ASPECT-ORIENTED MODELING FOR REPRESENTING AND INTEGRATING SECURITY ASPECTS IN UML MODELS

SRIVAS VENKATESH

A THESIS
IN
THE DEPARTMENT
OF
CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEM SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

JULY 2013

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By Srivas Venkatesh

Entitled Aspect Oriented Modeling For Representing and Integrating Security Aspects in UML Models

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Information System Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Amr Youssef (**CIISE**)	Chair
Dr. Thiruvengadam Radhakrishnan (**CSE**)	External Examiner
Dr. Chung Wang (**CIISE**)	Internal Examiner
Dr. Mourad Debabbi (**CIISE**)	Supervisor

Approved____

Chair of Department or Graduate Program Director

__________ 20____

Dr. Christopher Truman,
Dean Faculty of Engineering and Computer Science

Abstract

Aspect-Oriented Modeling for Representing and Integrating Security Aspects in UML Models

Srivas Venkatesh

Security is a challenging task in software engineering. Traditionally, addressing security concerns are considered as an afterthought to the development process and security mechanisms are fitted into pre-existing software without considering the consequences on the main functionality of the software. Enforcing security policies should be taken care of during early phases of the software development life cycle; this benefits the development costs and reduces the maintenance time. In addition to cost saving, this encourages development of reliable software. Since security related concepts will be considered in each step of the design, the implications of inserting such concepts into the existing system requirements will help mitigate the defects and vulnerabilities present in the system. Although integrating security solutions into every stage of the software development cycle, results in scattering and tangling of security features across the entire design. The traditional security hardening approaches are tedious and prone to many errors as they involve manual modifications. In this context, the need for a systematic way to integrate security aspects/mechanisms into the design phase of the development cycle should be considered.

In this work, an aspect-oriented modeling approach for specifying and integrating security aspects in to Unified Modeling Language (UML) design model is presented. This approach allows the security experts to specify generic security aspects and weave them into target software base model early in the software development phase. In contrast to traditional approaches, model-to-model transformation mechanisms discussed in this approach are designed to have an efficient and a fully automatic weaving process. This work further discusses additional components that are introduced into the weaving process. These newly introduced components allow the security experts to provide more appropriate security hardening concepts. Furthermore, the additional components are designed based on object-oriented principles and allow the security experts to exercise these principles in the model-to-model transformation. The additions to the weaver application are tested using the Session Initiation Protocol (SIP) communicator as a base model. The description of the additional components and the results of testing of the weaving process are discussed further in this thesis.

Acknowledgments

It is my pleasure to thank all those who made this thesis possible. First and foremost I thank God for granting me the chance to pursue my studies and for guiding me all through the way.

I owe my deepest gratitude to my supervisors and mentors, Dr. Mourad Debbabi, and Dr. Makan Pourzandi for introducing me to the field of academic research and providing me with the opportunity to conduct this research. Their constant guidance, support, and encouragement played the major role in making this research possible.

I would also like to thank all my colleagues in the *Model-Based Engineering of Secure Software and Systems (MOBS2)* project for their participation in this research and for making it a joyful experience. I would like to extend a special gratitude to Dr Djidjiga Mouheb, Ms. Raha Ziarati, Dr. Dima Ahladidi and Ms. Mariam Nouh with whom I closely collaborated in my research.

All my colleagues have greatly supported me during my first months in the lab, they provided invaluable assistance and above all they were great mentors and dear friends. My special thanks to Dr. T. Radhakrishan for motivating me to pursue my higher studies at Concordia University. My heartfelt thanks go to my family and my closest friends for their constant support throughout the journey which was invaluable.

Table of Contents

List of Figures

List of Tables

List of Algorithms

`

Chapter 1 Introduction

Today, computers have impacted all aspects of our lives and have improved the quality of life significantly. Computers have emerged into different aspects of our lives. Education, telecommunication, health care, transportation, military, and many other domains of our society depend heavily on computers and their applications. These spectrums of smart devices from simple gadgets like microwave ovens to complex satellites have made a profound influence on the lives of people. These technological innovations have made us more productive by keeping us connected even on the move: providing audio, video, and data connectivity. , We have computers working in the background unobtrusively aiding us in acquiring, storing, analyzing, understanding large amounts of data and appropriately exercising control so as to perform variety tasks be it: health monitoring, traffic control, financial transactions, personal entertainment; in a timely and efficient manner.

As these devices become smarter and more powerful the software controlling them also has grown in its complexity. Competition and market pressures often push out these complex systems without thoroughly verifying and validating them. Consequently, they have become susceptible to defects and vulnerabilities. Such high dependency on computers and software systems has facilitated the fact that huge amounts of critical information are now contained within these systems. A lot of personal and very sensitive information is available and is being transferred on data networks between servers and clients for example the software system in a military environment could be

`

dealing with top secret information, a health care package could contain private information, social media could contain tons of personal information. In all of these scenarios the common man has implicitly started trusting the systems. It becomes imperative to maintain and enhance the trust as more and more applications, which handles sensitive data are becoming operational every day. Software Engineering must now play a predominant role in the process of building secure and reliable software.

In today's computing world, security takes an increasingly predominant role. The awareness of security issues has increased among researchers in the software engineering community, which has led them to the understanding that although it is important to assure that software systems are developed to meet the users' requirements, it is also important to assure that these systems are equally secure. The industry is facing challenges in public confidence at the discovery of vulnerabilities, and customers are expecting security to be delivered out of the box, even on programs that were not designed with security in mind.

One of the primary reasons as to why the current approaches are unsatisfactory is, Software developers rely heavily on knowledge and experience. The construction of such complex software-related systems includes, in brief, requirements engineering, design, code implementation and testing, and in all these phase of the software design cycle, software engineers do not consider security as a major issue. Security being relatively new, the number of software security experts with the required level of knowledge who have dealt with a variety of security issues is still limited compared to

`

the existing number of software developers. Non security experts find it a challenging task to define the needed semantics and properties of its requirements.

Microsoft Security Intelligence Report for 2012 illustrates the vulnerabilities on applications that are uncovered since 2002. The report shows software vulnerabilities consisting of those that affect Operating systems, Applications, or both. It is difficult to draw a distinct line between operating systems and applications vulnerabilities. In Figure 1, vulnerabilities that affect both operating systems and applications are shown in red. In this setting, the security engineering of such software-intensive systems has become a major concern. This is emphasized by the fact that, in spite of significant efforts on software security from academia and industry, the scale and severity of security breaches have been increasing with no complete victory against attacks.

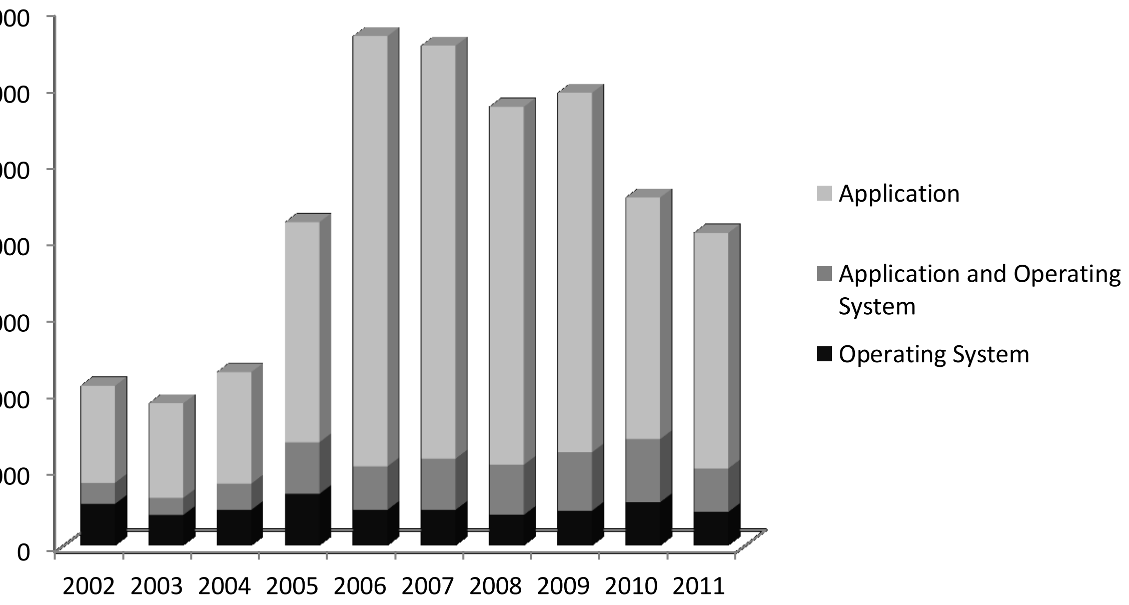


Figure 1 – Application and Operating System Vulnerability as Described in Microsoft Security Intelligence Report [1]

Security has been designed and implemented by non-security experts, conventionally; security concerns are addressed as an afterthought to the software being developed.

`

They are usually retrofitted into pre-existing designs without the consideration of whether this would jeopardize the main functionality of the software and produce additional vulnerabilities. The general practice for addressing security concerns is that the developers sprinkle the security mechanisms all over the original application. This often causes difficulties in tracking the changes and testing the application for vulnerabilities and defects.

It has been shown in recent research [2] that considering the requirements of Security during the early stages of software development results in more secure and cost effective solutions. It has been shown in Table 1 that costs of repairing a software flaw during maintenance is approximately 500 times higher than fixing it earlier in the design phase itself. For example, research conducted by Cigital shows an average cost savings of over $2.1 million (on a code base of 2 million LOC) when vulnerabilities are identified during development, where source code analysis is most often leveraged [3] as illustrated in Table 1.

Fixing Bugs Later				Fixing Bugs Earlier			
Stage	Critical Bugs Identification	Cost of Fixing 1 Bug	Cost of Fixing all Bugs	Stage	Critical Bugs Identification	Cost of Fixing 1 Bug	Cost of Fixing all Bugs
Requirements		$139		Requirements		$139	
Design		$455		Design		$455	
Coding		$977		Coding	200	$977	$195,400
Testing	50	$7,136	$356,800	Testing		$7,136	
Maintenance	150	$14,102	$21,115	Maintenance		$14,102	
Total	200	$22,809	$377,915	Total	200	$22,809	$195,400

Table 1– Cost of Fixing Vulnerabilities Later v/s Fixing Vulnerabilities Early as Described by Cigital Labs [2]

`

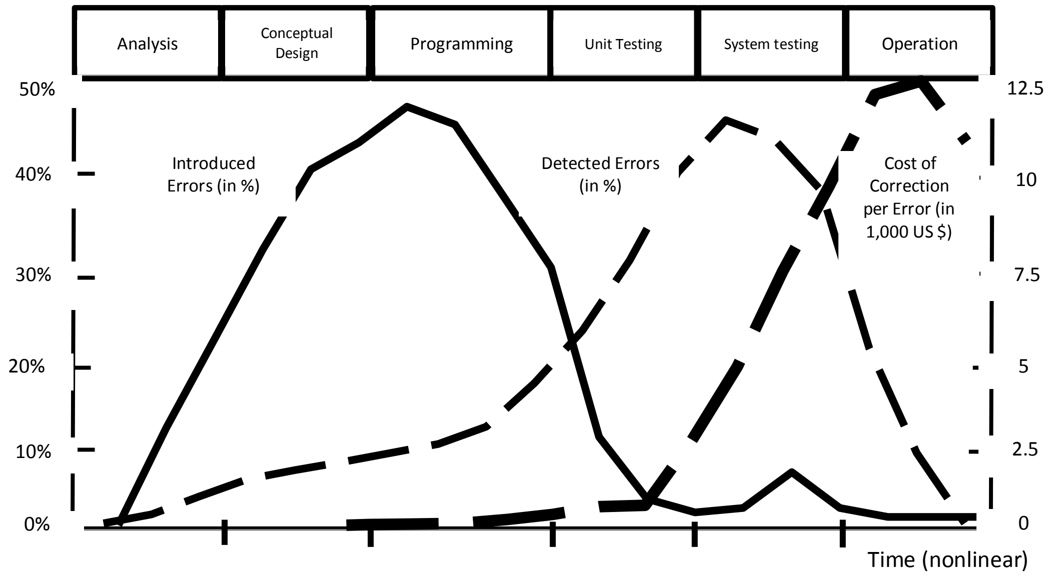


Figure 2 – Software Life Cycle and Error Introduction, Detection and Repair Costs [4]

Figure 2, concisely presents the manner in which the quanta of introduction of errors, detection of errors, and the cost incurred for repair (per error), and varies during the software development lifecycle of a typical application. By analyzing the graph we can conclude that higher the percentage of errors addressed during the design phase the cost of fixing remnant bugs that are uncovered in later stages decreases very rapidly. We can gather from Figure 2 that the percentage of detected errors during the design phase of the lifecycle is approximately 15% and the cost of fixing these flaws and vulnerabilities at this stage is very low compared to fixing these flaws during the testing phase.

Nowadays, the challenge is even greater when legacy systems must be adapted to fit into high-risk environments. Software maintainers must face the challenge to improve security of the program but are often underequipped to do so. In some cases, little can

`

be done to improve the situation, especially for Commercial-Off-The-Shelf (COTS) software products that are no longer supported, or their source code is lost. However, whenever the source code is available, as it is the case for Free and Open-Source Software (FOSS), a wide range of security improvements could be applied once a focus on security is decided. As a result, integrating security into software is becoming a very challenging and interesting domain of research.

In this thesis, we introduce an approach to tackle correcting the security flaws in the earlier stages of the design. In order to address security concerns throughout the development life cycle, we have adopted the use of "Security hardening". As the term suggests, we already have a working system and known security issues that have to be strengthened. To do the "security hardening" right, it should be conceptualized in the beginning of the system development process itself. Therefore, we use the prevalent concepts of Model Driven Architecture (MDA) [5] paradigm and the Unified Modeling Language (UML) [6].

MDA is an architecture using object technology, to distribute application integration, by guaranteeing reusability of components, interoperability & portability, basis in commercially available software [5]. Model-driven architecture is developed by the Object Modeling Group (OMG) to produce code from abstract models. These Models are designed to elaborate the system structurally and behaviorally using the OMG standardized modeling language such as UML.

UML is a standardized general-purpose modeling language in object-oriented software engineering used to specify, visualize, modify, construct and document the artifacts of

`

an object-oriented software-intensive system under development [6]. Systems Modeling Language (SysML) is a subset of UML, which is a general-purpose modeling language for systems engineering applications.

UML profiles are a specialized set of rules. These profiles are used when there is a need to define a new language to model a system that either restricts the number of UML elements or adds some constraints or modifies them while respecting the original semantics. UML elements can easily be customized by using extensions provided by UML [7].

Utilizing these concepts towards integrating security solutions at the software modeling level may result in the scattering and tangling of security features throughout the entire software models. To address these issues, Aspect-Oriented Modeling (AOM) [8]paradigm emerges as an appropriate approach for security hardening at the software modeling level by using these aspects at different stages of software development. The concept of AOM is to merge the existing concepts of aspect-oriented paradigm to find and define essential characteristics of crosscutting concerns in UML models, composition of such models that have common oriented aspects from a more abstract level. Using AOM we can assist security experts in designing security mechanisms in isolation without altering the logic of the application. Besides, using AOM, developers with limited security knowledge can systematically integrate those security mechanisms into their software models.

Aspect Oriented Programming (AOP) paradigm defines concepts like pointcuts, join points, advices and weaving.   A pointcut is an expression that allows the selection of a

`

set of points in the application flow to inject advices. The selected points that are matched for the targeted application are called join points. An advice is a piece of code, or behavior, that is injected into the target application when the application reaches the given join point during execution. Additionally, the process of injecting the advice into the application is commonly called weaving. Furthermore, other than advices, aspects contain a set of pointcuts and introductions.

This thesis is part of the research initiative supported by **Ericsson Canada** Software Research. This cooperation program aims at developing a Model-Based Framework for Engineering Secure Software and System (MOBS2). The targeted security concerns are: capturing security requirements, specification and design of security mechanisms, verification and validation of security properties/policies, and automatic generation of secure code. In the following section, we enumerate the objectives of this research work along with the proposed approach to achieve these goals.

1.1. Objectives

The main objective of this thesis consists of elaborating an approach for systematic integration of security requirements by defining new join points and advices and adding them into the MOBS2 framework in order to apply them during the design phase. This is achieved by expanding the specified UML profile by designing new aspects for security hardening. We also expand the weaving framework for the injection of security aspects into the UML design models. In particular, this thesis aims at:

`

- Elaborating a framework for specifying new security aspects and their systematic integration into UML models.
- Designing and implementing the proposed aspects.
- Validating the implemented aspects through various case studies.

In addition, the integrating procedures need to be transparent to the developer; this means that the proposed approach should be automated as much as possible in order to hide the complexity and enable a smooth learning curve of our approach.

1.2. Contributions

This section lists the main contributions of this thesis in relation to the objectives stated above. The main contributions of this thesis are:

- Expanding UML profile to specify new security requirements as aspects over design models.
- Elaborating the model transformation rules that allow for weaving the proposed security aspects into UML design models.
- Designing and implementing the proposed security aspects over UML models within Rational Software Architect environment.
- Conducting a variety of case studies to demonstrate the feasibility of the proposed aspects.

`

1.3. *Framework*

In this section we briefly describe the framework used in this thesis. The framework required an environment with a powerful UML modeler to aid the developer to create UML design models. For this purpose the environment employed is IBM Rational

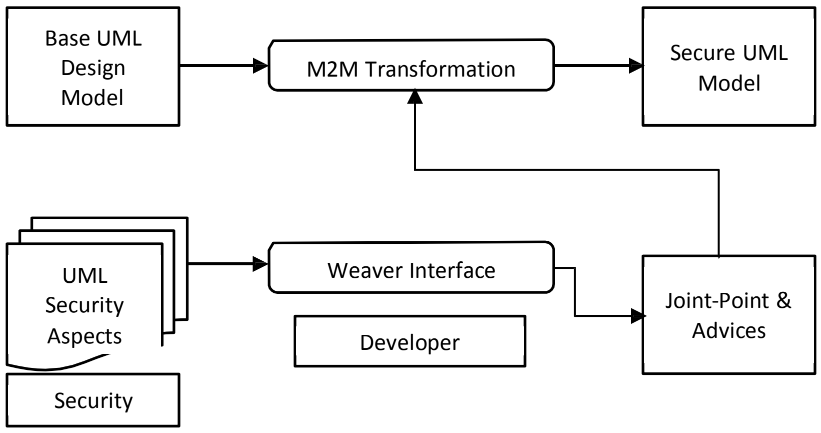


Figure 3 – MOBS2 Framework Overview [49]

Software Architect (RSA). The main components of the framework are illustrated in Figure 3. The framework comprises of three major components: The first one is the UML security aspects tailored by the security experts armed with a detailed list of all security requirements. Additionally, these experts design appropriate aspects into security aspect libraries in terms of UML concepts. This component is responsible to translate the security requirements into models and properties. The Developers utilize the second component to construct the rules for the weaving interface to create the joint-points and the advices that are required to incorporate the proposed security aspects into the base UML models. Also this component is utilized by the developer to design the pointcuts into the weaving interface using the Object Constraint Language (OCL) and the Query View Transformation (QVT) language. The third component is the model-to-model (M2M) transformation unit. This unit aids the developer in

`

transforming the unsecure UML models by adding security features into the UML models, and generating the secure UML model. This unit comes as part of the IBM RSA and its elements are accessed using the eclipse plug-ins.

