การสร้างมโนทัศน์แบบจำลองความต้องการเชิงลักษณะ

นายชาญวิทย์ แก้วกสิ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2545

ISBN 974-17-2370-9

CONCEPTUALIZATION OF ASPECT-ORIENTED REQUIREMENTS MODEL

Mr. Chanwit Kaewkasi

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2002

ISBN 974-17-2370-9

Thesis Title	Conceptualization of Aspect-Oriented Requirements Model
By	Mr. Chanwit Kaewkasi
Field of Study	Computer Engineering
Thesis Advisor	Associate Professor Wanchai Rivepiboon, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master 's Degree

..........[illegible].......... Dean of Faculty of Engineering

(Professor Somsak Panyakeow, D.Eng.)

THESIS COMMITTEE

..........[illegible].......... Chairperson

(Assistant Professor Korbkul Tejavanija)

..........[illegible].......... Thesis Advisor

(Associate Professor Wanchai Rivepiboon, Ph.D.)

..........[illegible].......... Member

(Assistant Professor Wiwat Vatanawood)

..........[illegible].......... Member

(Lecturer Arthit Thongtak, D.Eng.)

..........[illegible].......... Member

(Assistant Professor Suyut Satayaprakorb)

ชาญวิทย์ แก้วกสิ : การสร้างมโนทัศน์แบบจำลองความต้องการเชิงลักษณะ (CONCEPTUALIZATION OF ASPECT-ORIENTED REQUIREMENTS MODEL) อ.ที่ปรึกษา : รองศาสตราจารย์ ดร. วันชัย ริ้วไพบูลย์, 78 หน้า. ISBN 974-17-2370-9.

จากข้อเท็จจริงที่แนวทางเชิงลักษณะสามารถแก้ปัญหาเกี่ยวกับการตัดขวาง ในการพัฒนาซอฟต์แวร์ ซึ่งไม่สามารถแก้ได้โดยใช้แนวทางเชิงวัตถุได้นั้น ผู้พัฒนาจึงได้นำแนวทางเชิงลักษณะ มาปรับใช้กับจัดการความต้องการซอฟต์แวร์ ซึ่งเป็นสิ่งสำคัญในกระบวนการการพัฒนาซอฟต์แวร์ ความต้องการซอฟต์แวร์จึงควรที่จะได้รับการจัดการโดยใช้แบบจำลองเชิงลักษณะเพื่อที่จะสนับสนุนรูปแบบเชิงลักษณะนี้ในระยะถัด ๆ ไปของการพัฒนา

วิทยานิพนธ์นี้เสนอแบบจำลองความต้องการที่ขยายแนวทางการพัฒนาซอฟต์แวร์ที่ขับเคลื่อนด้วยยูสเคส แบบจำลองนี้สนับสนุนกระบวนการพัฒนาซอฟต์แวร์แบบยูนิฟายด์ และเสนอกระบวนการที่ใช้คู่ขนานไปกับกระบวนการความต้องการของกระบวนการยูนิฟายด์ วิทยานิพนธ์นี้นำเสนอแผนผังระดับความต้องการแบบใหม่ ที่เรียกว่าแบบจำลองกองซ้อนของการตัดขวาง โดยได้รวมกลุ่มของสัญกรณ์เพื่อใช้จำลองแบบลักษณะที่สกัดออกมาจากกระบวนการการชำระยูสเคสและการสกัดลักษณะ ผู้วิจัยได้พัฒนาเครื่องมือชื่อ อาสเร็ม เพื่อสนับสนุนการทำงานกับแผนผังดังกล่าว นอกจากนี้วิทยานิพนธ์นี้ได้นำเสนอแผนผังสำหรับการอธิบาย เรียกว่าแผนที่สำหรับวัตถุ/การตัดขวาง/ยูสเคส โดยรวมเอาสัญกรณ์เพื่อใช้ในกระบวนการการอธิบายยูสเคสและลักษณะและกระบวนการการทำให้ลักษณะเป็นจริงไว้ด้วยกัน ผู้วิจัยได้พัฒนาเครื่องมือชื่อ ออคคุม เวคตรา เพื่อสนับสนุนการทำงานกับแผนผังนี้ โดยเครื่องมือทั้งสองข้างต้นมีคุณสมบัติเป็นตัวเสริมให้กับโปรแกรม เรชันแนล โรส

ภาควิชา...วิศวกรรมคอมพิวเตอร์..	ลายมือชื่อนิสิต..
สาขาวิชา..วิศวกรรมคอมพิวเตอร์..	ลายมือชื่ออาจารย์ที่ปรึกษา..
ปีการศึกษา .....2545.............	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม......................................

