertAfter	Context-specific inserted after StepC

Appendix F: Samples of black box template for the security aspect composition

procedure of Test case

Variant #	Input section			Output section
	Input variables		Input actions	Expected results
	Primary	File	Generic	
			Select Security level	
1	tc5	tcGen	securedHost	Securing StepA(out), stepB(in)
2	tc4	tcGen	securedComponent	Securing StepA(out), StepB(in), StepC(out)
3	tc6	tcGen	securedService	No match
4	tc8	tcGen	securedHost	No match
5	tc11	tcGen	securedComponent	Securing StepC1(out), StepB(in, out), StepC(in)
6	tc13	tcGen	securedService	No match
7	tc15	tcGen	securedHost	No match
8	tc17	tcGen	securedComponent	Securing StepC1(out), StepA2(in), StepC2(out), StepC(in, out)
9	tc19	tcGen	securedService	Securing StepC(in, out)
10	tc22	tcGen	securedHost	Securing StepC1(out), StepA2(in), StepC2(out)
11	tc24	tcGen	securedComponent	Securing StepC1(out), StepA2(in), StepC2(out), StepA3(in), StepC3(out)
12	tc23	tcGen	securedService	Securing StepC(in, out)
13	tc28	tcGen	securedHost	No match
14	tc64	tcGen	securedComponent	Securing StepC1(out), StepA2(in), StepC2(out), StepC(in)
15	tc68	tcGen	securedService	Securing StepC(in, out)

**Appendix G: Samples of black box template for the general aspect composition
procedure of Case study**

Variant	Input section			Output section	
	Input variables			Input actions	Expected results
	Primary file	Generic file	Point-cut expression	Select composition type	
1	GetCustRegPage	genericSSL	step.name = getCustRefImgs	insertBefore	Context-specific inserted before getCustRefImgs
2	GetCustRegPage	genericSSL	step.stepComponent .name =webserver	insertBefore	Context-specific inserted before getCustRegPage
3	GetCustRegPage	genericSSL	step.stepComponent .componentHost.name = ServerProc	insertAfter	Context-specific inserted after {getCustRefImgs, getCustRegPage}
4	GetCustRegPage	genericSSL	step.stepComponent .host.name != ClientProc	insertAfter	Context-specific inserted after {getCustRefImgs, getCustRegPage}
5	GetCustRegPage	genericSSL	step. hostDemend <=0.7	replace	Context-specific replaces getCustRefImgs

6	GetCustRegPage	genericSSL	step.component.componentHost. opTime = 1.0	replace	Context-specific replaces { requestCustRegPage, getCustRefImgs, getCustRegPage, displayPage}
7	GetBuyConfirm Page	genericSSL	step.name = requestPGE	insertAfter	Context-specific inserted after requestPGE
8	GetBuyConfirm Page	genericSSL	step.stepComponent.componentHost. multiplicity >4.0	replace	Context-specific replaces { insertOrderLineRecord, updateItemstock, insertAddrRecord, matchAddrRecord, insertOrderRecord, requestPGE}
9	GetBuyConfirm Page	genericSSL	step.stepComponent.host. opTime != 1.0	replace	No match
10	GetBuyConfirm Page	genericSSL	step.component.name = database	insertBefore	Context-specific inserted before { insertOrderLineRecord, updateItemstock, insertAddrRecord, matchAddrRecord, insertOrderRecord}

11	GetBuyConfirm Page	genericSSL	Step.description= service=checkout	insertBefore	Context-specific inserted before sendrequest
12	GetBuyConfirm Page	genericSSL	step.component.host .name > ServerProc	insertAfter	No match
13	GetBuyConfirm Page	genericSSL	step. probability = 0.5	insertAfter	No match
14	GetBuyConfirm Page	genericSSL	step.stepComponent .name = eb	replace	Context-specific replaces requestBuyConfirm Page
15	GetBuyConfirm Page	genericSSL	step.stepComponent .name != database & step.stepComponent .name != webserver	replace	Context-specific replaces {requestBuyConfir mPage, getBuyConfirmImgs , requestPGE}

Appendix H: Samples of black box template for the security aspect composition

procedure of Case study

Variant #	Input section			Output section
	Input variables		Input actions	Expected results
	Primary file	Generic file	Select Security level	
1	GetCustRegPage	genericSSL	securedHost	Securing getCustRegPage(in), getCustRefImgs(out), displayPage(in)
2	GetCustRegPage	genericSSL	securedComponent	Securing getCustRegPage(in, out), getCustRefImgs(in, out)
3	GetCustRegPage	genericSSL	securedService	No match
4	GetCustRegPage	genericSSL	securedHost (ClientProc)	Securing displayPage(in)
5	GetCustRegPage	genericSSL	securedComponent (webServer)	Securing getCustRegPage(in, out)
6	GetBuyConfirm Page	genericSSL	securedService	Securing checkout(in, out)
7	GetBuyConfirm Page	genericSSL	securedHost (ServerProc)	Securing getBuyConfirmPage(in), sendrequest(in), getAuthorization(in), replyPGE(in,out)
8	GetBuyConfirm Page	genericSSL	securedComponent (PGE)	Securing requestPGE(in, out)
9	GetBuyConfirm Page	genericSSL	securedService	Securing checkout(in, out)
10	GetBuyConfirm Page	genericSSL	securedHost (DBProc)	Securing matchAddrRecord(in), insertOrderRecord(in, out), requestPGE(in, out)