1.4. Structure of the Thesis

The remainder of the thesis is organized as follows:

- Chapter 2 reviews the theoretical background literature of the concepts of modeling languages, aspect–oriented modeling, and the unified modeling language.
- Chapter 3 describes in detail the MOBS2 Framework and the various components. It also describes several UML artifacts and tools utilized to support the framework.
- Chapter 4 presents the design of new security aspects for both structural and behavioral UML models. The chapter also describes the semantics of these newly designed aspects and their outcome on the UML base models.
- Chapter 5 describes the impact of the aspects fabricated in this thesis. The case studies includes Role-Based Access Control (**RBAC**) for the security model, a cross-section of Session Initiation Protocol (**SIP**) Communicator base model
- Chapter 6 presents the summarizing conclusion of the thesis.

`

Chapter 2 Literature Survey

In this chapter, the primary concepts utilized are briefly explained and relevant work in this area is described. The core idea of the work is to support the incorporation of security concepts at the highest level that is the *model* level. The modeling language used in this thesis is UML as it is very popular and has been extensively adopted by the industry. The philosophy of the whole work is to let the security experts do their part of the hardening activity without requiring to have too much of in-depth knowledge of the functionality of the software. This naturally allows us to adopt the Aspect-Oriented paradigm. Security and functionalities can be modeled independently and then stitched together. In this context the chapter begins by discussing the concepts of software security. Then, we discuss UML modeling and its concepts. Afterwards we discuss the aspect-oriented paradigm concepts and how these concepts can be combined.

2.1. Software Security

Software security encompasses measures taken throughout the software's life-cycle to prevent breach of the security through flaws found in the design, development, deployment or maintenance phases of the software. The prerequisites for software artifacts to exhibit secure properties [9] are:

a. Transparency: Disclosure of all functions of the software to the user. i.e. there is no hidden agenda.

b. Obedience: The software follows only the directions given by its controlling entities and does nothing more than what is required to be done.

`

c. Purity: The software performs only the actions that it has advertised to the user and performs no unrelated functions.

d. Loyalty: The software serves only the interests of the authorized controlling entities and cannot be subverted to perform functions for some other entity.

Once it is ascertained that the software entity is “well-behaved”, the security features can be incorporated into the software [10]. Typical attributes of secure software that are visible to the user are [10]:

a. Authentication – Ensure that the user of the software (either machine or human user) is the entity it claims to be.

b. Authorization – Ensure that the authenticated user has sufficient privilege to perform the intended function.

c. Audit – Be able to trace every action performed by the software entity by maintaining appropriate logs of all transactions (possibly critical ones only).

d. Confidentiality – Disclose the sensitive information only to the authorized entities.

e. Integrity – Prevent unauthorized or improper modification of systems and information.

f. Dependability :

    i. Reliability – The software performs its assigned tasks without any failure and provides the same results for the same set of inputs all the time.

    ii. Availability – The software is available all the time to do its intended function.

`

g. Non-Repudiation: Maintaining irrefutable proof of a transaction for both parties that have participated in the transaction. This requires audit trails to be maintained [11].

2.2. Unified Modeling Language (UML)

The development of any large non-trivial system is a very complex task. Typically, large systems will use diverse sets of agents for completing their tasks. To satisfy these tasks the system has to be utilized effectively. At the beginning of the project, the requirements of each stakeholder will have to be captured and documented. At this stage, in order to ensure that requirements have been captured accurately and even during the design and implementation phase, it is a commonly accepted engineering practice to build models. These could be either hard models like small scale versions of planes or soft models like building plans etc. These models are created to study and analyze various characteristics of the target system in order to ensure that all critical requirements can be satisfied once the system is realized. Various types of models of the system can be created each highlighting a set of aspects of its proposed behavior. These can then be vetted by the stakeholders and would serve as an appropriate launch point for the detailed development phase. With the advent of high performance computer-based tools and their easy availability, it has become practical to develop high fidelity models and even simulate them in order to understand and capture the critical performance parameters. Building large Software systems is no different. The concept of modeling the intended system behavior has come into wide spread use after the

`

OMG standardized and promulgated a modeling language; known as the Unified Modeling Language (UML) in 1997.

It was called the unified modeling language as predated to it there were different modeling languages being used by different research groups; most active proponents of this were Jim Raumbaug, Grady Booch and Ivan Jacobson [12] who had evolved their own modeling notations named OOSE, Booch, and OMT [12], respectively. UML is a standard language for specifying, visualizing, building the software artifacts that constitutes the system being developed. It also serves as a good documentation mechanism. Due to the existence of this standard, it has now been adopted as the ideal vehicle for technical exchange of design information by groups/teams of software developers. UML however, neither is a methodology, nor does it prescribe any particular process, nor is it a programming language.

To understand UML, a conceptual model has to be formed that portrays the basic elements. These elements are known as UML basic building blocks. The building blocks of UML are: *Things* which are the abstractions that are the objects in a model. Objects are the basic building blocks used to create a well-formed model of the application; *relationships* tie these objects together. Relationships are the basic relational building blocks of UML, and lastly *diagrams* group the interesting collections of the objects. Diagrams are the graphical representations of these elements mostly rendered in a graph where the vertices are the objects and the arcs are the relationships [12].

A system's architecture is the most important artifact that can be used to manage the different agendas from several stakeholders, and so to control the iterative and

`

incremental development of a system throughout the software development life cycle. The UML architecture should portray the organization of a software system, a proper selection of elements and their interfaces to compose the system along with the behavior in collaboration and finally the architectural style that guide the organization. Figure 4 illustrates the views that best describe the architecture of a software-intensive system each of these views is a projection into the organization and structure of the system, focused on a particular aspect of that system.

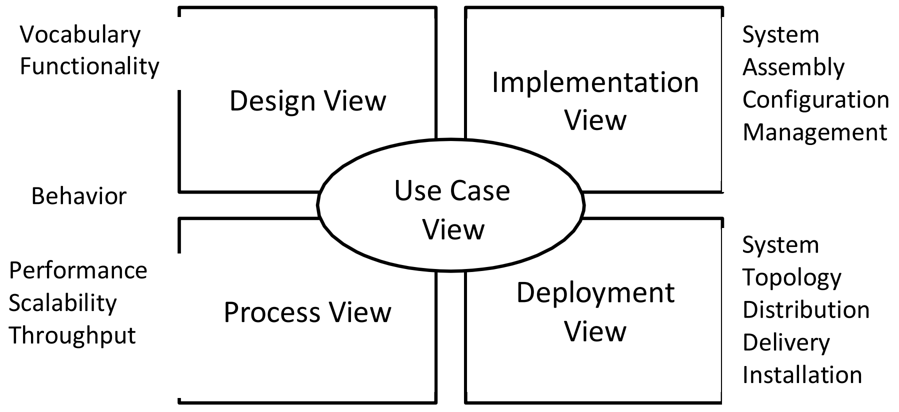


Figure 4 - Different UML Views [12]

As illustrated in Figure 4, the use case view of a system describes its behavior as seen by the different users. With UML, the static aspects of this view are captured in use case diagrams, class diagrams, package diagrams, object diagrams and many other diagrams; the dynamic aspects of this view are captured in the interaction diagrams, state chart diagrams and activity diagram. The design view of a system encompasses the functional requirements of the system objects that form the vocabulary of the system. The design views are captured in class diagrams and object diagrams. The process view of a system encompasses the process that forms the system's parallel and synchronization mechanisms. The implementation view focuses on the components and files that are

`

used for the purpose of configuration and assembling the system and is depicted in the component diagrams. The deployment view focuses on the system topology on which the built system has to run. This view is depicted by deployment diagrams. Each of these views can stand alone and show different perspective to different stakeholders. With these separate views the stakeholders can focus on the issues that concern them.

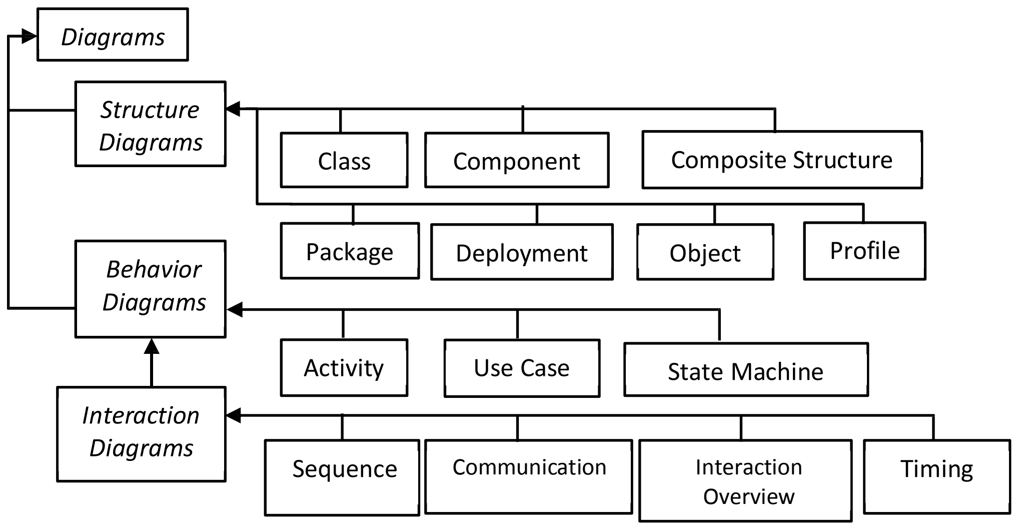


Figure 5 - UML Diagrams [12]

Figure 5 shows the different UML diagrams that are used by these stakeholders.

As illustrated in Figure 5 , UML Diagrams are of several different types they are [12]:

- *Use case diagram* captures system functionality as seen by users. It is built in early stages of development by system analysts and domain experts interactively with the users. Its purpose is to specify the context of a system; capture the requirements of a system; validate a system's architecture, drive implementation, and generate test cases.
- *Class diagram* captures the vocabulary of the system. It depicts the model elements and their dependencies. This is built by analysts, designers, and implementers. The

`

main purpose is to gather the required concepts and associate unique names for these entities; depict the collaborations that are required between the various entities and the constraints imposed on these entities, and finally group the entities that together will form the logical database schema for the system.

- *Object diagram* captures the instances of the various data entities and inter-relating links. This is built by the analysts, designers and implementers. Its purpose is to depict the structure of the various objects and data entities in the system and capture the instances of the interaction between the various entities as snapshots.
- *Component Diagram* is developed by architects to provide information/specification to the programmers. It maps the logical structure to the physical structures that have to be implemented. The main purpose is to specify the physical database and construct the executables.
- *State Machine diagram* captures the "states" of the various objects/components of the system. It is also built by the architects and specified to the programmers. The main purpose is to define the various operating modes i.e. the dynamic behavior of the system entities; define the data/events that effect transition between the various states/modes of the object and also define the start state of each object and hence the system.
- *Sequence diagram* captures the interactions between the entities in a chronological order and also depicts the dependencies between the various events. It also captures the dynamic behavior of the various system entities. It is usually created by the architect and specified to the programmer.

`

- *Activity diagram* depicts the whole system as a composition of activities and shows the flow/sequence of these activities that will be pursued in order to achieve the functionality.
- *Deployment diagram* maps the complete software system onto the physical hardware structure. It captures the various physical nodes and their interconnections that work collaboratively in order to perform the tasks.

Table 2 provides a brief description of all these diagrams. One can note that each diagram has a different purpose and a precise strength for performing particular tasks inside the software development process. Choosing the right set of diagrams to model a system is very important to make the design understandable and approachable.

UML Diagrams	**Represents**
Use-Case	System functionality from the user's viewpoint
Activity	A sequence of actions of a flow within the system
Class	Class, entities, business domain, database
Sequence	Interactions between objects
Interaction Overview	Interactions at a general high level
Communication	Interactions between objects
Objects	Objects and their links
State Machine	The run time life cycle of an object
Composite Structure	Component of object behavior at run time
Component	Executables linkable libraries
Deployment	Hardware nodes and processor
Package	Subsystems, organization units

Table 2 - UML Diagrams [13] [14]

`

UML is defined as an open-ended modeling language, which could be extended with new entities i.e. building blocks, new properties, and semantics so as to customize it to various problem domains. The extensibility mechanisms defined for UML are: stereotypes, tagged values and constraints [13] [15], and they can be described as follows:

- *Stereotypes*: These define an extension mechanism based on which one can derive a new model element in the lines of an existing one. It can inherit some properties of the parent element and also have some very specific properties meaningful for the particular problem domain. With these primitives and basic modeling elements the problem domain can be better captured and visualized. This forms the basis for creating various profiles of the system. For example, exceptions in Java can be cast as a stereotype that can then be customized to the behavior that is meaningful in the particular problem domain instead of "one solution suits all" kind of handling of exceptions.
- *Tagged Value*: This specifies attributes of the model element and also values associated with them. This could be associated with stereotypes so that the extension based on the stereotype can also inherit these values. This should be treated more as metadata as it is not instance dependent. Examples are association of say a particular language compiler with the object or versioning of the element, which does not depend on a particular instantiation of the object.
- *Constraints*: These are properties, which define assertions that have to be true at all instants of time with respect to a particular modeling element. The UML building

`

blocks can be enhanced with new rules or modification of existing rules with this feature. This, for instance, could be a method of capturing timing constraints or deadlines for various tasks in a hard real-time system. OMG has defined the Object Constraint language (OCL) to express these constraints on models elements.

2.2. Aspect-Oriented Paradigm

The idea of Separation of Concerns (SoC) has its origin in the evolution of the ideas of encapsulation so that visibility is limited to the interfaces required for manipulating the information without worrying about the internal architecture of the particular entity [16]. The aspect-oriented paradigms and specifically the Aspect-Oriented Software Development (AOSD) methodology enhances these concepts and presents modularization in ways which ease the task of cross-cutting concerns. A concern in this context is basically a property that is critical or important for a particular stakeholder. A particular system will have different stakeholders and each of them will have certain concerns. Core concerns of a module can be usually realized using standard OOP techniques [17] where this concern can be implemented in a single module with adequate encapsulation. However, there are certain functionalities of the system such as performance monitoring, concurrency control, transaction management, and security that cannot be captured or localized in just one module and will have to be implemented in various modules of the system. This concern therefore is a cross-cutting concern that spans quite a few modules and gets implemented in a distributed manner. Conventional way of implementing it would have a sprinkling of the code all over the modules and leads to tangling of the code if a number of cross-cutting concerns have to

`

be addressed. The concept of aspect-oriented programming evolved to address such cross-cutting concerns. Here, each of these cross-cutting concerns can be viewed independently of the actual functionality of the system and appropriate design and implementation can be evolved to satisfy this concern. A language, which has constructs for specifying the core concerns as well as the cross-cutting concerns of the system is necessary for supporting this design methodology.   The language should also be implemented so that the code that will be generated to address the above concerns will adhere to the language specification and will translate to code, which will execute as desired. The idea is that the concerns are specified and then each of these implemented concerns will have to be weaved along with the modules that implement the core concerns to realize the final system. This requires that a set of weaving rules be defined based on which the modules can be stitched together. The core and the cross-cutting concerns are normally implemented using standard object-oriented languages like C++ and Java. The weaving rules will have to specify "where and what" has to be weaved into the particular section of the code so as to achieve the desired functionality. The system exposes certain points in its execution such as execution of methods, communication between tasks, exceptions, creation of objects or destruction of objects, etc. These identifiable points are defined as *join points*. Next, support is required for selecting the particular join-point into which the code corresponding to the cross-cutting concerns will have to be weaved. Such specification is called *pointcuts*. Once the join point has been identified then the particular code, which modifies the behavior of the code to take care of the cross-cutting concern, called the advice, will have to be

`

specified. This advice can be invoked before the execution of the code at the join-point or after or instead of it. Such power of expressiveness allows fairly complex behavioral modifications to be implemented in the system. The advice that needs to be stitched together with the main code will itself be captured in an entity called the *aspect.* To summarize, the implementation of the cross-cutting concerns are captured as advices in modules called aspects, the points in the code where the advice has to be weaved are specified as pointcuts and the places where the system is amenable to behavioral modification are the join-points of the system. The primary driving factor for adopting AOP for developing secure software is that security is normally a cross-cutting concern. An introduction of *information access control* may have to be enforced wherever the information is being accessed in the application .For example, in a banking system shown in Figure 6. A simple banking application showing the working of an ATM module, connecting to a bank module to the provide account holder information. The accounting, ATM and database modules may have to utilize the services of the security module in a number of places. With the result though the particular interface to the security module might be defined by an API, the calls to this Security API is sprinkled all over the application in various modules. This leads to tangling of the code. Also modifications to the APIs may result in modifications to a number of modules in the entire application.

`

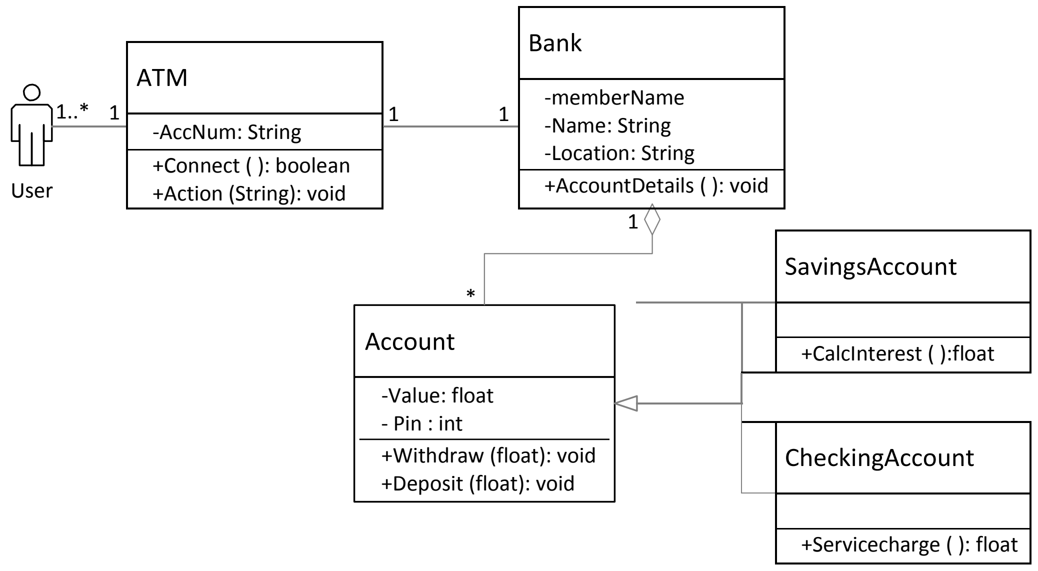


Figure 6 – Simple Banking Module [17]

Adoption of AOP would be very natural in such a situation. In this case the functionality to satisfy the cross-cutting concerns will be in one module called security module that logically interacts with the other modules shown in Figure 7.