4470696821 : MAJOR COMPUTER ENGINEERING

KEY WORD: REQUIREMENTS ENGINEERING / ASPECT-ORIENTED / UML

CHANWIT KAEWKASI : CONCEPTUALIZATION OF ASPECT-ORIENTED REQUIREMENTS MODEL. THESIS ADVISOR : ASSOC. PROF. WANCHAI RIVEPIBOON, Ph.D., 78 pp. ISBN 974-17-2370-9.

Due to the fact that aspect-oriented (AO) paradigm can solve the problem of crosscutting concerns in software development. And this kind of software cannot be solved properly by the object-oriented approach. Software requirements, which are the important artifacts in the development process, should be managed using a model for AO to fully support this paradigm at the later development stages.

This thesis proposes a requirements model that extends the use-case driven approach. This model supports the unified process. It introduces a new parallel process to the requirements workflow of the unified process. This thesis presents a new requirements-level diagram called the Crosscutting Stack Model, including a set of notations for modeling the aspects that are extracted by the use-case purification and aspect extraction process. This thesis also presents a scenario descriptive diagram called the Object/Crosscutting/Use Case Maps, including a set of notations for use-case and aspect explanation in the aspects realization process. The modeling tool, named ASREM, is developed to support the CMS diagram. The modeling tool, named OCUM Vectra, is developed to support the OCUM diagram. Both modeling tools are developed as Rational Rose add-in.

Department....Computer Engineering....	Student's signature..
Field of study...Computer Engineering....	Advisor's signature..
Academic year ..2002......	Co-advisor's signature.....................................

ACKNOWLEDGEMENTS

This thesis could not be complete without help and support from others. I would like to thank my thesis advisor, Associate Professor Dr. Wanchai Rivepiboon, for his supports, comments, suggestions and reviews since I started to do this thesis until it is completed. I would like to thank Assistant Professor Korbkul Tejavanija for her reviews, and useful comments. I also thank Assistant Professor Wiwat Vatanawood, Dr. Arthit Tongtak, and Lecturer Chate Patanothai. I am thankful for the great support from the School of Computer Engineering, Suranaree University of Technology, and Assistant Professor Suyut Satayaprakorb.

Finally, I would like to thank all persons that I cannot mention here, and I dedicate this thesis to my parents for their best support while I am working on this research.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF TABLES

TABLE OF FIGURES

TABLE OF FIGURES (CONTINUED)

CHAPTER 1
INTRODUCTION

1.1 Problem Statements and Motivation

Researches that are related to requirements engineering (RE) have been increasingly studied. It has long been known that RE is an important role in the software development process. Efficiency use of RE can reduce overall cost for the later stages of the development [1, 2, 3]. Recently, aspect-oriented software development (AOSD) [4] has influenced the current software development processes. AOSD proposes the other concern of the software development that can be used with the object-oriented (OO) approach to lead the software process into the new era.

AOSD proposes crosscutting concerns for extracting tangled things among software artifacts into a new modular unit called aspect. The intension of AOSD is to increase more modularization to the software. AOSD covers several phases of the development. Firstly, it has been introduced as the aspect-oriented programming (AOP) [5] in the implementation phase. After that AOSD has played a role in the design phase. Several more recent works introduced an aspect into the design diagram, such as in the Unified Modeling Language (UML) [6]. In those recent works the authors proposed a new kind of the UML classifier to represent the concept of an aspect for using in the class diagram. AOSD is not intended to replace the OO software development. It is to support the OO approach. For example, the AOP language AspectJ [7], which is an AOP implementation for Java, knows the traditional Java code, while that Java code does not know any existence of its AO code as illustrated in Figure 1.1. This is one of advantages of the AOP that it can be used to support the traditional code without any modification of the old software.

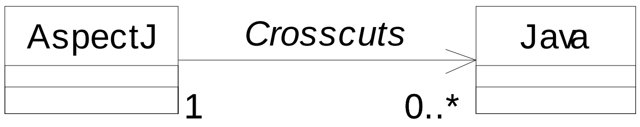


Figure 1.1 Relationship between AspectJ and Java code

The influence of AO popularity affects the RE as well. Recent works proposed a number of approaches that applied the AO paradigm to the requirements phase. There are several models such as the early aspects model proposed by Rashid and et al [8]. Their approach is based on the viewpoint-oriented requirements engineering [2]. The authors proposed a model to identify candidate aspects. Six activities stated in that work includes specifying of aspect dimension that associates candidate aspects with software artifacts that they influence, and will be mapped to. The work proposed only the model and did not clearly specify how to map their candidate artifacts to the late stage of the software development. In more recent work, the authors [9, 10] extended their previous work with composing aspects, which are nonfunctional properties, to the UML use-case model. Additional stereotypes were introduced in that work.