The *join points* in each of these modules will be specified and when encountered, based on the *pointcut* criteria, the corresponding *advices* contained in the *security aspect* will be woven into the modules. All interactions with the security module per-se are contained in the particular security aspect. Any changes done to the security module will only result in the modification of the security aspect, but not the other modules.

In spite of the inherent advantages of the AOP methodology for addressing security concerns, there are certain drawbacks [19] that are present in the system namely isolation of faults is difficult as the fault could be in either the source code of the main functions, the aspect or the woven code.

`

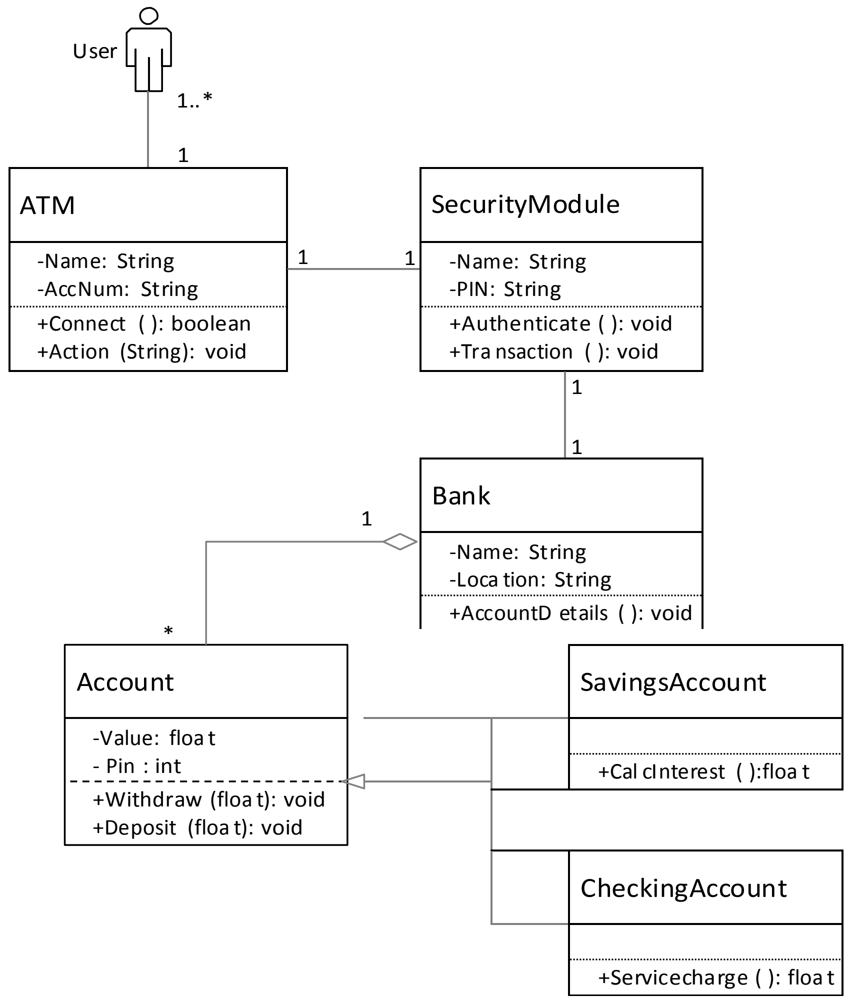


Figure 7 – Banking Example with a Security Module [18]

Another issue is when there are many to many relationships between aspects and the primary modules then understandability becomes difficult as all interactions will have to be understood by each of the implementers of the module. In spite of these deficiencies, the AOP methodology has come to be widely accepted in the recent past. Keshnee et al. [19] have established various categories of aspect-oriented security research.

The major categories identified are: access control and authentication, cryptographic controls, information flow controls, protection from intrusions, security kernels,

`

verification and security software engineering. In the following, we briefly discuss each category:

- *Access Control and Authentication*: Access policies have to be enunciated and implemented so that every access to the "secured entity" will be controlled using these policies. Various types of architectures for implementing this are proposed [20] defining an "Enforcement Agent" who will be approached for mediation when an access to the "secured entity" is attempted. Another proposed approach is to specify the "secured entity" in a container and manage access to the container using the predefined policies. Research in this area has also focused on discretionary access controls [21], role-based access control [22] and mandatory access control [23].
- *Cryptographic Controls*: In this category, the information that has to be accessed, modified or transferred is kept in an encrypted form. This ensures that the data is visible only for the people with appropriate access rights as they alone will have the method of decrypting the code, thus maintaining its confidentiality. Assurance can be given about the authenticity of the information, and its integrity too. The AOP methodology enables the data-items to be securely hardened using appropriate encryption aspect accessing a security module. Without having to modify the application module, security measures can be enforced [19].
- *Information Flow Controls*: These will basically ensure that information will be available only to the authentic software objects. The information flow can be between objects in the same node or objects resident on different nodes. AOP

`

defined point-cuts can specify appropriate join points before or after data transfer. Advices at these join points can verify the authenticity of the destination or source of information as the case may be and also perform sanity checks in order to ensure that information meets the constraints that might be imposed on these data transfers.

- *Protection from Invasive Software*: AOP has been adopted in performing this function where an advice when invoked will perform self-checks so as to ensure that the target module has not been modified. This could be done using verification techniques, which may depend on the various types and instances of invasion against which the software has to be hardened.
- *Security Kernels*: A security kernel is defined to be the hardware/software component that implements the concept of a reference monitor. The objective of the security kernel is to integrate the security mechanisms into a part of the operating system [24]. Using the concept of AOP, it is possible to upgrade or modify the security enforcing mechanisms dynamically as needed to ensure the security of the system [25] [19].
- *Verification*: AOP techniques have been used in the process of verification where it has to be ensured that the data entities generated by the system obey the constraints imposed on them. This could be implemented by creating specific validation agents who verify the compliance. These validation agents can be introduced using the AOP paradigm and then can be modified in order to enforce other constraints or policies without any effect on the application modules [19].

`

- *Security Software Engineering*: This deals with all phases of the lifecycle of software development straight from the elucidation of requirements to the implementation and deployment [19]. AOP has been found very useful because using this methodology; the application modules are divorced from the security module as there is a strong separation of concerns. The security module implements all the security mechanisms and the security aspect module contains all the advices that are weaved into the application module appropriately. Since the security issues are not handled in multiple modules, it will be easy to reason about the correctness of the security system during all phases of the development lifecycle.

2.3. Aspect-Oriented Modeling with UML

Over the last decade, AOM is moving from being a work of curiosity to actual usage where a number of diverse stakeholders have to contribute in building the system. The adoption of AO principles early in the development process chain will allow subject experts to function fairly independently from the beginning. They can independently conceptualize and easily prove correctness even at higher levels of abstraction. UML, being a widely accepted high-level object-oriented modeling language, becomes a natural choice for supporting the AOM methodology. In the following, some of the important contributions in the area of AOM using UML are reviewed.

Theme [26] is one of the approaches, which proposes the adoption of AOM straight from the *Requirement analysis* phase. Theme has two components: Theme/Doc and Theme/UML. Theme/Doc supports visualizing requirements specifications and their inter-relationships. It enables the refinement of views of requirements so that cross-

`

cutting functionalities can be defined and the point-cuts identified. The main focus has been to discover ‘aspects’ from requirements. The requirement specification is analyzed and the actions and their interconnections are extracted from it. The requirements and actions are usually many-to-many relationships. The shared requirements are the aspects. Once all the requirements are analyzed and mapped, they are split into groups. Groups that are self-contained constitute the base, while the groups that have links with other groups show the cross-cutting concerns. Theme/UML models features and aspects of the system along with the scheme of combining them. The main achievement is that traceability from requirements to views of requirements (Theme/DOC) to UML models (Theme/UML) is supported. Fuentes et al. [27] have proposed certain generic UML2 meta-models. Each meta-model is a composition of four core packages namely, entities for defining aspects, join points for identification of execution points at which interception can happen, AO-behavior for specifying the behavior that needs to be introduced at the join points, and aspect composition rules, which define the relationship between the join points and the aspects. Extra packages can be added in order to model for instance the introduction of new methods to classes, and adding interfaces to classes etc. These can supplement the core packages to better model the system. They have also mapped the UML-2 elements into these essential entities of AOP. The containers and components can be used for modeling aspects. Stereotypes or classifiers can be used to model the behavioral and structural features of aspects. The relationship between aspects and the base model can be depicted using association and dependencies. The concept of ports can be used to model the relationship between

`

aspects and the classes. Hooks can be used for defining join points. These define some possible mappings between the UML2 constructs and those that are needed to implement AO in the software design phase. However, neither implementation of these concepts, nor a suitable development environment has been reported. Similarly, Gupta et al. [28] have proposed extensions to UML based on the meta-model concept for providing support for aspects and modularization of cross-cutting concerns by adding elements to UML to represent point-cuts, join-points, advices, classes and aspects. However, their work is limited to a few of the UML artifacts and they do not provide any integrated environment for their work. They have however demonstrated the efficacy of their approach using a case study. Cottenier et al. [29] have studied the problem of cross cutting concerns in the context of distributed and embedded systems. They also promote the concept of meta-model to represent composition semantics. They have defined a model weaver, called SDL, for state-charts, which serves primarily to simulate and validate the design of state-charts. The work presented in [29] is confined to only few artifacts of UML. Aldawud et al. [14] define generic profiles in UML to support AOSD. Profiles are basically predefined set of extension mechanisms for a particular domain, technology or methodology. The typical extension mechanisms of stereotypes, tagged values, and constraints have been utilized to evolve profiles specifically for satisfying the needs of AOSD. They classify aspects as synchronous and asynchronous. Synchronous aspects are those for which the associated advice modifies the behavior of the core class. The asynchronous ones are basically self-contained functions which are triggered by a particular event during the execution of the core class. Relationships are

`

defined as dependencies and associations. They have also defined profiles for collaboration diagrams and state-charts. They have used constructs of AspectJ [30] for realization of these concepts. The authors have stated that these profiles have enabled bridging the gap between OO and AO. However, standardization requires a lot more effort. Bustos et al. [31] propose class diagrams and sequence diagrams to represent the static and dynamic views. State diagrams are used to represent the advice. They propose this to build formalism into the process so that proving correctness gets simplified. They propose to introduce these formalisms into the early stages of the software development cycle itself. Similarly, Pawlak et al. [32] defines point-cuts as relations to pointcuts, and aspect classes. The aspect classes would contain the advice that needs to be incorporated at the appropriate point-cut. The point-cut relationship establishes the mapping between the base class and the advice. They have defined the concepts at a high level and are planning to support the full AO model lifecycle in future. Basch et Al. [33] propose a scheme where aspects and components are separated using encapsulation into different packages. Each aspect package is self-contained and has its own class. They also proposed an additional modeling element for depicting the join points in the main application model. These were recommended for incorporation into the standard. The concepts have been explained using graphical representations but this process of actually implementing them into a tool has not been dealt with in the paper. The paper also does not report any activity towards the development of a complete development environment using these concepts. Kande et al. [34] argue the case for viewing the cross-cutting concerns as 'first class citizens' and introducing models for

`

capturing ‘aspects’ at the UML-level. They examine suitability of UML for the purpose of software system modeling. UML supports model refinement with features like stereotypes, tagged values, and constraints. Constraints are expressed using OCL. Suitability of UML for AOP concepts is examined by drawing parallels between AOP and UML constructs. This has been examined in the context of ASpectJ and its constructs. Method calls are the most convenient and intuitive join points. Introduction of interceptor classes can help in injecting advices at the appropriate points. Mediation is done at connection points into which the advices can be weaved during weaving. This will modularize even at higher levels of abstraction. They propose definition of new stereotypes of UML to enable new aspect classifiers.

Khan et al. [35] have comprehensively surveyed important research work performed in this domain of providing extensions of UML for modeling AOS recently. They have defined certain evaluation criteria for determining the suitability of the various methods to provide all the desirable features. These are: Coverage- the extent to which AOP techniques are covered by the constructs; Supported AOP constructs; UML Artifacts- a list of artifacts that are extended to model the AOP SW; Case study- whether certain case studies have been implemented using the techniques; Adherence to UML standards; Modeling of the weaving process and supporting language. The authors have studied the existing approaches and classified them based on the above criteria. They have come to a conclusion that none of the proposals meet all the above listed criteria in their entirety.

`

2.4. Query/View/Transformation (QVT)

QVT (Query/View/Transformation) is the standard defined by OMG for model transformation. It consists of three components: two declarative (relations and core) and one imperative (operational mappings). The relations language implements the transformation by providing links that identify relations between elements in the source model to elements in the target model. The core language is also a declarative language; it is simpler than the relations language. It is actually used to specify the semantics of the relations language. These two languages are good for simple transformations where the source model and the target model have a similar structure. However, when it comes to more complicated and sophisticated transformations where elements in the target model are being built with no direct correspondence with elements in the source model, declarative languages can be a limitation. Thus, the need for an imperative language becomes a must. Therefore, QVT proposed the third language, which is the operational QVT.

2.5. Related Work on Weaving and UML Secure Design

In this section we review the work closely related to secure design. We begin the section by discussing the current weaver tools that are developed. Then further on, we discuss the work related to UML Secure Design and AOM.

2.5.1. Weavers

Various approaches have been proposed for weaving aspects into UML design models. Weaving is the process of injecting the advice specified at the identified join points

`

selected by pointcuts. In Figure 8, an overview of a weaver tool is illustrated. Commonly, the weaver tool has the base models, aspect models, and the identified joint points as inputs. The combined output, which is the woven model, is produced for further analysis in the software development lifecycle.

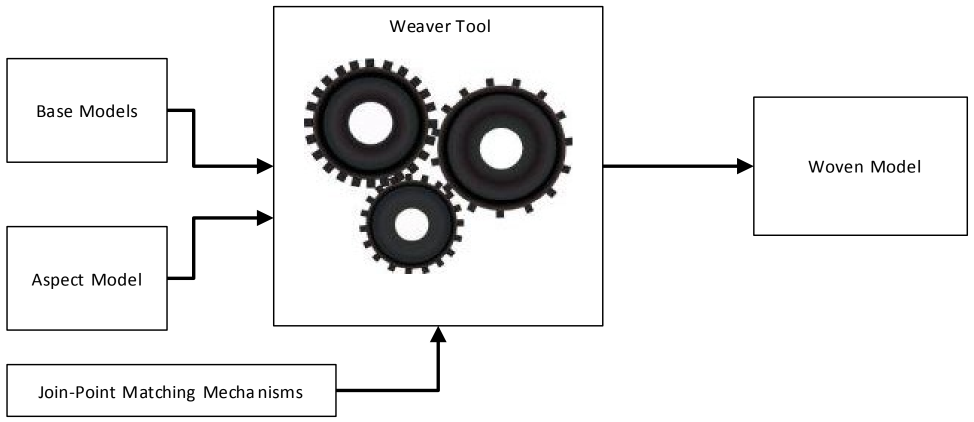


Figure 8 – Weaver Overview [36] [37]

In this section, we present the relevant work done on the weaver tool. Zhang et al [38] present Motorola WEAVR; an AOM plug-in for weaving aspects into executable UML state machine models. The weaving process involves two phases: advice instantiation and advice instance binding. During the first phase, advices are instantiated based on the pointcuts defined. Then, the matched join points are linked to their corresponding advices. During phase two, the aspects are woven into the base models. Motorola WEAVR supports two types of join points that are action and transition join points. Aspect interference is handled by allowing precedence relationships to be specified at the modeling level. This weaver is tool-dependent, not portable and lacks graphical representation of the woven models. Fuentes and Sanchez [36] proposed a weaving process implemented as a set of XMI (XML Metadata Interchange) representations of

`

the models. Using XPATH (for join points selection) and XSLT (for advices injection), a XMI representation is generated from the output model. This weaver does not support graphical representation of the woven model. However, this approach targets only executable UML models.

Morin et al. present the GeKo [39], a generic model weaver. GeKo supports weaving of class diagrams, state machine diagrams, and sequence diagrams. The weaving is implemented as model transformations using Kermeta language [83]. The GeKo approach is based on the definition of mappings between the different views of an aspect, based on graphical syntax associated to the domain specific frameworks. The tool uses Prolog-based patterns matching to automatically identify the join-points. GeKo, which is still under development, keeps the graphical representation of the weaving between an aspect model and the base model.

Groher and Voelter [40] present the XWeave; a weaver that supports the weaving of models and meta-models. The weaver is implemented as a model-to-model transformation using OpenArchitectureWare and it is based on the Eclipse Modeling Framework (EMF) [40]. The tool selects the join points by: matching the name of the element in the aspect model with the element in the base model or by explicit pointcut expression specified in the OAW to select elements of the base model. XWeave is limited only to the addition of new model elements to the base model. It does not support removing or replacing existing base model elements. In summary, most of existing work on model weaving is either tool-dependent, based on the XMI representation of the models, limited in terms of the supported diagrams, limited in

`

terms of the supported aspect adaptations, or consider only a restricted join point model.

2.5.2. UML Secure Design and AOM Relevant work

One of the important set of applications that is driving the AOM methodology is designing and implementing secure software. In this section, we review some of the relevant work in this area. Harikrishna et al. [41] have proposed a complete framework for building secure software using the AOM approach, using SAM (Software Architecture Model) for specifying the base model and the aspects. Software architecture model use *Petri-nets* for modeling combined temporal logic. It also supports hierarchical models, which allows the modeler to change the level of abstractions. Join points are mapped to "connectors" in the SAM model. The work is still at a very conceptual stage and the adoption of modeling scheme, which is not very familiar to SW practitioners, might become a severe bottleneck.