The use-case driven [1, 3] is a software development approach that supports the use of the UML [6] and the object-oriented technology. It does not support the modeling of other kinds of concern except services of the system. In fact, there are several kinds of concern that can be found from stakeholders' requirements. And those concerns are finally mapped to parts of the software system and usually implemented using some kinds of object-oriented technology including OOP. AOP has matured to be recognized as a programming technique that can increase maintainability and adaptability of the software system. To utilize the use of AOP, the early identification of a software artifact that will be represented as an aspect using the AO paradigm should be considered. The identification should be processed in the early step of the development, the requirements phase. This motivates us to propose this work for extracting those artifacts out of the use-case model to support AOSD and make it possible to use with the use-case driven approach.

Our approach alternatively presents aspect-oriented techniques that are designed for the requirements phase of the software development. The work will propose an aspect-oriented requirements model that covers both functional and nonfunctional requirements for the use-case driven approach. Generally, nonfunctional requirements (NFR) usually are quality attributes of the system [2]. There is currently no any set of notations that represents NFR in the UML use-case model [6]. The unified process [1, 3] suggests keeping NFR as supplementary documents.

From the studies, it has been found that not only NFRs crosscut the use case of the system, but also some kinds of functional requirements. This idea was also stated in [8] that aspects can also be functional concerns. Functional requirements usually are considered services of the system. Functional requirements can be classified into, at least, three classes. They are primary, secondary, and optional services. Our approach assumes that primary services of the system cannot crosscut other services. There are several functional requirements that cut across the primary services of the system as illustrated in Figure 1.2, and our approach is to model them.

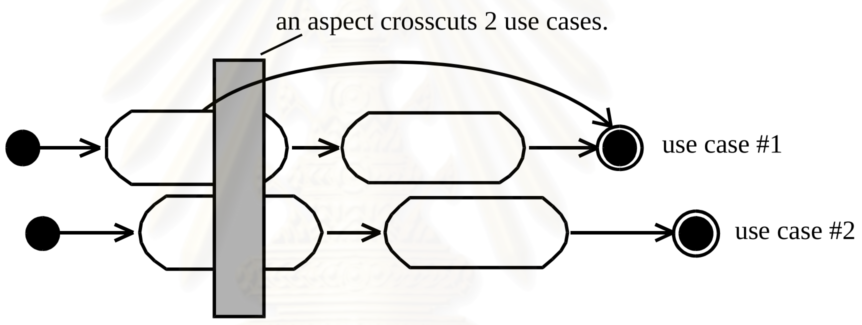


Figure 1.2 Crosscutting behavior of an aspect across two use cases

Additionally, this approach is also intended to model some kinds of NFR. This work will provides several contributions as follows. First, this work proposed processes that make the preliminary use cases, captured from a software requirement specification, to be more *purified*. Our approach is intended to extract aspects from those use cases and model them into a diagram called crosscutting stack model (CSM). Second, this work proposes a set of graphical languages that extend a notation of the UML to use with our purification and extraction processes. The new UML extension will be used in the CSM. Third, this work additionally proposes a process to realize model elements in CSM for using in the next phases of the software development process. This is to create an aspect-oriented design model for further supporting the code mapping to aspect-oriented programming language.

1.2 Objective

To design, and develop a new requirements model that supports AOSD to help requirements engineers capturing requirements in the aspect-oriented paradigm.

1.3 Scopes of the Research

The scopes of the research are as follows.

1.3.1 The research covers only functional and nonfunctional crosscutting artifacts. Pseudo requirements are not considered in this work.

1.3.2 A complexity index will be used to compare a traditional use-case model with our approach.

1.3.3 A case study is in the Web application problem domain.

1.3.4 The research will not explicitly concern about the impact of requirements changes.

1.3.5 The concerned software development process in this research is the Unified Process that employs the use-case driven approach.

1.3.6 Resulting software tools will be implemented as add-in applications of the Rational Rose.

1.4 Steps of Research

This research follows the steps as follows.

1.4.1 Study related works.

1.4.2 Design a new technique and a set of notations, and a software tool.

1.4.3 Apply the approach to a case study.

1.4.4 Compare our techniques to the traditional approach.

1.4.5 Conclusions.

1.5 Contribution

This research has contributions as follows.