Matheson et al. [42] define patterns for expressing security concerns like authentication, authorization, and data privacy. Security concerns are not homogeneous. Different levels of security may have to be enforced for guarding different assets and sometimes for the same asset as well. Each concern is associated with a design pattern, which is represented using the class and interaction diagrams. It is possible to have alternate design patterns to address a particular concern. Catalog of design patterns are maintained. The order of composition of these patterns is very critical for proper functionality of the software. The concepts have all been proposed at a high-level of abstraction and have mostly been proposed in the context of access

`

control. However, these concepts have not yet been mapped onto any particular technology for addressing the implementation aspects. Matheson et al. [42] address the issue of multiple independent aspects combined to form the application. One of the main issue addressed is the conflict of various aspects interacting with the main model. Process guidelines that are applicable in general to any AOM have been provided. These concepts can be adopted in various frameworks.

Win et al. [21] have successfully defined necessary requirements for a good AOP environment: Definition of an easy and optimal design process, ability to monitor the performance of the various application modules even in integrated systems, support for testing and debugging, which is complicated due to integration of the aspects into the application code, ensure no security holes in the process. AspectJ has been used for the development of AOP techniques.

Doan et al. [25] describe the evolution of a formal framework for modeling design states. The authors propose a checker component to ensure that design state instance meet the constraints. They also extend the UML design framework with MAC, lifetime and RBAC constraints. Moreover, they developed a security satisfaction design program, which basically acts like a watchdog during the development process and gives information on the security requirements that are being complied with in the current design state. The same can be used in post-development in order to check compliance with a particular set of requirements. Such tools are very helpful when developing complex systems and different versions have to be released. The total integration of

`

these into a tool framework has not yet been implemented. This also does not incorporate the ideas of AOM.

`

Chapter 3 A Framework for Model-based Secure Software and System Engineering (MOBS2)

The concept of AOM and its applicability or suitability to separate the cross-cutting concerns from the actual function in the context of security hardening has been introduced. A survey of the literature also revealed that there was no comprehensive tool, which could support all phases involved in the practical implementation of the above methodology [37] [43]. The MOBS2 programming environment was conceptualized to satisfy the above requirement of a comprehensive tool. In the following sections, the features of the MOBS2 tool suite are related to the phases of the proposed methodology. The MOBS2 project was a joint research project pursued by Concordia's Information Systems department with Ericsson as the Industrial partner.

3.1. Requirements of the tool suite

In this section, the various phases of the development methodology are outlined and the required tool support for each of these phases is identified. As stated in chapters 1 & 2, the key idea is the separation of concerns for the various non-functional requirements. Though the non-functional requirements are more add-ons they are no less important. Their deterministic and reliable operation would actually enhance the acceptability and usability of the functional package.

In short, the requirement of the tool will be to support the AOM at a very high level. The tool was envisaged to provide the required support for various models of the system as specified in UML. Typically, both the structural and behavioral UML models are used to

`

define the system. The most used and preferred model among the structural models is the "Class diagram", while the preferred behavioral models are the Activity diagram, communication diagram and the state machines.

The mechanisms that will be provided in the tool are general enough to be applicable across the various models. In order to support the methodology at a minimum the following are required:

- An unambiguous specification of conditions to define the pointcuts.
- A method for identifying the positions, possibly multiple, in the base model where the conditions defined in the point cut are satisfied. This may need to be dynamically evaluated. The positions that satisfy the point cut conditions are defined as *'join points'*.
- The additional/alternate functions that need to be executed at the join points, which are called "adaptations" to the base model.
- Merging the "adaptation" models with the base model. This is called "weaving".

The MOBS2 framework described above is shown in a high level diagram in Figure 9

`

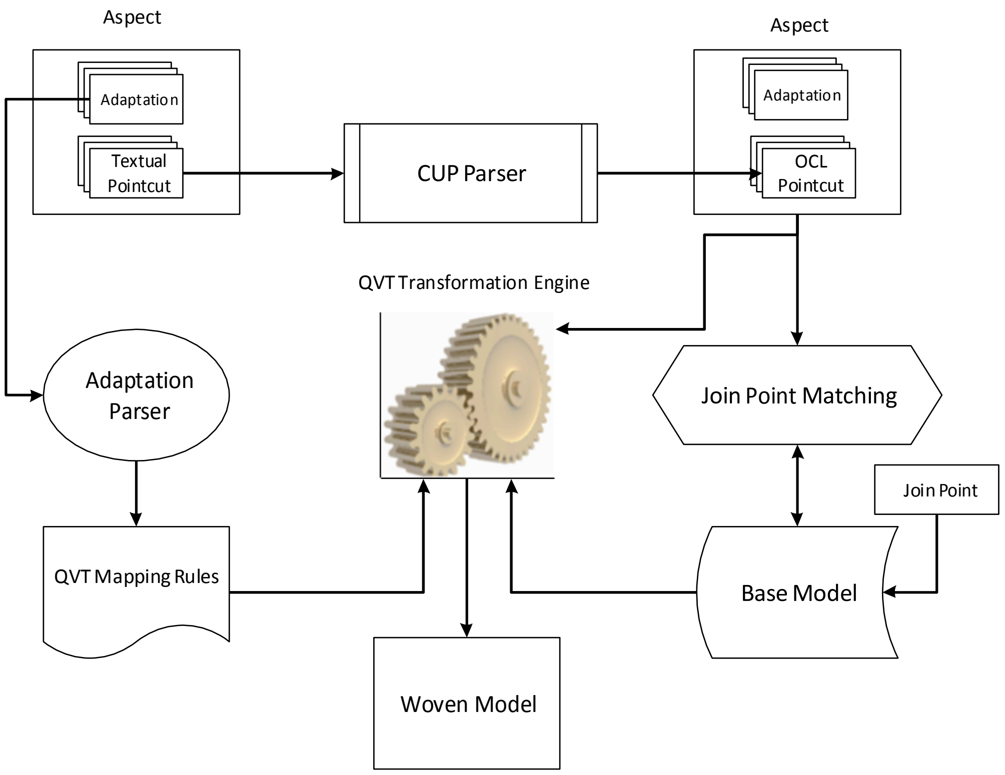


Figure 9 – MOBS2 Framework Overview [37]

3.2. Pointcut Definition

The requirements unambiguously and precisely state the conditions that define the pointcut. This will make use of the phrases meaningful to the model space. However, another important point to note is that the security expert should not be burdened with the necessity of learning a new language for defining the pointcuts.

The legal sentential forms are required to conform to a grammar. The legal sentential forms are then translated to generate an expression that conforms to the OCL specifications. A Java-based parser generator “CUP” is used to generate a parser for the defined grammar. This CUP parser generator takes the input as the grammar defined for the customized *‘pointcut’* specification language and outputs a parser. Then the

`

sentential forms defining the pointcuts are converted by the parser to an OCL expression.

In this context, the MOBS2 tool uses the pointcuts that are defined by using *"text string"*, meaningful to the model, linked with each other using conditionals and logical operatives. The pointcut language used in the framework designates the UML elements in software design. For example, consider the pointcut expression to designate a package 'p1' containing a class 'c1' as shown in Equation 1

$$(\quad) \qquad (\quad)$$

Equation 1

The OCL Expression equivalent to the textual expression is generated using the Java CUP parser. The CUP parser uses a well-defined grammar that helps translating the textual pointcut into the OCL expression that is required by the weaver. The generated expression is as shown in the following Equation 2.

$$(\qquad)$$

$$(\quad)$$

$$(\qquad (\quad) \qquad)$$

Equation 2

3.3. *UML Profiles for AOM*

The software system that is to be developed would be represented by a combination of structural and behavioral models of UML. The structural models mostly used by designers are the *'class diagrams'* and the behavioral models mostly used are the *'state-machines'* and *'activity diagrams'*. This section represents the AOM profiles that extend UML concepts. It contains a set of adaptations and pointcuts. An adaptation specifies

`

the modifications on the base model done by the aspects. In the MOBS2 framework, the AOM profiles are represented as stereotype packages as shown in Figure 10. Profiles have been created for the various UML models in the MOBS2 framework. The security expert will customizes these UML profiles to implement security philosophies specific to the application.

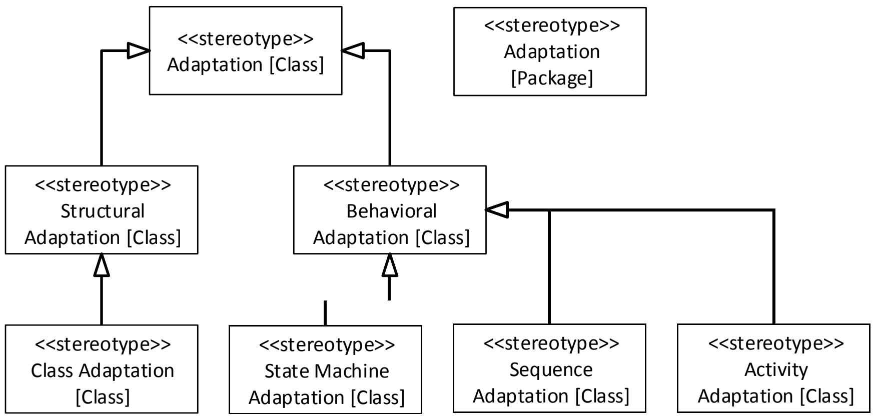


Figure 10 – AOM Profiles in MOBS2 Framework [37]

As illustrated in Figure 10, the security profiles have been created in the MOBS2 environment, which will aid the security architect to concentrate on the *'secure'* features that need to be designed into the application. The security profiles are generalized meta-models and are specified as stereotypes with tagged values and constraints. This enables the security expert to easily adapt the profiles to the particular application domain. The MOBS2 framework supports all the above model representations. Graphical representation and manipulation of all these models are supported in the eclipse development environment.

`

3.4. *Join Points*

The positions in the model instance under consideration, which satisfy the conditions defined as a pointcut is to be identified. A matching algorithm is adopted to define the positions in the base model to insert pointcuts. The algorithm takes in an “OCL expression”, which defines the pointcut as an input and scans through the model to identify the likely candidate points. The whole concept and the matching algorithm are quite general in nature and can be easily implemented to handle the various types of model diagrams. The matching algorithm that is implemented is as given in Algorithm 1.

```
                        (        )
        (                           )
                       (         )

                                  ( )
```

Algorithm 1 – Join Point Matching [37]

Algorithm 1 shows a general implementation of how the process is repeated for each pointcuts specified in the aspect. On a larger system, each of the pointcut elements belongs to a specific adaptation, so specifying the adaptation by passing the specific

`

base model elements. Including this filtering mechanism in Algorithm 1 improves the performance of join point matching in larger systems.

3.5. Transformation Tool & Weaving

Figure 11 illustrates the detailed process of the framework. The adaptations will have to be invoked at specific points in the base model. The adaptation will be associated with pointcut definitions. A two phase process of identifying the join points in the base model (matching) and integrating the adaptations into the base model (weaving) is supported by MOBS2. The point-cut as indicated earlier will be specified using a “text” string. This string would be converted into an OCL expression. This OCL expression will then be used as the match string for the pattern matcher which scans the input model and identifies the join points.

The transformation tool holds a set of transformation definitions targeting a particular UML diagram, and contains a set of mapping rules which defines the transformation of each element in the corresponding diagram. The transformation tool’s architecture facilitates extensions to cater to a wide range of UML diagrams by plugging in without modifying the existing architecture. The transformation tool uses the base model as an input, and defines appropriate mapping rules for specific diagrams using the underlying QVT engine. QVTO (Query, View, transform operational) is an OMG standardized set of rules and functions. This defines the way the base model will be modified and the appropriate adaptation is merged with the base model by transforming the base model, i.e. the adaptation will be weaved into the base model.

`

3.6. Summary

To use the MOBS2 tool, there are few steps to follow in order to use the weaving capabilities. The weaving process is organized into four steps: (a) Aspect specialization, where the application-independent aspect provided from the security aspect library is going to be instantiated and will produce an application-dependent aspect. (b) Pointcut translation where each textual pointcut defined in the aspect is translated into an equivalent OCL expression. The previous two steps can be considered preliminary steps before the actual weaving begins. (c) Join point matching, where the generated OCL expression is evaluated on the base model to identify the locations where weaving is to be performed. (d) QVT transformation rules generation phase, which takes as input the set of identified locations from the previous step along with the set of adaptations specified in the aspect. It then selects the appropriate transformation rules to be executed on the base model. Finally, the woven model is produced as the output.

`

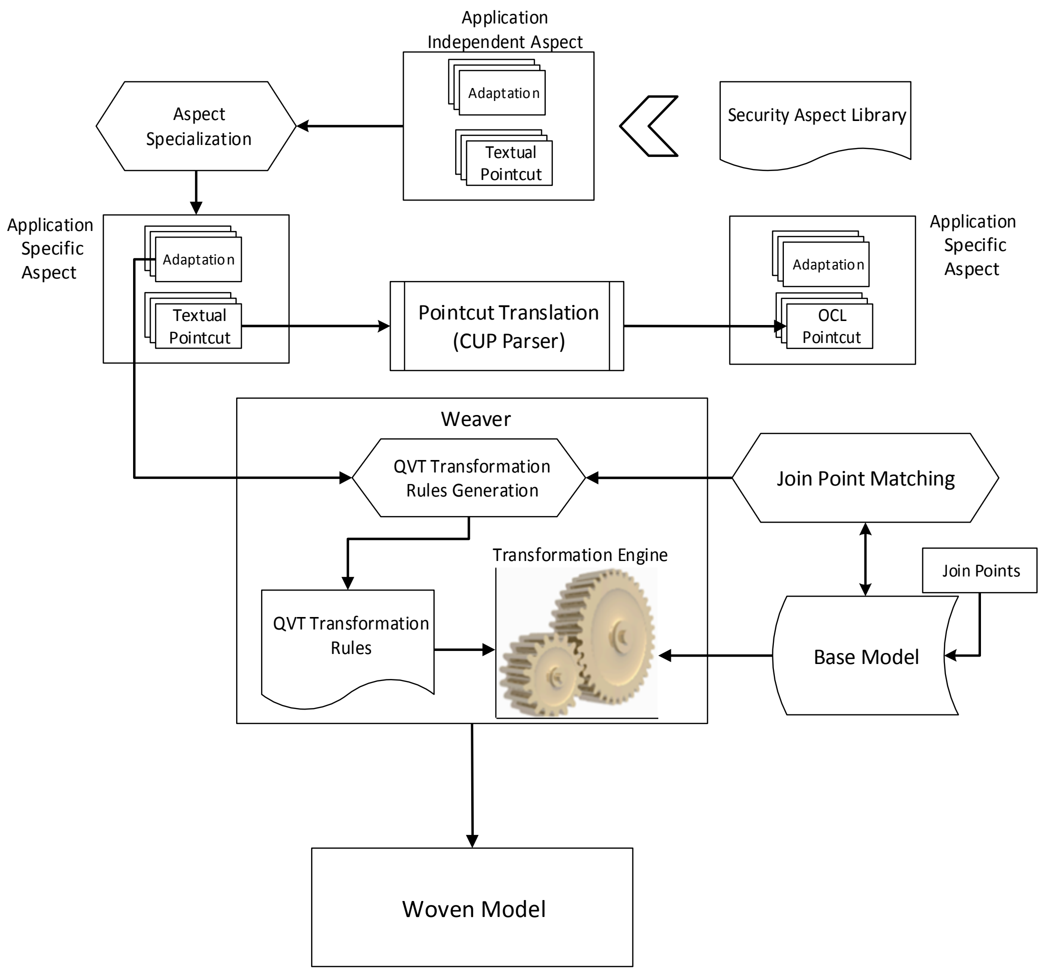


Figure 11 – MOBS2 Complete Framework [37] [43]

`

Chapter 4 Additions of New Pointcuts and Advices

This chapter elaborates on the steps taken to design new *Join Points*. The *Join Points* are designed for both structural and behavioral diagrams of the UML models. The new specifications as aspects help to further harden the base model to meet the requirements of the system. To precisely describe the additions and their implications, two example models common to the entire chapter are adopted. The models that are used are as listed hereafter.

Bank ATM Base Model: The 'Bank ATM' cross section is the first model used as the base model for the whole chapter. The common example used to describe the additions made in the application and the implications of these additions. Figure 12 illustrates the base model used in the chapter.

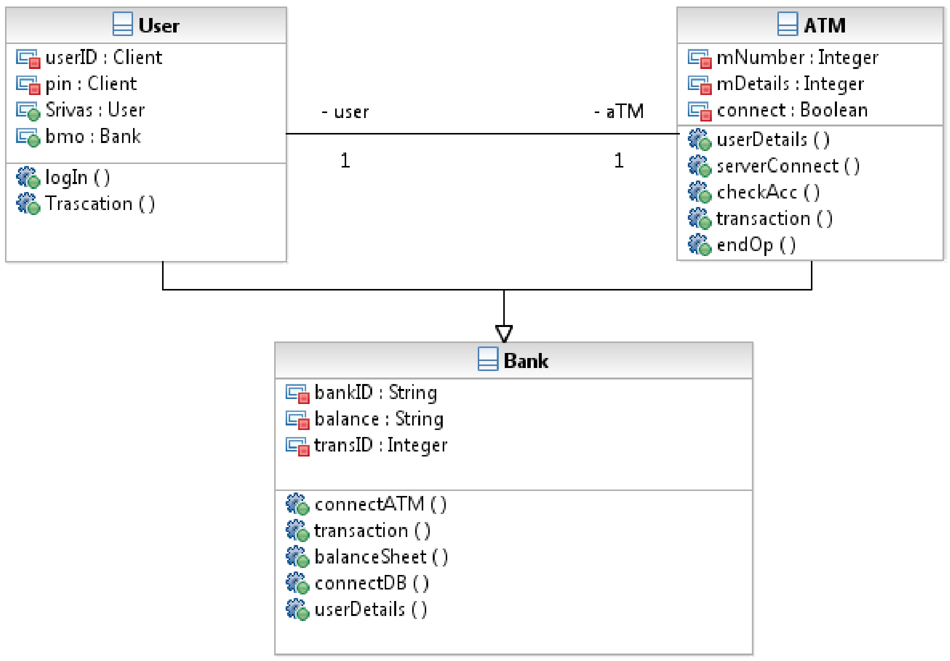


Figure 12 – Bank ATM Cross-section Base Model

`

The base model contains 3 classes :

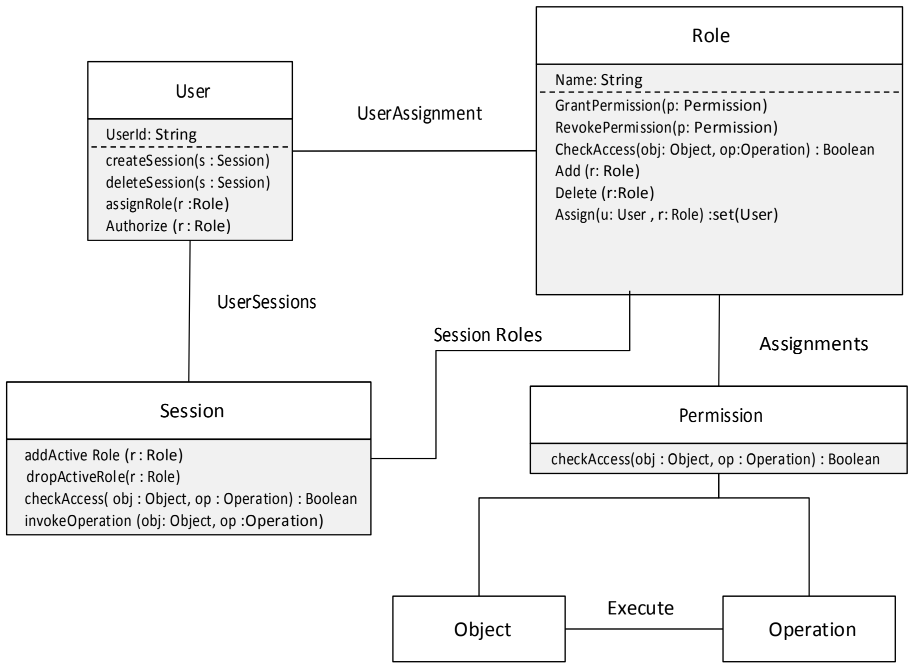
- User Class – The class contains the information pertaining to the user of the system
- ATM – The machine that contains the system at the background.
- Bank – The ATM and the user use the bank interface to perform transactions and then commit the changes to the appropriate user account.