1.5.1 A technique for purifying and extracting aspects from the use-case model.

1.5.2 A set of aspect-oriented notations that extend the use-case package of the UML with a software tool.

1.5.3 A complexity index for measurement of a use-case model.

1.5.4 A technique to realize an aspect-oriented use-case model to other phases of the software development with a software tool.

The next chapter will discusses the related works, especially several requirements models proposed for the AO paradigm, and the related theories.

CHAPTER 2
LITERATURE REVIEWS

2.1 Related Works

This section discusses several previous works in aspect-oriented requirements engineering that directly relate to the work proposed in this thesis.

2.1.1 Early Aspects: An Aspect-Oriented Requirements Model

Early aspects model, proposed by Rashid et al [8], suggested that it is necessary to include aspects as the primitive modeling at the requirements-level of the software process. The important objectives of that work are:

- To support separation of crosscutting concerns those are functional and nonfunctional properties of the system, identify, and manage conflictions of these tangled representations.
- To present mapping and influence properties of those requirements-level aspects for the later stages of the development.

Their approach introduced a model that employs viewpoint-oriented requirements engineering [2] as an underlying methodology. The model consists of six activities described in Figure 2.1. The process of the model starts by identifying concerns and discovering requirements. Both activities can be repeated before stepping to the next activity. The works in those activities are recommended by the authors that requirements engineers and stakeholders should perform them. Relating the concerns to the requirements is useful as the concerns may constraint the requirements. The next step is to specify concerns for providing more details. If a concern crosscut several requirements, it is considered a candidate aspect. Specifying the detail of candidate aspects in the next activity is to refine, make them more concrete, and identify interactions and conflictions among them. To resolve the conflictions, prioritizing those aspects should be done. Identifying mapping and influence dimension of the aspects is the last activity of the model.

Aspects in the early stage can have an impact to the system that can be identified as two dimensions, mapping and influence.

- Mapping: requirements-level aspects can be mapped to other artifacts of the system. Aspects may be mapped to, for example, functional, simple methods, decision of architecture choice, design or implementation details, or other system properties. Because of this mapping principle, aspects at the requirements-level are called candidate aspects.

- Influence: aspects may influence to several points, and phases in the development cycle. For example, availability aspect of the system influences the system architecture while response-time aspect influences both the system architecture and the detailed design of the system.

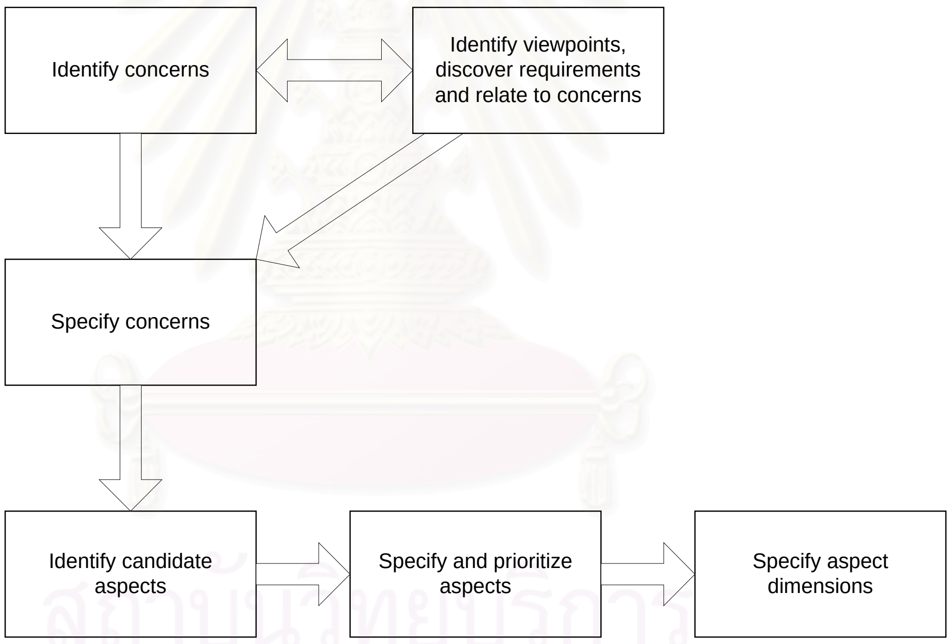

Identify concerns
Identify viewpoints, discover requirements and relate to concerns
Specify concerns
Identify candidate aspects
Specify and prioritize aspects
Specify aspect dimensions

Figure 2.1 Rashid et al's Aspect-oriented requirements model [8]

Their approach just offered a model and guideline. This approach describes aspects in the form of a problem frame, and a plain text. It is quite difficult to manage those aspects, especially in complex systems, because there is no notation or modeling tool to support this approach.