In the class diagrams just enough methods have been shown which when executed in the right sequence will enable the user to complete a transaction on the ATM.

RBAC Security Aspect Model: The Second model adopted for this chapter is the Role-Based Access Control (RBAC) template as the security aspect model. The RBAC common template enforces access control based on user roles and permissions. It involves five key concepts – user, role, session, operation, and object [44]. Role represents a job function with certain authority and responsibility in an organization. A role is represented as a relation between users and permissions, and a user assigned to a role acquires the permissions given to the role. Session represents an instance of a user's dialogue with the system. A user can create or delete a session, and activate or deactivate a role in a session. A session may be defined as a mapping of a user to the set of roles that are activated by the user. An object represents any information resource (e.g., files, databases) to be protected in the system. Operation is an access request to an object invoked in a session. Permission represents an authorization to perform an operation on an object and can be represented as a relation between an operation and

`

an object. Roles may be structured in a hierarchy to reflect an organization's lines of authority and responsibility. Figure 13 illustrates the RBAC template.



Role
Name: String
GrantPermission(p: Permission)
RevokePermission(p: Permission)
CheckAccess(obj: Object, op:Operation) : Boolean
Add (r: Role)
Delete (r:Role)
Assign(u: User , r: Role) :set(User)
User
UserId: String
createSession(s : Session)
deleteSession(s : Session)
assignRole(r :Role)
Authorize (r : Role)
UserAssignment
UserSessions
Session Roles
Assignments
Session
addActive Role (r : Role)
dropActiveRole(r : Role)
checkAccess(obj : Object, op : Operation) : Boolean
invokeOperation (obj: Object, op :Operation)
Permission
checkAccess(obj : Object, op : Operation) : Boolean
Object
Execute
Operation

Figure 13 - Role Based Access Control Template [37]

In our approach we use Flat RBAC model as the security aspect. A Flat RBAC is the core model that embodies the essential concepts of RBAC: users, roles, and permissions. It specifies the assignment of users to roles and the assignment of permissions to roles. In order to enforce RBAC access control mechanisms on the different resources of our application, we need to introduce the primary concepts of RBAC to our application. In our approach, Flat RBAC template that depicts the essential structure and concepts of RBAC has been adopted. The security aspect model uses the AOM profiles defined for the weaver application. The AOM profiles define the model's properties and depict the role that the model will play in the weaving process. Figure 14 shows the use of AOM profile in RBAC designed for this application.

`

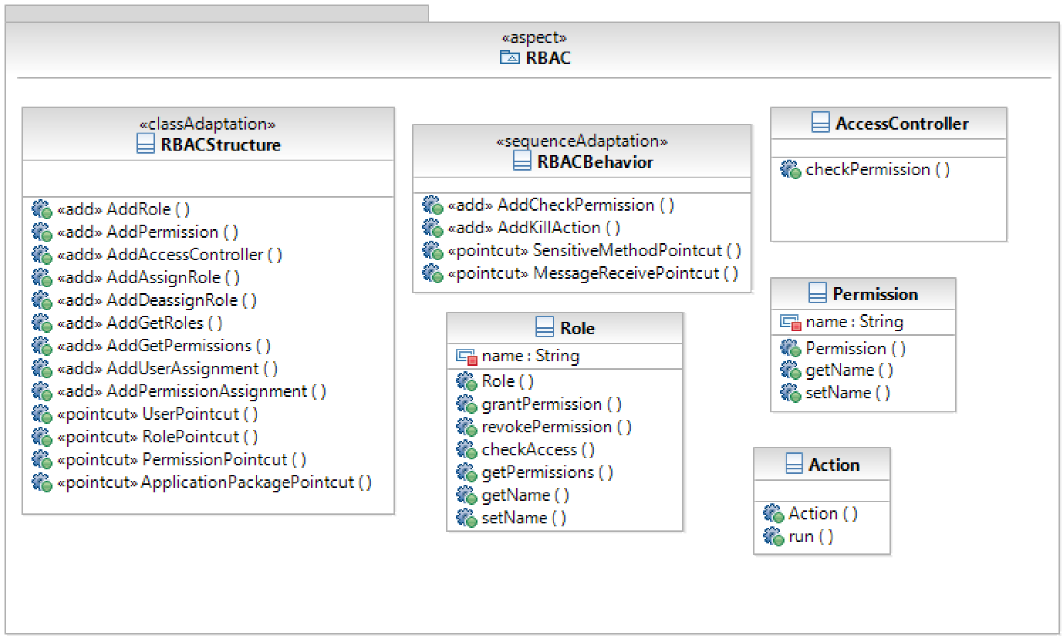


Figure 14 – RBAC, Security Aspect Model

As illustrated in Figure 14, the AOM profile defines the RBAC templates as an *'<<aspect>>'* stereotype. The stereotype forges the RBAC template to function as an aspect model to the weaving process. In a similar fashion, the RBAC template further incorporates stereotypes like *'<<classAdaptation>>'* ,*'<<sequenceAdaptation>>'* and many more to specify the roles the classes play in the weaving process. These stereotypes provide the necessary property specification to the weaver application to perform the specific weaving to the base model. For example the *'<<classAdaptation>>'* performs only weaving on the structural aspects of the base model and weaves the specific security components in the system; Similarly, *'<<sequenceAdaptation>>'* performs weaving into the behavioral aspects of the base models.

There are also stereotypes such as *'<<add>>','<<pointcut>>'* and *'<<remove>>'* which provide information such as: advise to add, at what position, and if that advise needs to

`

be removed from the specified position. These stereotypes provide the appropriate property specification to the weaver application and allow the weaver to perform the directed weaving. The chapter is dived into 2 main sections. First section describes the additions made to the parser in order to generate the required OCL expressions that are used in the weaver application. The second section describes how the generated OCL expression used in the weaver application as inputs and how the additions affect the system.

4.1. Additions to the Parser

This section describes the additions and their implications on the parser. The Java-based CUP parser generates the OCL expression with a valid textual model as the input. The textual model is tokenized to pick the elements for the OCL expression. The expression built by the CUP parser is adopted as inputs by the weaver. To generate the OCL expression from the textual model a set of tokens comprised of the terminals and non-terminals in the model are defined. The definitions for such textual models follow Example 1

$$(\quad)\qquad\qquad(\qquad\qquad)$$

Example 1

As illustrated in Example 1, the textual model contains two tokens namely: 'class' and 'inside_package'. The tokens are circulated through the parser to create the OCL expression. Algorithm 2 depicts the generation of the OCL expression by the CUP parser. The parser searches for the tokens and builds the OCL expressions. The tokens namely,

`

‘class’ and ‘inside_package’ are extracted from the string. These tokenized strings are then matched with the appropriate productions in the grammar in order to create the OCL expression. The resultant expression is utilized as the pointcut expression and in the weaver application.

```
Input: PointcutExpr::= set (ClassPointcutExpr)
      Loop while (length (PointcutExpr))
          if (PointcutExpr = Class) then
          Result = newExpr String “self.oclIsTypeOf (Class)” + “self.name “
          end if
          if (PointcutExpr = InsidePackage) then
              Result = newExpr String “self._package.name =” + “self.name”
          end if
      end Loop
```

Algorithm 2– Class Pointcut Expression

The OCL expression [1] generated by the CUP parser through Algorithm 2

(   )

[1]

The details of the additions are explained in the following sections. The parser presently caters for some Structural and Behavioral components of the model. However, in order to be able to express the new discriminants we need to update the parser to handle these newly defined structural and behavioral components. The components are modified at the parser level to incorporate the new aspects designed for the weaver

`

system. The following section explains the additions made on the structural components.

4.1.1. *Structural Components Enhancement*

This section describes the additions made to the set of structural components that the CUP parser handles. The structural components are comprised of the following: Class Diagrams, Objects diagrams, Deployment diagrams, Package diagrams, Composite structure diagrams and Component diagrams.

The additions to the parser are illustrated using the new class diagram components that we are introducing. These additions affect the class/package section of the system structure. However, as these are only structural additions, the functioning of the system is not affected in any way. The additions are as follows:

a. *Arguments* (ARGS) *Pointcut*

Arguments are items present in an operation as parameters. These arguments are used by the various classes to pass information (values) to the operations. To generate the appropriate OCL pointcut expression, the textual model expression should contain the token ‘arguments’, the addition of ‘ARGS’ along with the operation pointcut definition allows the parser to generate the OCL expression. To accurately describe ‘ARGS’ the following example is used.

( ) ( )

(( ) ( ))

Example 2

`

In Example 2, the textual model expression contains a class *‘ATM’* inside the package *‘BankCrossSection’* that contains an operation *‘LogIn’,* which contains arguments *‘iPin’*. The parser examines the textual model expression and finds the *‘iPin’ argument.* The token *‘cotains_args’* is adopted by the parser to generate the appropriate OCL pointcut expression. The parser implements Algorithm 3 to generate the desired OCL equivalent *‘self.ownedArgument’* of *arguments*.

```
                              (                         )
               (       (              ))
        (                                  )
                                                             (     )
```

Algorithm 3– Argument Pointcut Algorithm

As depicted in Algorithm 3, the parser checks the expression for the token *’contain_args’* in the class and the operation specified in the expression containing the required specific argument. The textual model expression must include a valid argument name specific to that operation. For the expression of Example 2 the generated OCL pointcut expression is as follows:

```
                    (     )
                                      (
                         (                        )
                         (                        ))
```

[2]

`

The OCL expression [2] illustrates the generated expression which will be used as the pointcut expression in the weaver application for the item 'ARGS'. The expression contains a class *'ATM'* inside a package *'BankCrossSection'* which contains an operation *'LogIn'* and a specific argument *'iPin'*. This generated pointcut expression is used as the input the weaver application to identify and add a *Join Point* in to the base model.

a. Attributes (ATTR_TYPE) *Pointcut*

Attributes are logical data values of an object. The parser searches for attribute tokens based on the attribute name in the textual model expression. The 'ATTR_TYPE' token allows the parser to search attributes based on the UML Property Types (data type) of the attribute, currently. The parser supports two UML property types: 'Integer' and 'String'. The data type in the textual model expression has to contain either 'DT_INTEGER'or'DT_STRING' to specify the type of the attribute. Using an example to described prominently the 'ATTR_TYPE' item.

(   )   (   )

((   )   (   ))

Example 3

In Example 3, the attribute '*AccNo*' is of type '*Integer*', the parser will use the textual model expression and generate the OCL expression; Using Algorithm 4 the parser generates the OCL expression for the text sting with 'ATTR_TYPE" specifier.

As depicted in Algorithm 4, the parser checks the expression for 'attr_type' token specified in the class and the specific data type. The textual model expression must include a valid data type.

`

```
                         (                        )
              (       (              ))
       (                             )
                                                    (     )
```

Algorithm 4- Attribute Pointcut

For the Example-3 mentioned earlier, the generated OCL point cut expression [3] is as shown below.

```
                  (      )
                                     (
                      (                     )                                  )
```

[3]

The OCL expression [3] illustrates the generated expression which will be adopted as the pointcut expression in the weaver application for the item ‘ATTR_TYPE’. The expression contains a class *‘ATM’* inside a package *‘BankCrossSection’,* which contains an attribute *‘AccNo’* of type *‘Integer’*. This generated pointcut is used as the input expression in the weaver application to add a *Join Point* into the base model.

These were the additions on the structural part of the CUP parser. To summarize the additions to the structural component set of the parser expands the pointcut expressions that the Parser can generate in order to handle the new discriminants. The discussion on the pointcut expressions will be further illustrated in the weaver section of

`

this chapter. The next section discusses the additions to the set of behavioral components that are currently handled by the parser.

4.1.2. Behavioral Component Enhancement

The following section describes the additions to the set of behavioral components handled by the parser. The behavioral components comprise the following: Activity Diagrams, State Diagram, Use Case Diagram, Sequence Diagram and Timing diagrams. The additions to the parser are illustrated using a sequence diagram. These additions affect the overall working of the application. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the sequence of messages exchanged between the objects needed to carry out the function. The additions are as follows:

a. Lifeline Pointcut

The lifeline is an instance of an object, it demonstrates the roles of different processes or objects that live simultaneously. Modification performed by the parser on the lifeline brings out changes in the behavior of the model. As mentioned earlier, the lifeline shows the instances of the classes that are involved in the interaction as objects. The textual model expression should contain *'LifeLine'* artifact. For example, the developer specifies in textual model expression the definition of a new lifeline representing the role of the User. The appropriate pointcut expression is generated.

( )

Example 4

`

As illustrated in Example 4, the textual model expression contains *'lifeline'* token. The parser will search for the token and then will generate the appropriate OCL equivalent. To do so, the parser uses the following Algorithm 5 :

```
                              (                         )
                (        (               ))
          (                              )
                                                            (        )
```

Algorithm 5 – Lifeline Pointcut

Depicted in Algorithm 5, the parser searches for the *'Lifeline'* token from the textual model expression and generates the OCL expression. For the mentioned, the generated OCL pointcut expression is shown below in expression [4], is input to the weaver application.

(   )

[4]

b. Message Pointcut

A message conveys information from one instance, which is represented by a lifeline, to another instance in an interaction. This allows the specification of runtime scenarios in a graphical manner. The parser searches for *'Message'* token and generates the OCL

`

expression. The parser follows Algorithm 6. To clearly describe the algorithm, an example is used to portray the OCL expression generation.

( )

Example 5

As illustrated in Example 5, the textual model expression contains *'Message_Call'* token. The parser will search for the token and generate the appropriate OCL equivalent expression. The parser also adds the information on the type of the message call. A message call can be one of two types: synchronous or asynchronous. Synchronous calls, which are associated with an operation, have a send and a receive message. A message is sent from the source lifeline to the target lifeline. The source lifeline is blocked from performing other operations until it receives a response from the target lifeline. The parser uses the following Algorithm 6 to generate the OCL expression.

```
                    (                    )
        (      (            ))
    (                        )
                                        (        )
```

Algorithm 6 – Message Pointcut

`

As depicted in Algorithm 6, the parser searches for the *'Message_Call'* token from the textual model expression and generates the equivalent OCL expression. For the above mentioned example the generated OCL pointcut expression is shown below

( ) ()

(

)

[5]

OCL Expression [5] listed above is adopted as an input to the weaver application with a message named '*LogIn*'.

4.1.3. Summary

This section captured the effect of the modifications on the target application. However, at a low level structural and behavior changes only generate half the output. The parser is not responsible for weaving the aspects into the base model; these operations are completed by the weaver application. The parser generates only the necessary pointcut expressions in OCL, which are input into the weaver application. The modifications to the weaver are made at the application level. The next section describes the modification performed to the weaver application to equip the application to handle the enhancements.

`

4.2. Enhancements of the Weaver Application

The weaver application is responsible of performing the weaving of the aspects into the base model. The application includes two main components: *Join Points Matching Module* and *Transformation Rules Engine*. The join points matching module is responsible for querying the base model using the generated OCL expressions, and returning the appropriate set of elements that are validated. The process of weaving aspects into UML models is considered as a transformation process, the base model is being transformed into a new model enhanced with new (security aspect) features. The language used in the weaver application is the OMG standard application *Query View Transformation* (QVT).

Transformation rules engine is executed on well-defined join points. A well-defined join-point constitutes a valid pointcut expression along with an *'add'* or a *'remove'* and a *'pointcut' s*tereotype. The rules are implemented using the *Eclipse Model-to-Model* (M2M) plug-in, in the Rational Software Architect environment. The QVT black box mechanism allows the use of external functions along with the QVT language. As discussed in chapter 3, the OCL expression generated by the parser is passed into the Join Point Matching Module, which validates the base UML model against the OCL expression. In an effort to increase scalability, the additions are performed to the application in both *structural* and *behavioral* sections. These additions help in increasing the security hardening process of the base models.

The modifications on the application are divided into two sections; firstly the *structural modification* that directs the security model to achieve structural transformations (class

`

and package diagrams) of the base model. Secondly the *behavioral modification* that directs the security model to achieve workflow transformations (interaction and activity diagrams) on the base models. As mentioned earlier at the beginning of the chapter, two sample models, namely *"BankCrossSection"* and *"RBAC",* are used to depict the process of performing security hardening using the new pointcuts and advices woven into the base model.

4.2.1. Structural Enhancements

This section describes the additions and their effects on the structure of the base model. The structural components in the weaver application comprises of the following: class diagrams, objects diagrams, deployment diagrams, package diagrams, composite structure diagram and component diagram. The additions to the application are illustrated using class diagram components.