2.1.2 A Model for Early Quality Attributes with UML

The recent work proposed by Moreira et al [10] presented an approach to include quality attributes to the use case model. This model is to identify and specify quality attributes that crosscut the requirements. It includes the systematic processes to integrate those quality attributes into the functional requirements captured as use cases at the early stages of the development process. The model is a UML compliant process and is composed of three main activities: identification, specification, and integration of requirements. The process overview is illustrated in Figure 2.2.

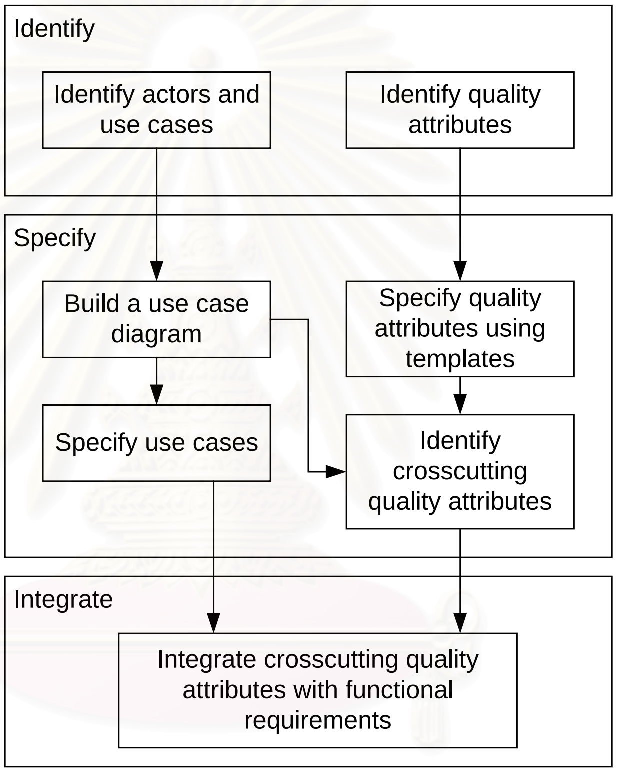

Identify
Identify actors and use cases
Identify quality attributes
Specify
Build a use case diagram
Specify quality attributes using templates
Specify use cases
Identify crosscutting quality attributes
Integrate
Integrate crosscutting quality attributes with functional requirements

Figure 2.2 A requirements model for quality attribute [10]

The first activity is to identify all quality attributes relevant to the application domain from all requirements. The second activity can be divided into two main parts: 1) specifying use cases, and specifying quality attributes using the special templates; 2) identifying crosscutting quality attributes from the attributes in the templates. The third activity is to integrate crosscutting quality attributes with functional requirements capturing as use cases.

The authors stated that the special template for specifying crosscutting quality attributes was inspired from Chung et al [11] and Malan and Bredmeyer [12]. The template is in Table 2.1.

Table 2.1 A template for specifying quality attributes [10]

Name	The name of the quality attribute
Description	Executive description
Focus	A quality attribute can affect the system (i.e. the end product) or the development process
Source	Source of information (i.e. stakeholders or documents)
Decomposition	Quality attributes can be decomposed into simpler ones. When all (sub) quality attributes are needed to achieve the quality attribute, we have an AND relationship. If not all the sub quality attributes are necessary to achieve the quality attribute, we have an OR relationship
Priority	Expresses the importance of the quality attribute for the stakeholders. A priority can be MAX, HIGH, LOW, and MIN
Obligation	Can be optional or mandatory
Influence	Activities of the software process affected by the quality attribute
Where	List of the actors influenced by the quality attribute and also a list of models (e.g. use cases and sequence diagram) requiring the quality attribute
Requirements	Requirements describing the quality attribute
Contribution	Represents how the quality attribute affects other quality attributes. This contribution can be positive (+) or negative (-)

The properties to help identifying what quality attribute is crosscutting are at the rows Where, and Requirements from template. If those properties indicate that the quality attribute traverses several models and requirements, then it is crosscutting.

2.1.3 Aspect-oriented requirements with UML

The work by Araújo et al [9] presented an approach to manage crosscutting concerns at the requirements stage using the UML [6]. The authors reported that their approach could be a mechanism to help requirements engineers managing and understanding the whole system requirements. According to [8], the crosscutting concerns can also be functional and nonfunctional. But the work reviewed here proposed only techniques for composing nonfunctional aspect to the use-case model. The aspect-oriented requirements engineering model from [8] was used with slightly modification to make the model possible to use with the UML. Figure 2.3 shows the requirements model.