These additional components increase the capability of the developer to perform stronger security hardening on base models. However, these additions will not modify the workflow of the model. The following are some of the newly added join points for structural modification, designed in an effort to increase the degree of security hardening on the base model.

a. Arguments Advice Weaving

The operation join point designed in QVT performs an addition of an operation in the base model based on the specifications mentioned by the pointcut expression. The Join

`

Point matching algorithm for selecting the operation specified in the pointcut expression is presented in Algorithm 7

```
                    (                         )
                (              (          (          )))
                               (            )
                              (             )
```

Algorithm 7 – Operation Join Point Matching

Algorithm 7 illustrates the process of weaving a new operation into the base model. The application begins my matching the join points with the base model according to the specified pointcut expression. If the specified join point is present, the application appends the operation into the base model as *'newElem'*. The algorithm is designed to add new operations to the base model based on valid *'class'* and *'package'* elements of the base models. The algorithm is modified to accommodate the addition of arguments in the operation. The join point matching algorithm is designed to work around the QVT language restrictions. Essentially, *'Operations'* and 'Arguments' are two mutually exclusive entities in UML models. The solution to work around constraint of QVT is to split the pointcut expression into two separate expressions. The first expression contains the *class* and *package* specifications and the second expression contains the specific operation and the arguments. The algorithm is modified in order to overcome the QVT restrictions the solution to the restrictions of QVT.

`

```
                    (                                  )


    (           (        (              (      ))))


     (           (        (              (      )))
                                    (          )
                                     (                 )
```

Algorithm 8 – Argument Join Point Algorithm

Algorithm 8 is the modified version of Algorithm 7, which allows the two mutually exclusive UML entities *'arguments'* and *'operations'* to be used together to perform the join point matching with a given pointcut expression. The join point matching algorithm will validate the pointcut expression and weave a new operation into the base model. The algorithm can be appropriately described with an example. Using Algorithm 8, generated pointcut expression for the arguments in an operation. The pointcut expression is divided into two separate expressions. The first part of the expression contains the class and package specification where the join point matching algorithm has to select the position to insert the aspect. The first part of the expression is shown below as Part 1

$$(\quad)$$

Part 1

`

The first part of the expression as shown contains the class 'ATM' and the package 'BankCrossSection' specifications. Algorithm 8 uses this first part of the expression to select the class and package from the base model. The second part of the expression Part 2 is used by Algorithm 8 to select the operation with the appropriate argument.

```
                         (                )
            (                                )
```

Part 2

The separated pointcut expressions are validated. The validated expression creates a new operation that is to be woven into the base model. The modification on the operation allows the security developer to weave multiple operations with different arguments. The addition of arguments with operations supports polymorphism techniques in AOM and allows the developers to create complex designs.

b. Attribute Advice Weaving

The parser generates a pointcut expression with attributes along with the UML primitive type (data type). The attribute can be of type either *'Integer'* or *'String'*. The weaver application allows the developer to weave attributes with a specific data type to the base models. The use of this addition to the weaver application can be precisely described with an example.

```
                          (        )
                                                 (
                              (                           )                                            )
```

Example 6

`

Example 6, a generated pointcut expression adopted by the weaver application, contains the class, package and attributes specification with an ‘Integer’ data type specification.

```
                         (                                   )
              (                (           (                  (       ))))
                                      (             )
                                       (                       )
```

Algorithm 9 – Attribute Join Point Matching

Algorithm 9, describes the attribute join point matching performed on the pointcut expression. The Joint point matching module validates the expression and weaves the newly created attribute into the base model at the specified position. The attribute can further be specified as a static attribute by the weaver. A static attribute is a statically allocated variable whose lifetime extending across the entire application remains constant. The static attribute contains the name and the data type of the variables. Weaving the static property into the base model allows the allocation of constant values that can be used throughout the application. The addition of attributes with data type and property allows the developers to incorporate static attributes into the base models that extend lifetime throughout the system. In highly complex and large models, usage of abstract classes is common. The addition of static attributes allows the use of inheritance properties in UML models, with different classes and packages inheriting these attribute values from abstract classes or packages.

`

4.2.2. *Behavior Enhancements*

This section describes the components added in the behavior section of the weaver application and their effects on weaving process of the behavioral components of the base model. The additional components alter the working of the base models and add the desired control flow into the existing one. For the following section, we use a common sequence diagram as an example to describe the effects of the behavioral weaving. Figure 15 depicts a sequence diagram that illustrates the communication involved between ‘ATM’ and ‘Bank’ classes. A sequence diagram shows the interactions between the instances of objects depicted as lifeline and the interaction between these objects known as messages. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

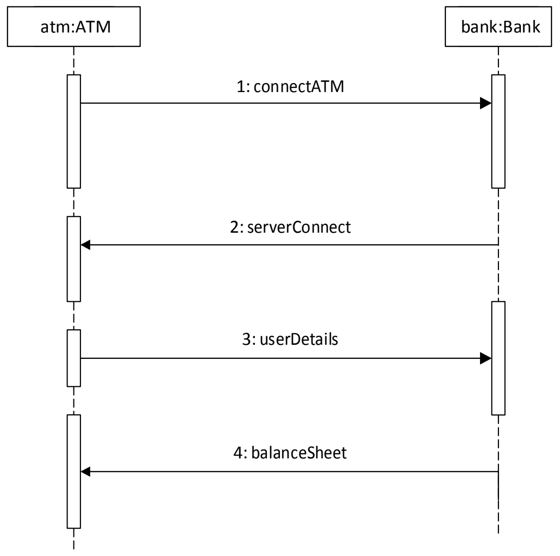


Figure 15 – Balance Sheet Sequence Diagram Example

`

The following are some of the additional behavioral components that are designed and added to the weaver application in an effort to modify the control flow of the existing base model. A sequence diagram depicts the control flow of the application.

a. Lifeline Advice Weaving

A lifeline of an object demonstrates a role it plays in the control of the model. The object generates the actions and provides an in-depth analysis of the objects and their functions depicted in the model. The adaption is a part of a sequence diagram; we are adding a lifeline as an object that changes the workflow of the base model. The weaver implements Algorithm 10 to weave the matching join point to the base model.

Example 7 is used to describe the algorithm.

(   )

Example 7

The pointcut expression directs the weaver application to add 'User', a new lifeline object, into the base model.

```
                    (                              )
        (              (          (                (      ))))
                                  (           )
                                     (                   )
```

Algorithm 10 – Lifeline Join Point Matching

The expression is matched with the sequence diagram of the system by the join point matching module against the existing base model. The new object 'User' lifeline

`

element is woven to the sequence diagram and the new object entry is added into the base model. The enhancements to the models add a layer of security hardening by defining a new role for the object. The addition of lifeline allows the developer to weave a security aspect as the new lifeline object that can change the behavior of the system. The addition of lifeline adds a structural advantage to the base model.

b. Message Advice Weaving

A message depicts the method calls between lifeline objects in a model. Messages are used for communication between lifeline instances in an interaction. The weaver application inserts new message into the base model. To explain the working of the weaving process involved in adding new message calls into the base models Example 8 is used.

( ) ()

(

)

Example 8

The message call *'LogIn ()'* in the pointcut has to be inserted between interactions of the 'ATM' class and 'Bank' class depicted in Figure 15 of the 'BankCrossSection' system. The weaver application implements Algorithm 11, to perform the weaving of message calls into the base model. As Illustrated in Algorithm 11, the join point matching module uses the pointcut expression fed into the application and selects the explicitly specified source and target lifelines within which the message will be woven.

`

```
                         (                              )
                              (                 (      ))
      (                  (          )
       (          (              (    )
                                
                                (                                    )
                                
                            (                                       )
                                     (                              )
                                
          (                  (           )
             (          (              (    )
                                            (                          )
                                           (                     )
```

Algorithm 11- Message Join Point Matching

The weaver application first determines the type of message call to be inserted, either 'Synchronous' or 'Asynchronous' message call. If the pointcut expression specifies a synchronous message, the weaver creates a `send' and a `receive' message call pair and inserts both of them into the base model. If the pointcut expression specifies an asynchronous message call, the application inserts a single message into the interaction

`

and appends the new message into the base model. The message pointcut expression is fragmented into two expressions that separately define the source and the target, along with the name of the method that specifies the object usage. The message advice adaptation adds method to specialize the roles the objects perform in the model. The message advice addition can be explained with pointcut expression Example 8. The expression contains an asynchronous call *'LogIn()'* , the message call is woven between 'ATM' and 'Bank' classes as depicted in the interaction diagram of Figure 15. The weaver application weaves the message call specified into the base model by adopting Algorithm 11. The addition of message call advice to the weaver application allows the developer to insert multiple secure message aspect components into the base model. This addition alters the workflow of the system, so that the developer has the ability to increase system reliability by weaving security aspects and harden the base model.

4.3. Summary

We conclude this chapter describing the additions made to the parser and weaver application to perform weaving on the base model. These additional components provide the developers the ability to specify discriminants by allowing them to precisely change the structure and the behavior of the base model thus presenting him additional capability for designing and implementing algorithms which will offer higher degree of security to the application. Developers are allowed to use techniques like polymorphism, inheritance, and abstraction. The developers can incorporate these techniques that are present in Object Oriented Paradigms (OOP) and Aspect Oriented Paradigms (AOP) to modeling languages and apply secure aspects to the base models.

`

The system hardened by the enhanced set of security aspects at the design level itself increases the scalability and reliability of target systems. The implications of these additions to the application will be described in the next chapter that explains these enhancements with the Session Initiation Protocol (SIP) communicator used as a case study. SIP [45] is a signaling communications protocol widely used for controlling multimedia communication sessions. The protocol defines messages that are sent between peers, which govern the initiation, establishment, termination, and other essential elements of a call (communication session). SIP can be used for creating, modifying, and terminating two-party (unicast) or multiparty (multicast) sessions consisting of one or several media streams. Other SIP applications include video conferencing, streaming multimedia distribution, instant messaging, presence information, file transfer, and online games. The following chapter discusses the implications of the enhanced component set to the weaver application.

`

Chapter 5 Case Study: Session Initiation Protocol (SIP) Communicator

This chapter illustrates the suitability and the effects of components added to the weaver application to perform security hardening on the base models. We show how security aspects can be integrated into a SIP communicator base model. Figure 16, illustrates the SIP communicator factory package that allows a multimedia application to use the SIP communicator protocol for multimedia communication sessions such as text, voice and video over IP networks.

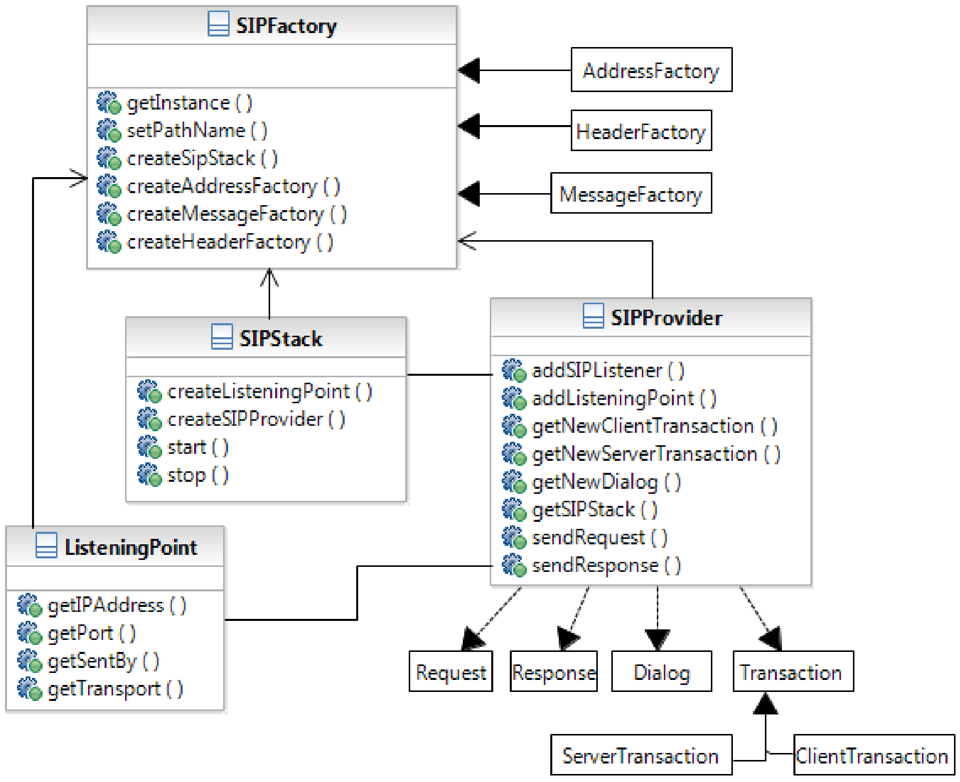


Figure 16 – SIP Communicator Factory Package [46]

The Session Initiation Protocol (SIP) is a signaling protocol used for establishing and maintaining communication sessions involving two or more participants. SIP was initially

`

designed for voice over IP and multimedia conferencing, and then was extended to provide support for other services such as instant messaging and presence management [47]. Today, SIP is also adopted for usage in 3G wireless networks, thus it becomes an integral protocol for ubiquitous environment. The SIP protocol defines the messages that are sent between peers. As stated earlier SIP can be used for creating, modifying, and terminating two-party or multiparty sessions consisting of one or more media streams. In this scenario, the SIP communicator defines a Factory package that allows an open source multimedia application to use the SIP protocol to create, modify, and terminate peers for the required sessions [46]. The depicted base model contains the following design patterns:

- Peer provider pattern – This pattern defines a platform-specific implementation of the SIP protocol and provides the corresponding peer information specifics for the multimedia application.
- Factory pattern – A factory is an intermediary pattern that encapsulates the method for accessing the SIP peer, and allows the application to obtain instances of the peer implementation classes.
- Event Listener pattern – When the SIP stack receives SIP messages (requests and responses) from the network, the SipProvider passes them as events on to the event listener, which is called SipListener.

In this scenario, the multimedia application starts the instance by preparing the sessions (text, voice, and video) between the peers using the peer provider pattern. Then, the application employs the factory pattern to provide the implementations of the peers.

`

Lastly, the application employs the event listener pattern to provide the SIP message to the designated peer. The application will use the package created by the SIP implementation package to initiate a request and send the request to the designated peer to avail a responding message from the peer. Figure 17 illustrates a cross section of the new message created by the multimedia application to initiate a request using the SIP implementation.

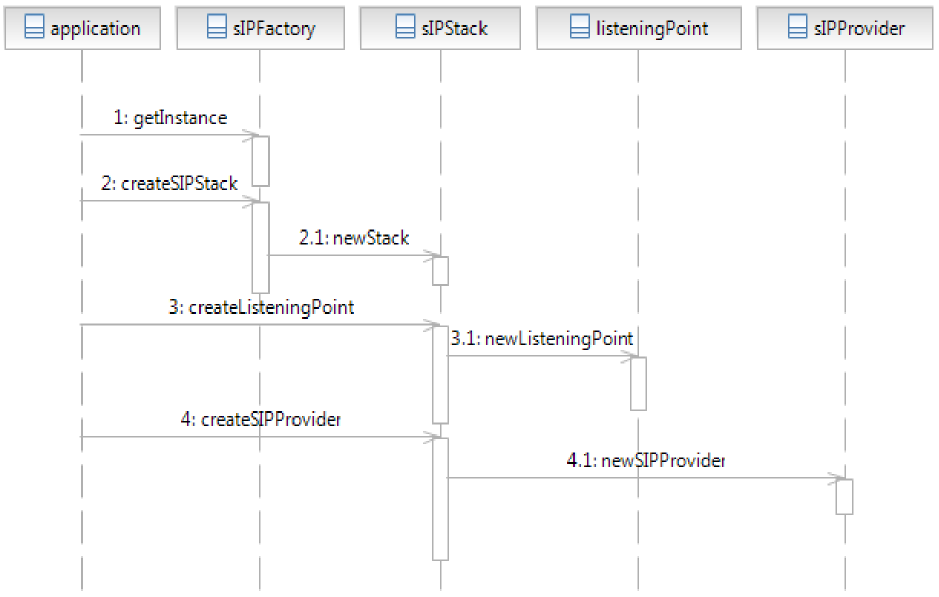


Figure 17 – Create New Message Sequence Diagram [46]

As illustrated in Figure 17, the multimedia application employs the *'SIPFactory'* existing in the SIP implementation package to create a new message. The SIP implementation package creates a new message employing the interfaces: *MessageFactory*, *AddressFactory,* and *HeaderFactory (not shown in the figure)*. The *'sIPStack'* provides the properties of the session and the network. The *'listeningPoint'* will use the properties provided by the 'sIPStack' to generate a new listening port for generate a new listening port for the messages over the IP network the message over the IP network. The SIP

`

communicator is susceptible to threats over the IP network. . Table 3 illustrates the possible threats the SIP communicator is exposed to.

Issues	**Solutions**
Eavesdropping – Unauthorized interception of voice packets, real-time media streaming and decoding hijacked messages	Encrypting the transmitted data over a Secure Socket Layer with various encryption mechanisms
Packet Spoofing – Impersonation of legitimate users while transmitting data	Send address authentication between peers
Replay – Retransmission of a genuine message so that the device receiving the message reprocesses it	Encrypt and sequence messages.
Message Integrity – Ensuring that the message received is the same as the message that was sent	Authenticate messages

Table 3 – SIP Communicator Security Issues [45]

In the following sections, we present our enhancements to the weaver application for weaving two security aspects into the SIP communicator base model: *Secure Socket Layer (SSL)*, and *Role Based Access Controller (RBAC)*.

5.1. Secure Socket Layer (SSL) Aspect

In this section, we show how the additional components designed for the weaver application allow weaving of Secure Socket Layer (SSL) aspect, and the Session Initiation Protocol (SIP) communicator base model. SSL is one of the most important components for online transactions, creating a trusted environment between the peers for communicating. The multimedia application employing a SIP implementation package must use the SSL mechanism to secure the communication environment for the peers. The SSL aspect employs encryption, decryption and certification mechanism to mitigate

`

the threats shown in Table 3 in the multimedia application employing SIP implementing package.