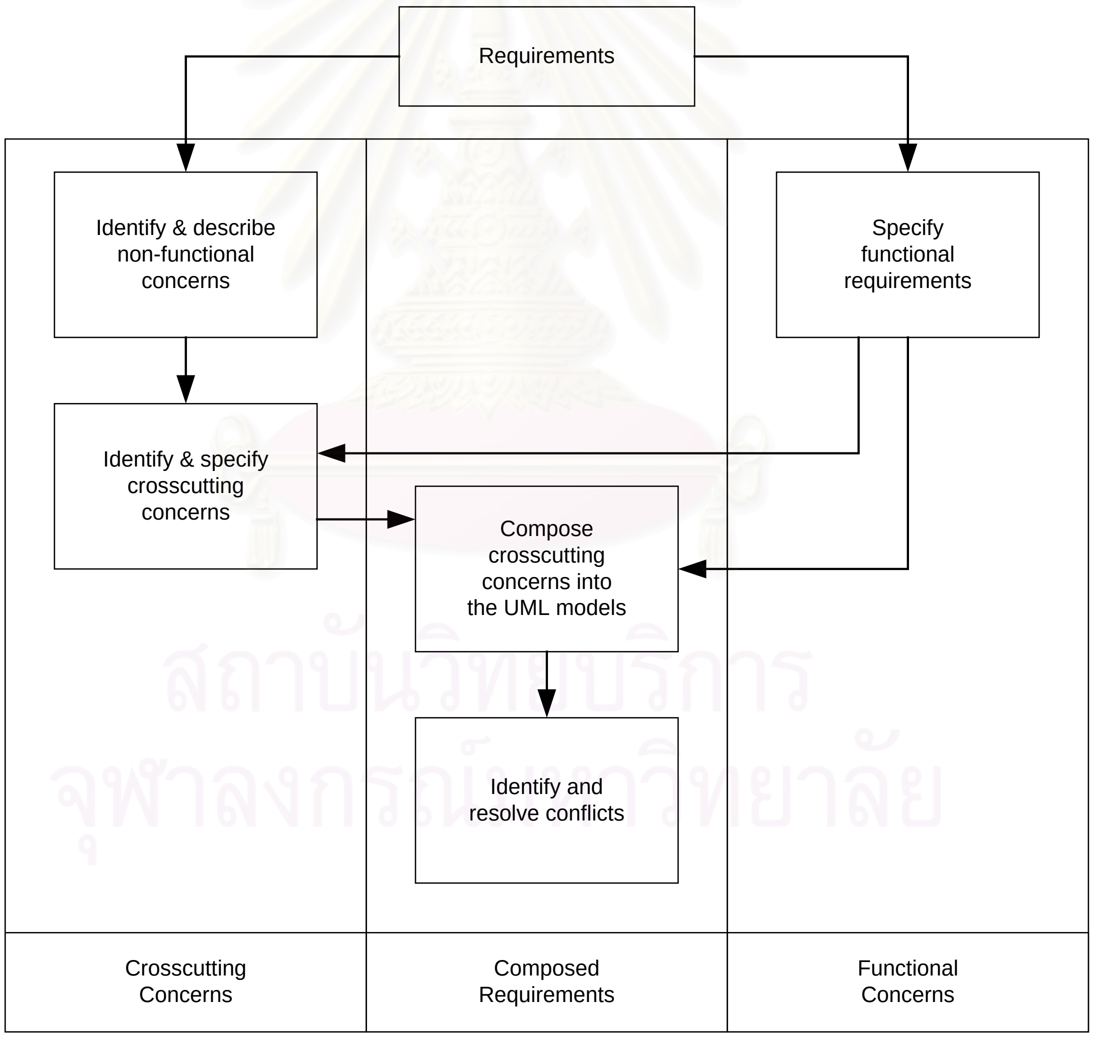

Requirements
Identify & describe non-functional concerns
Specify functional requirements
Identify & specify crosscutting concerns
Compose crosscutting concerns into the UML models
Identify and resolve conflicts
Crosscutting Concerns
Composed Requirements
Functional Concerns

Figure 2.3 A model for composing aspect-oriented requirements with UML [9]

2.1.3.1 Model Partition

The process is partitioned in three main parts, crosscutting concerns, functional concerns, and composed requirements.