Before illustrating the effects of the weaving into the base model, we provide a brief description of the SSL aspect model. The Secure Socket Layer (SSL) is designed to make use of the TCP/IP network to provide a reliable end-to-end secure service. Figure 18 illustrates the SSL aspect used in our approach. The SSL aspect includes:

- *SSLEngine:* The *SSLEngine* includes encryption, decryption, and the handshake protocol mechanism. The *SSLEngine* is responsible of encrypting the transmitted data to mitigate eavesdropping and *replay attacks.*
- *CertificationManger:* SSL uses the handshake mechanism to validate the certificates between the server and the client. These certificates allow all browsers to interact with secured web servers using the SSL protocol. However, the client browser and the server need certificates to be able to establish a secure connection. Allowing the use of certificates mitigates *packet spoofing* threat in the communication.
- SSLSocketFactory: The socket factory is responsible for creating and terminating requests and responses between clients and servers. The SSLSocketFactory has two distinct entities: server and client. The client entity initiates the transaction, whereas the server entity responds to the client and negotiates to determine the *SSLEngine that is to be used* for encryption of the information in the session

Figure 18 Illustrates the specification of the SSL aspect using the AOM profile presented in Chapter 3. The SSL aspect contains two kinds of adaptations: *Class Adaptation* and

`

Sequence Adaptation. The class adaptation adds a class named *SSLEngine* to the SIP communicator base model. In addition, it enforces the SSL concepts, of certification management, encryption and key management. Furthermore, the class adaptation adds two new operations, *addAuthentication* and addClientCertificate, to assign different encryption mechanisms and append various certificates used in the SIP-based communication sequence. The *sequence adaptation* in the SSL aspect adds *protocolcheck* behavior ahead of any attempt to call sensitive methods. The *protocolcheck* behavior is responsible for checking and securing the communication session between the peers and determining whether the user is trying to send messages, or make voice/video call to other peers over the multimedia application using the SIP communicator, and pass this information to the higher layers for managing the communication appropriately.

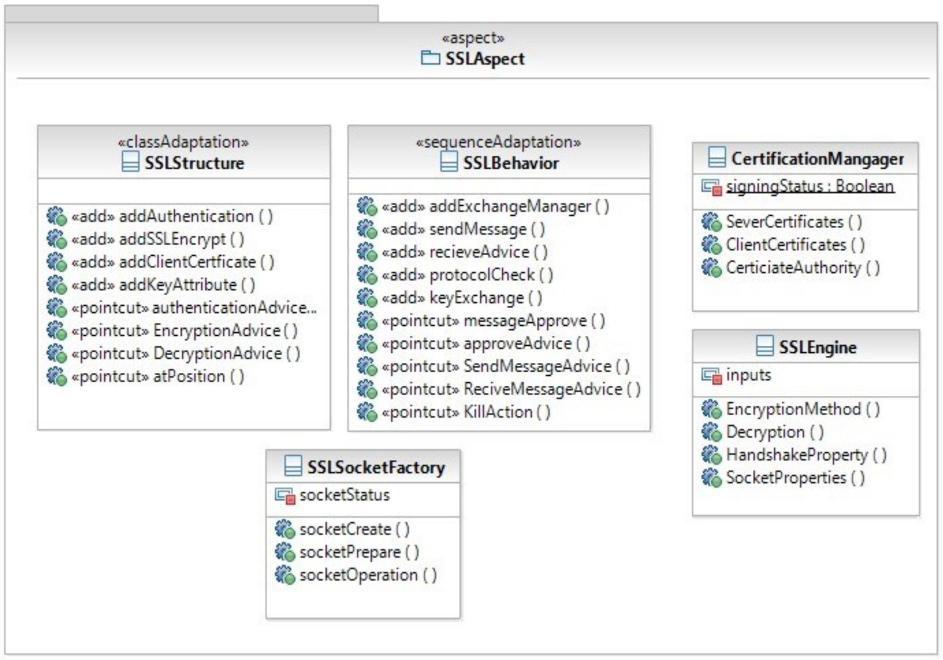


Figure 18 – Secure Socket Layer Aspect

`

5.1.1. Aspect Customization

After importing the SSL security aspects library specified above into the weaver application, the developer needs to customize the generic SSL aspects to suit the target application. This is done by mapping the abstract elements in the aspect into the actual elements in the developer's model; to achieve the mapping, the developer can use the *Weaver Interface* present in the MOBS2 framework. In this case, the developer will explicitly define the advices that are depicted in the weaving interface for weaving purposes. The additional components added to the weaver application allow developers to use polymorphism, abstraction, inheritance principles and weave them into the base models. The *class adaptation* in the SSL aspect weaves *addSSLEncrypt* operation, which is an operation with a specified parameter and weaves it into the SIP communicator model to perform secure message transmission between the peers.

5.1.2. Weaving SSL Aspect

Having customized the aspects to actual elements from the application, the developer selects the instantiated aspect and the application model in order to perform the weaving. During the weaving, each pointcut element is automatically translated into its equivalent OCL expression using the pointcut parser component. This expression is then evaluated on the elements of the base model, and the matched elements are selected as join points. After identifying all the existing join points, the next step is to inject the various adaptations into the exact locations in the base model. This is done by executing the QVT mapping rules that correspond to the adaptation rules specified by the security

`

expert. These mapping rules are then interpreted by the QVT transformation engine that transforms the base model into a woven model. Figure 19 and Figure 20 show the final result after weaving the SSL aspect into the service provider application base models. SSL aspect weaves *class adaptation* (*SSLStructre*) into the base model and alters the structure of the base model as shown below in Figure 19. The modifications on the base model are as follows:

- *addAuthentication*: Weaves a class *SSLEngine* into the base model to include the encryption, decryption, and handshake mechanisms into the SIP communicator base model.
- *addSSLEncrypt*: Defines an operation with a specified parameter and weaves it into the SIP communicator model to perform secure message transmission between the peers.
- *addKeyAttribute*: Weaves the key pairs as an static attribute along with the specified data type into the SIP communicator.

`

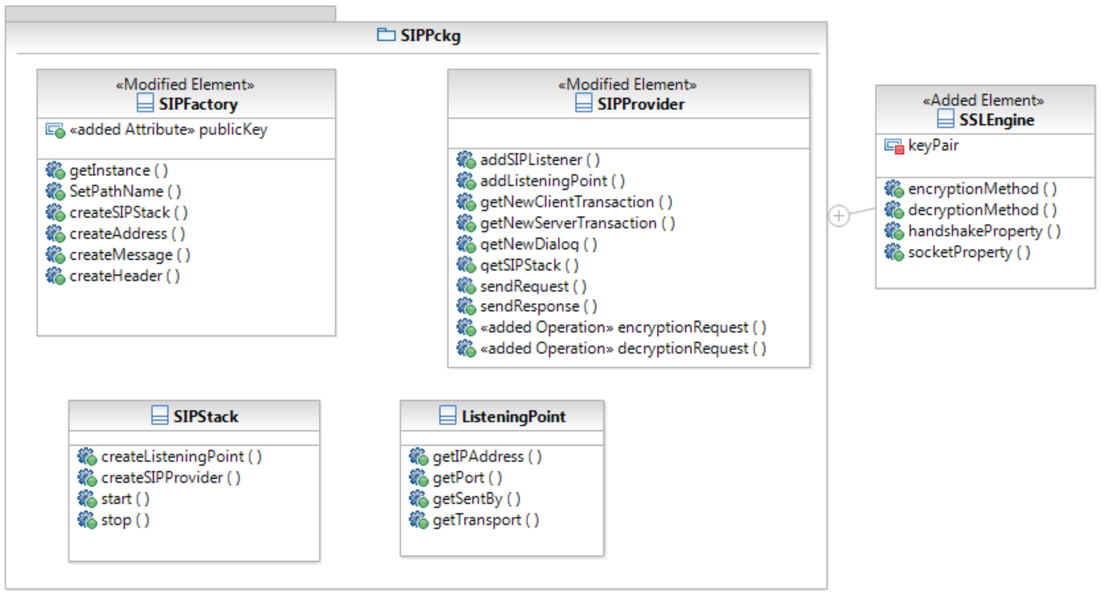


Figure 19 – Woven SIP Communicator Model with SSL Aspect

SSL aspect *sequence adaptation* (*SSLBehavior*) weaves many properties into the base model and alters the control flow of the base model as shown below in Figure 20:

- *addExchangeManager*: This aspect weaves a lifeline object *ExchangeManger*. The *ExchangeManager* is responsible for the exchange of the keys and certificates between the peers, and monitoring the secure environment over the communication in the SIP based communication.
- *protocolCheck*: This aspect weaves the *protocolCheck* behavior message into the base model. This message is responsible for securing the environment before the peers are allowed to transmit and receive data over the IP network. This message call is made before the rest of the sequence takes place.

`

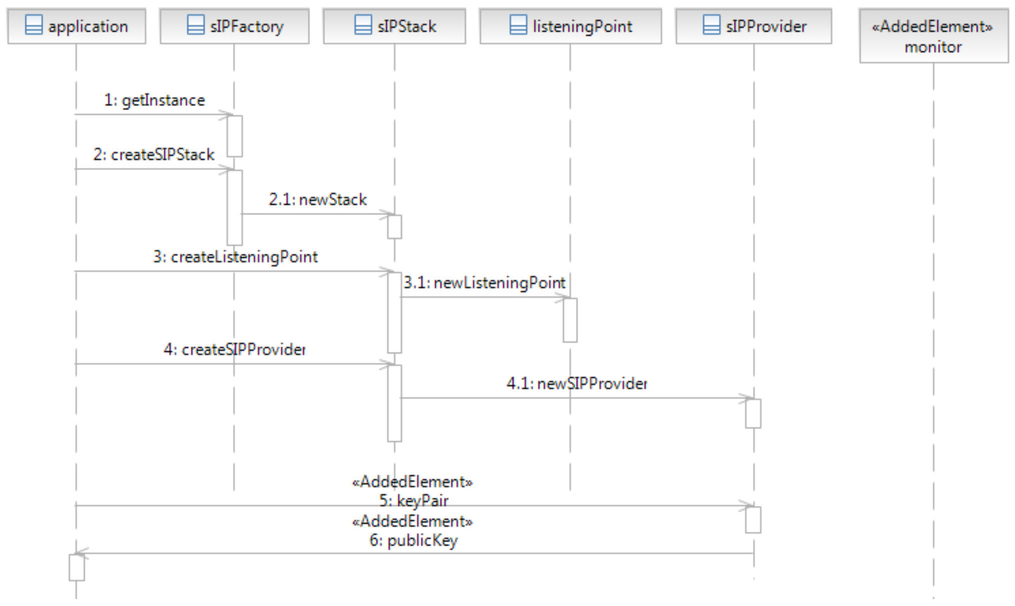


Figure 20 – Woven Behavior SIP Communicator

5.2. Role-Based Access Control (RBAC) Aspect

In this section, we illustrate an access control over SIP based communications. The usage of access control (UCON) [47] over SIP based communication will enable the prescribers to control the identification of their locations and approve or disapprove their subsequent connections, and to also set some parameters to determine whether a certain communication can continue or should terminate [47]. UCON can solve this issue by specifying policies that monitor the SIP communications before and during the call, message transmission, and video streaming. Moreover, it mandates and enforces continuous compliance to access conditions. In the case of noncompliance to these conditions, UCON provides mechanisms for revoking the access. Monitoring and enforcement mechanisms are now collocated and distributed within the SIP communicator.

`

Access control is the means by which the ability to access a specific computer resource is explicitly enabled or restricted in some way (usually through physical and system-based controls). With role-based access control, access decisions are based on the roles that individual users perform as part of an organization. In this section, a method of enforcing access control on the SIP communicator base model will be described. The access control that is used is the RBAC model. In this thesis, RBAC [48] model is used. In order to enforce RBAC access control mechanisms on the different resources of our application, we need to introduce the primary concepts of RBAC to our application.

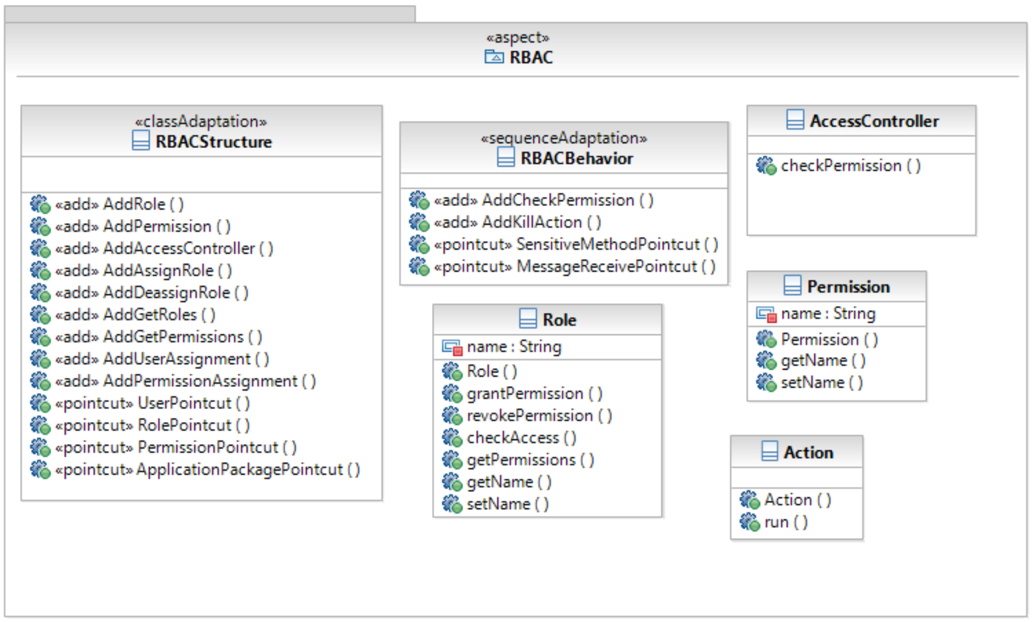


Figure 21 – RBAC security model

Figure 21 depicts the RBAC aspect model with AOM profiles that describe the adaptations that are used in the weaving process. The RBAC aspect model contains two kinds of adaptations; *Class Adaptation* and *Sequence Adaptation*. The *class adaptation* adds two classes, *Role* and *Permission* to SIP communicator base model. In addition, it

`

enforces the RBAC concepts, user-role assignment and role-permission assignment, by adding two associations between the classes (user, role) and (role, permission), respectively. Furthermore, the class adaptation adds two new operations, *assignRole* and *getPermissions*, to assign different roles to users and to get their permissions. The *sequence adaptation* in the RBAC aspect adds the behavior message that checks access ahead of any call to a sensitive method. The check access behavior is responsible of checking whether or not the user who is trying to access a given resource has the appropriate privileges. The RBAC also adds the *Action* class to create specific actions to the application, which carry out the actions depending on the *Roles* specified by the SIP Communicator base model.

5.2.1. RBAC Aspect Customization

The RBAC security aspects library is imported into the weaver application, the library can be customized by the developer to suit the target application. Mapping the abstract elements in the aspect to actual elements in the developer's model; to achieve the mapping the developer can use the *Weaver Interface* present in the MOBS2 framework. In this case, the developer will explicitly define the advices that are depicted in the weaving interface, for the purpose of weaving. The additional components added to the weaver application allow developers to use polymorphism, abstraction, inheritance principles and weave them into the base models. For Example, *RBACStructure* class weaves *addCheckPermission* operation, which is an operation with a specified parameter and weaves it into the SIP communicator model to create specific access privileges to perform message transmission between the peers.

`

5.2.2. *Weaving RBAC Aspect*

The developers customize the aspects to actual elements in the model and the instantiated aspect and the application model are selected to perform the weaving. The weaving application automatically translates each pointcut expression to an equivalent OCL expression using the pointcut parser component. This expression is then evaluated on the elements of the base model, and the matched elements are selected as join points. The QVT transformation engine will interpret the QVT mapping rules specified by the security experts in order to incorporate the adaptations at the selected join points. The transformation engine transforms the base model by adding the selected join points to generate the woven model.

Figure 22 and Figure 23; show the final result after weaving the RBAC aspect into the service provider application base models. RBAC aspect *class adaptation* (*RBACStructure*) weaves a variety of components into the base model and alters the structure of the base model as shown in Figure 22. The woven components on the base model are as follows:

- *addRole*: Weaves a class '*Role'* into the base model to include methods for performing assignment of roles , granting access, and revoking access into the SIP communicator base model.
- *addPermissions*: Weaves a class '*Permissions'* into the base model to include the various permission granting and control mechanisms into the SIP communicator model to enforce specific access control between the peers.

- *addAccessController*: Weaves a class *AccessController* into the SIP communicator to enforce the access control mechanisms in the structure of the application using the SIP-based communication.
- *addAssignRoles, addDeassignRole, addGetRole,addGetPermission,*: Weaves operations into the SIP communicator base model to invoke specific method calls to create appropriate access controls to the specified roles along with the specified permissions between the peers and allow associated messages to the assigned peers.
- *addUserAgreement*: Weaves an association into the SIP communicator base model to enforce user-role assignment and role-permission assignments to structure and allow the role assignments to create a Usage access control (UCON) [47] in the base model.

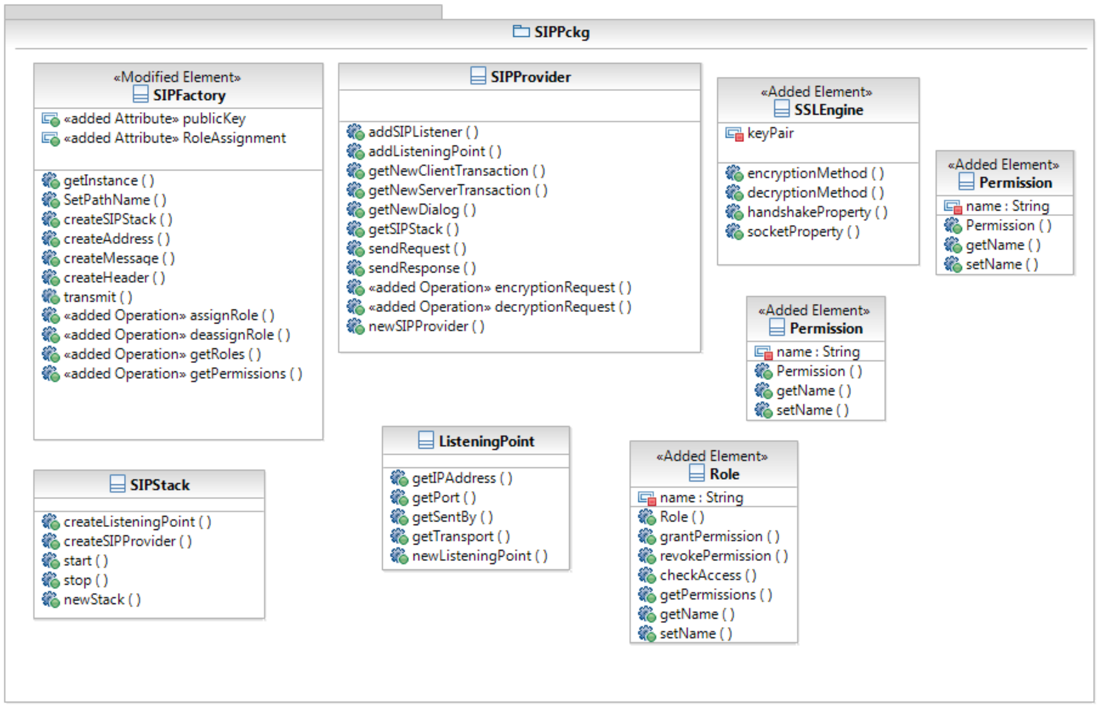


Figure 22 – Woven SIP Communicator Model with RBAC Aspect

`

RBAC aspect *sequence adaptation* (*RBACBehavior*) weaves properties into the base model and alters the control flow of the base model as shown below in Figure 23:

- *addCheckPermission*: The aspect weaves a lifeline object *'Access Control List* (*ACL*)'. The *ACL* is responsible for enforcing the access control in the SIP communicator by checking the permission to transmit data between the peers. The *ACL* also is responsible for granting or revoking access to the multimedia application employing the SIP communicator model.
- *addKillAction*: Weaves the *'destroy'* behavior message into the base model. This behavior message enforces the kill action concept of the RBAC to reject the access control before the peers are allowed to transmit and receive data over the IP network. This message call is made to ensure the role base access control behavior in the multimedia application.

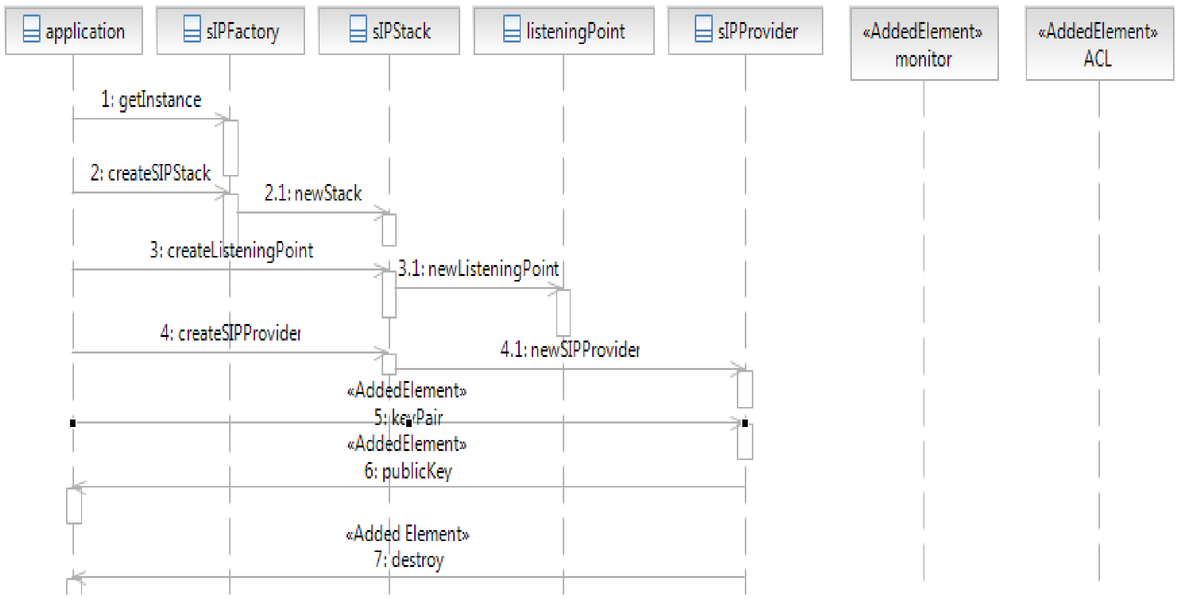


Figure 23 – Woven SIP Communicator Behavior

`

5.3. Summary

In this chapter, we presented the case study of SIP communicator to illustrate the feasibility and the effects of the new components added to the weaver application. In this case study, we demonstrated as to how our approach can be used to integrate the secure socket layer (SSL) protocol and access control mechanisms using role-based access control (RBAC) aspect into the application. These will help to secure the application from various attacks that tend to instrument user input in order to gain control over it.

`

Conclusion

The synergetic and rapid growth of computers and communications has led to connecting people and systems of the world with high-bandwidth digital data networks. The availability of inexpensive and highly performing computing elements has led to development of a plethora of products meant to improve the quality of life, which gather, communicate, and share a lot of information to achieve the desired ends. As the exploitation of these increases, not only will they be used in applications that control critical assets but will also gather and transfer valuable information. Application software, which control these gadgets and networks are being developed at a very fast pace. Often, the race to market the product leads to offerings which have serious shortcomings especially in the non-core functions, which are little appreciated by the general user.

Security is a major concern in a lot of these applications, and measures are often brought in the final stages of the development, which leads to scattering of the security related functions all over the application code. This afterthought practice makes it very difficult not only to validate and verify their functionality and efficacy but also to maintain the code. In this thesis, we have highlighted this problem. We have proposed a methodology and have developed the tools to facilitate systematic introduction of security measures into the application.

The sheer volume of application programs required to be developed, has driven developers to embrace model-centric development instead of the traditional code-centric methodology in order to improve productivity. UML with its rich set of models

`

has been widely accepted. Aspect oriented methodology provides well-proven and neat methods of separation of functional and non-functional concerns in the development process. In the introductory chapter of this thesis, we have elaborated on the features of aspect oriented modeling, listed the UML models widely used by software developers and also explored their application domains. We have shown how adopting AOM techniques and applying them to the various UML models naturally defines a methodology, which lends to easily addressing crosscutting concerns like security. The major advantage is that this can be done without the need of having an in-depth knowledge of the application itself. The need for providing high-level and comprehensive tools that enable security experts to operate in the model domain and focus on security measures without worrying about the low-level implementation issues has been discussed.

A comprehensive study was done on the topic of the adoption of AOM techniques with UML models for addressing the security concerns and they have been cited in the chapter on literature survey. It was found that there is no comprehensive tool available for providing this high-level support in its entirety to the security expert.

Definition and development of a model-based framework for engineering secure software (MOBS2) was initiated in collaboration with Ericsson Canada. The main focus of this thesis work has been to provide support in MOBS2 for definition of certain specific types of pointcuts, identify corresponding join points and incorporate the required advices at these points in the selected UML models: Class diagrams, Lifelines and sequence diagrams. In addition, mapping functions were defined using QVTO to

`

transform the base UML model to one with the advice weaved in it. The efficacy and usefulness of these new security constructs were illustrated using two sample applications namely the SIP Communicator and RBAC.

The key contributions of this research are 1) demonstrating how to build on and expand UML profiles to specify new security requirements as aspects over design models rather than as an after-thought; 2) offering guidelines to elaborate the model transformation rules that allow for weaving security aspects into UML design models at the very early stages of as well as throughout the various phases of the development life cycle; 3) illustrating how to design and implement the security aspects to UML models within Rational Software Architect environment; and 4) validating the feasibility of our approach in the context of a variety of case studies, including in a banking transaction as well as in multimedia communication environments to contend with and resolve a variety of security challenges.

Specifically, in this research work, we introduced new components to the weaver application for specifying improved security hardening concepts to aid the integration of cross-cutting concerns, and security aspects into UML design models. We presented a detailed comparative study of the various techniques in security hardening of software design models. We highlighted that there is a clear need to provide security experts with a more expressive and generic AOM language for the specification of security hardening solutions on both structural and behavioral UML diagrams given a glaring lack of such capability at this time. As a result, we elaborated UML profiles, which allow the specification of aspects for adaptation in the weaver. We also discussed in detail the

`

MOBS2 framework that currently employs the model-to-model transformation concept to efficiently automate the weaving process. In addition, we discussed UML-specific pointcut language to designate the main UML join points.

The premise of this thesis work revolved around the need, motivation, and capability for designing additional components based on *object-oriented principles*. These principles allow the security experts to use along with their traditional modeling methodoly that results in an extension of the weaving process and allows the security expert to employ improved security hardening concepts. As noted above, we conducted case studies to demonstrate the effectiveness of incorporating the additional components to the weaver application. Demonstrating the effectiveness of the additional components to the SIP communicator base model showed the improvement in security hardening concepts in both structure as well behavior of the SIP model. Using this research as a springboard, future research can examine how the framework and applications presented here can be further extended to accommodate more components that allow security experts to create a more secure, functional and reduced vulnerable systems environment.

`

Bibliography

[1] Microsoft Security Intelligence Report, "The evolution of malware and the threat landscape – a 10-year review," www.microsoft.com/sir, 2012.

[2] www.fortify.com, "The Case for Application Security," Cigital Labs - Fortify Software Inc., San Mateo CA, 2008.

[3] Fortify, "The Case for Application Security," Fortify Software, San Mateo, CA, 2008.

[4] J.-P. K. Christel Baier, Principle Of Model Checking, Cambridge , MA: MIT Press, 2008.

[5] Soley, Richard; OMG Staff Strategy Group, "Model Driven Architecture," Object Management Group, Needham, MA , 2000.

[6] Object Management Group, Inc. , "Introduction to OMG's UML- Unified Modeling Language," OMG, 1997-2013. [Online]. Available: http://www.omg.org/gettingstarted/what_is_uml.htm. [Accessed 14 Febuary 2013].

[7] R. Fernández , F. L. Nicolet and R. Carniel, "UML and Model Engineering," *The European Journal for the Informatics Professionals,* vol. 5, no. 2, 2004.

[8] M. M. Kandé, J. Kienzle and A. Strohmeier, "From AOP to UML: Towards an Aspect-Oriented Architectural Modeling Approach," Software Engineering Laboratory , Swiss Federal Institute of Technology , Lausanne, 2002.

`

[9] P. Meunier, "Software Properties and Behaviors," CERIAS, Purdue University, 2008.

[10] I. Traor´e and Y. M. Liu, "Properties for Security Measures of Software Products," Dixie W publishing Corporation, 2007.

[11] D. Wu, "Security Fuctional Requirements Analysis for Development," ProQuest Information and Learning Company, Ann Arbor, MI, May 2007.

[12] G. Booch, J. Rumbaugh and I. Jacobson, "Unified Modeling Language," in *The Unified Modeling Language User Guide*, India, Dorling Kindersley, 2006, pp. 44-58.

[13] B. Unhelkar, Verification and Validation of Quality of UML 2.0 Models, Wiley-Interscience, 2005.

[14] A. Bader, T. Elrad and O. Aldawud, "A UML Profile for Aspect Oriented Modeling," Illinois Institute of Technology & Lucent Technologies, Chicago, 2002.

[15] J. Farhad, "The UML Extension Mechanisms," University College London, London, 2002.

[16] A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, M. Wimmer and K. G, "A Survey on Aspect-Oriented Modeling," WIT Women's Postgraduate College for Internet Technologies, Veinna, 2006.

[17] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W. G. Griswold, "An Overview of AspectJ," Xerox Palo Alto Research Center, La Jolla, CA, 2001.

[18] Graz University Of Technology, "Software Paradigms (Lesson 9) - Universal Modeling Language (UML)," AG Networked Learning, 2010. [Online]. Available:

`

http://coronet.iicm.tugraz.at/sa/scripts/lesson09.htm. [Accessed 1 March 2013].

[19] K. Padayachee and N. Wakaba, "A Taxonomy Of Aspect-Oriented Security," *Review of Business Information Systems – First Quarter 2008,* p. Volume 12, 2008.

[20] P. Charles P, Security in Computing, New Jersy: Pearson Education Inc., 2003.

[21] B. D. Win, W. Joosen and F. Piessens, "Developing Secure Applications through," Department of Computer Science, Leuven, Belgium, October 31, 2002.

[22] J. Pavlich-Mariscal, L. Michel and S. Demurjian, "Enhancing UML to Model Custom Security," Department of Computer Science & Engineering, The University of Connecticut, Connecticut, 2007.

[23] J. Ramachandran, Designing Security Architecture Solutions, New York: Wiley Computer Publishing,Robert Ipsen, 2002.

[24] J. P. Anderson, "Computer Security Technology Planning Study," USAF, Massachusetts, October 1997.

[25] M. Engel, "Advancing Operating Systems via Aspect-Oriented Programming," Philipps-Universität, Marburg, 2005.

[26] S. Clarke and E. Baniassad, "Theme: An Approach for Aspect-Oriented Analysis and Design," Trinity College, Dublin, 2004.

[27] L. Fuentes and P. S´anchez, "Elaborating UML 2.0 Profiles for AO Design," University of M´alaga, Spain, 2007.

[28] P. Gupta, S. Garg and K. Khalon, "Designing Aspects Using Various UML Diagrams in

`

Resource-Pool Management," *Advanced Engineering Science and Technologies,* vol. 7, no. 2, pp. 228 - 233, 2011.

[29] T. Cottenier, A. Van Den Berg and T. Elrad, "Modeling Aspect-Oriented Compositions," CISE NSF & Motorola Labs., 2006.

[30] A. Schmidmeier, S. Hanenberg and R. Unland, "Implementing Known Concepts in AspectJ," Institute for Computer Science, Herbruck, German.

[31] A. Bustos and Y. Eterovic, "Modeling Aspects with UML's Class, Sequence, and State-Diagrams in an Industrial Setting," Pontificia Universidad Católica de Chile, Santiago,Chile, 2007.

[32] R. Pawlak, L. Duchien, G. Florin, F. Legond-Aubry, L. Seinturier and L. Martelli, "A UML Notation for Aspect-Oriented Software Design," Laboratoire LIFL, Paris, France, March 4, 2002.

[33] M. Basch and A. Sanchez, "Incorporating Aspects into the UML," Department of Computer and Information Sciences, Jacksonville,FL, December, 2002.

[34] M. M. Kandé, J. Kienzle and A. Strohmeier, "From AOP to UML - A Bottom-Up Approach," Software Engineering Laboratory ,Swiss Federal Institute of Technology, Lausanne, Switzerland, 2002.

[35] S. A. Khan and A. Nadeem, "UML Extensions for Modeling of Aspect Oriented Software: A Survey," Center for Software Dependability,MAJU, Islamabad, Pakistan, 2010.

[36] L. Fuentes and P. S´anchez:, "Designing and Weaving Aspect-Oriented Executable

UML Models," *Journal of Object Technology,* vol. 6, no. 7, pp. 109-136, August, 2007.

[37] M. Nouh, "Model-To-Model Transformation Approach For Systematic Integration Of Security Aspects into UML2.0 Design Models," Concordia University, Montreal, Quebec, Canada, July 2010.

[38] J. Zhang, T. Cottenier, A. van den Berg and J. Gray, "Journal of Objet Technology," *Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver,* vol. 6, no. 7, pp. 89 - 108, August, 2007.

[39] B. Morin, J. Klein, O. Barias and J.-M. Jézéquel, "A Generic Weaver for Supporting Product Lines," in *"International Workshop on Early Aspects at ICSE'08*, Leipzig, Germany, May 12, 2008.

[40] I. Groher and M. Voelter, "XWeave: Models and Aspects in Concert," ACM, Vancouver, BC, 2007.

[41] P. H. Krishna, A. Ngaraju and D. Kumar, "Secure Software Architectures Using Aspect Oriented Model," *International Transactions in Mathematical Sciences and Computer,* vol. 3, no. 1, pp. 67-72, 2010.

[42] D. Matheson, "Security Concerns in an Aspect-Oriented Modeling Approach," Computer Science Department Colorado State University, Fort Collins, Colorado, 6. March 2005.

`

[43] V. N. De Lima, "Framework for Automatic Verification of UML Design Models: Application to UML 2.0 Interactions," Information Systems Engineering, Concordia University, Montreal, QC, 2010.

[44] I. Ray, R. France, N. Li and G. Georg, "An aspect-based approach to modeling access control concerns," *Information and Software Technology,* no. 46, pp. 575-587, 25 April 2003.

[45] Cisco Systems, Inc, "Security in SIP-Based Networks - White Paper," Cisco Systems, 2002. [Online]. Available: http://www.cisco.com/warp/public/cc/techno/tyvdve/sip/prodlit/sipsc_wp.pdf.

[46] M. Ranganathan, P. O'Doherty, J. v. Bemmel, S. Gallanos and B. Evans, "jsip.java.net," December 2010. [Online]. Available: https://java.net/projects/jsip , http://www.eetimes.com/design/embedded-internet-design/4209253/Internet-multimedia-communications-using-SIP---Part-1--The-JAIN-SIP-API?pageNumber=2.

[47] A. Lakas and E. Barka, "Integrating Usage Control woth SIP-Based Commuications," College of Information Technology, UAE, 2008.

[48] E. J. Khayat and A. E. Abdallah, "A Formal Model for Flat Role-Based Access Control," Centre for Applied Formal Methods, London, South Bank University.

[49] D. Mouheb, "Model-Driven Aspect-Oriented Software Security Hardening," Computer Security Laboratory / Concordia Institute for Information Systems Engineering, Montreal, QC, 2010.

`

[50] K. Sohr, G.-J. Ahn* and L. Migge, "Articulating and Enforcing Authorisation Policies with UML and OCL," ACM, St. Louis, Missouri, USA, 7. July,2005.

[51] D. Basin, M. Cavel and M. Egea, "A Decade of Model-Driven Security," ACM 978-1-4503-0688-1/11/06, Innsbruck, Australia, 2006.

[52] R. M. Perea, "Internet Multimedia Communications Using SIP : A Mordern Approach Including Java Practices," Burlington, Morgan Kaufmann, 2008, pp. 150 - 312.

[53] K. Wang and R. Liscano, "A SIP-based Architecture model for Contextual Coalition Access Control for," School of Information Technology and Engineering, University of Ottawa, Ottawa, 2005.

[54] R. Telang, D. Hall, A. Arora, C. Piato and A. Ramsey, "Measuring Risk-Based Value of IT Security Solutions," IT Professional, Nov - Dec 2004.

[55] K. Soo Hoo, A. W. Sudbury and A. R. Jaquith, "Return on Security Investments," *Secure Business Quarterly,* 2002.

[56] B. W. Boehm, J. R. Brown and M. Lipow, "Quantitative evaluation of software quality," in *International Conference on Software Engineering - ICSE*, Zurich, Switzerland, 1976.

[57] N. Xie and N. R. Mead, "SQUARE Project: Cost/Benefit Analysis Framework for Information Security Improvement Projects in Small Companies," Carnegie Mellon University (CMU/SEI-2004-TN-045), Pittsburgh, Pennsylvania, November 2004.

`

[58] C. Baier, J.-P. Katoen and . K. Guldstrand, Principles of model checking, 55 Hayward Street, Cambridge, MA: The MIT Press, 2008.

[59] E. Colbert, D. Wu and Y. Chen, "Costing the Development of Secure Systems," *18th Annual COCOMO II, Software Costing Forum & 8th Annual PSM Users' Group Conference,* pp. 26-30, July 2004.