- Crosscutting concerns: this part handles identifying and describing of non-functional concerns, and then identifies which of those are crosscutting. If a non-functional concern crosscut several requirements, then it is a candidate aspect.
- Functional concerns: this part contains an activity for identifying and specifying functional requirements.
- Composed requirements: this part takes functional requirements and crosscutting concerns as its input. Composing crosscutting concerns to the UML models that are functional requirements are performed in the first activity of this part. The final activity in this part is the process of identifying and resolving conflictions that may be raised by composing those crosscutting concerns to the functional requirements.

2.1.3.2 Composing Parts

That work proposed modeling composed requirements in the use-case model. Composed requirements are functional requirements that are composed with candidate aspects. Functional requirements as use cases in the use-case model are associated with aspects. The association is attached with information that provides a composition semantic as its stereotype. The authors suggested three composition parts as follows.

- Overlapping: the requirements of aspect modify the functional requirements that they crosscut. The behavior of aspect partially substitutes at the beginning and the end of the basic requirements.
- Overriding: the requirements of aspect superpose the functional requirements that they crosscut. The behavior of aspect fully substitutes the basic requirements.

- Wrapping: the requirements of aspect encapsulate the functional requirements that they crosscut. The behavior of aspect wraps, before and after, the basic requirements.

That work also proposed a specification frame for describing crosscutting concerns. The specification table is illustrated in Table 2.2.

Table 2.2 Specification of crosscutting concerns [9]

Crosscutting concern	<Name>
Description	<Executive description>
Priority	<Max, Med, Min>
List of requirements	<Requirements that describe the concern>
List of UML	<UML models influenced by the concern>

In that paper, the authors pointed that composing aspects with use cases may raise conflictions among those aspects. Thus, the further consideration is a process of resolving those conflictions.

The case study illustrated in the work reviewed here are the simplified version of the Portugese motorways network [13]. Figure 2.4 shows the use-case model of that system. Figure 2.5 shows the composed UML model with the "Toll gate response time" aspect. Composing the aspect was done by associating the aspect notation (the use case with <<TollGateResponseTime>> stereotype) with three basic use cases.

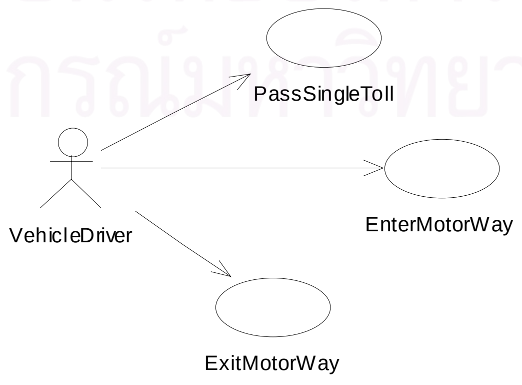


Figure 2.4 The use-case diagram of the toll gate collecting system [13]

Table 2.3 from [9] shows an instance of template specification of the composed requirement modeled as in figure 2.5. There are four requirements relating to the "Toll gate response time" concern. Those requirements was described in [9] and not showed in this review. It is clearly observed that there are three use cases in the composed model. Those use cases are also illustrated in the template.

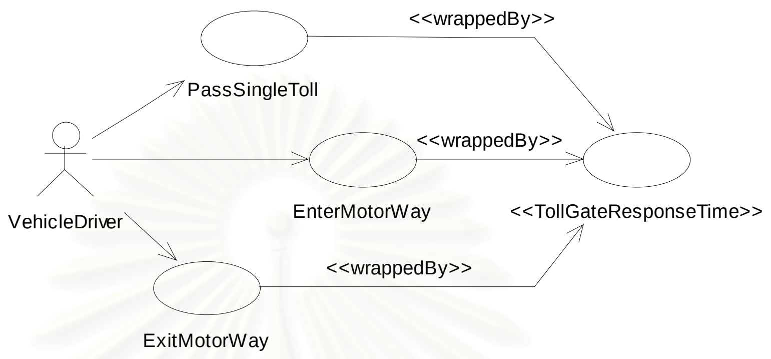


Figure 2.5 The use cases composed with the aspect [9]

Table 2.3 Crosscutting template specification for toll gate response time [9]

Crosscutting concern	Toll gate response time
Description	Tollgates should react before the driver leaves the toll gate area
Priority	Max
List of requirements	R1, R2, R3, R4
List of models	Usecases: 1. PassSingleToll, 2. EnterMotorway, 3. ExitMotorway

Although, the model introduced by [8] has suggested that functional requirements may crosscut the basic services of the system, but the work reviewed here did not cover that kind of aspect. Besides, they introduced composing technique to use UML as base notations for describing aspects in the use-case model, but it is clearly observed that the approach increases complexity to the use-case model when modeling several aspects.

2.2 Related Theory

This section discusses background knowledge that will be referred by the further sections.

2.2.1 Aspect-Oriented Programming

Aspect-oriented programming (AOP) [5] has been recognized by the software development communities to be one of important technologies. It is expected by the community that AOP will mature enough to change the programming style in the near future. Many communities reported that AOP could solve some kind of programming problems properly than OOP [4, 7]. AOP reduces like of codes significantly comparing to OOP for the same task. It additionally provides more maintainability and adaptability to the software system. Programming with AOP is a process that separates tangled codes out of the software. Those tangled codes may spread across classes, and other points of the source program. In [4, 5, 7] the authors reported that it is necessary to manage the tangled codes because it affects the maintainability of the system. Besides, it caused some hidden bugs that are difficult to find without the use of AOP. AOP groups those kinds of code into a new modular unit called *an aspect.* This makes the source program cleaner and easier to maintain, and evolve when the requirements are changed. Aspects can be compiled back to be the software using an aspect compiler called *a weaver*. This process is called a weaving process. The weaver weaves aspects source with the traditional source code. The resulting source code called the woven source can now be compiled with the traditional compiler to produces executable program.

There are a number of compilers that support AOP. Many of them can be found on then AOSD communities [4], but one of the most famous aspect compilers is AspectJ [7], which is aspect weaver for Java [14] programming language. It is acceptable to use language features of AspectJ as AOP semantic references because the maker of it and the authors of AOP paper came from the same organization. AspectJ has introduced a number of language features to support AOP semantics as follows.

2.2.1.1 Join point abstraction: the abstraction concept of join points has been introduced in the AOP. Join points are well-defined points in the execution flow of the program.

2.2.1.2 Pointcut designators: pointcut designators, or pointcuts for short, are selected join points. Those joint points are grouped and may be describe using regular language.

2.2.1.3 Advice codes: they are code fragment that will be performed by their associated pointcuts.

2.2.1.4 Introductions: extra data or method members can be late introduced to the classes using the concept of introduction. After weaving an aspect to the old code, introducing data or method members will be automatically added to the woven class.

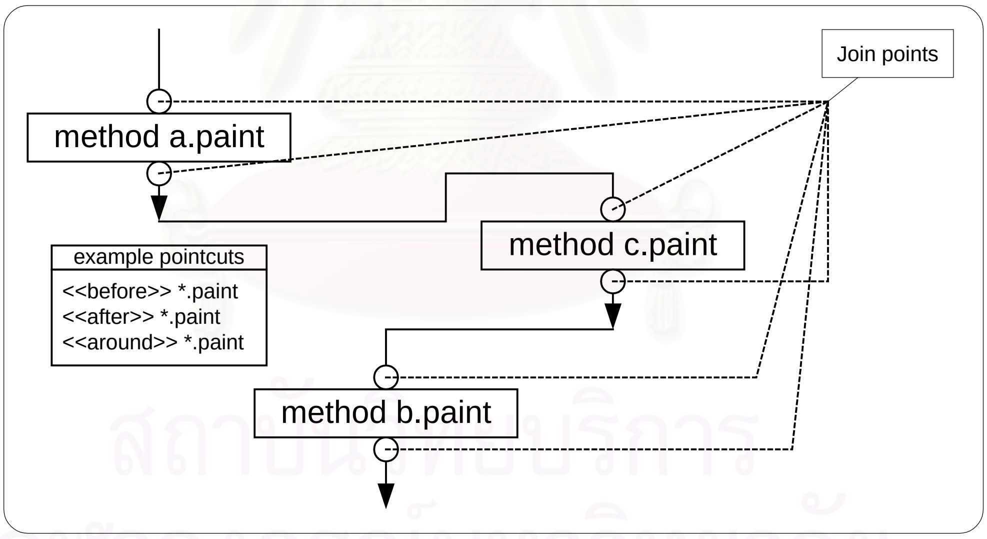


Figure 2.6 Six join points are defined here, with three sample pointcuts

Figure 2.6 shows a concept of join points and their possible pointcuts. Each rectangle block represents a method. Circle shapes placed before and after those blocks indicate possible join points in the system. Figure 2.7 shows a code fragment written in AspectJ. In Figure 2.7, there is the aspect *canvasUpdating* that contains a pointcut designator *afterPainting*. This pointcut cuts across three classes, the *figure*, the *figureA*, and the *figureB*. This makes after calling of the *paint* methods of those three classes; their associating canvases will be also called the *update* method.

```
aspect canvasUpdating {
  pointcut afterPainting(figure f):
    target(f) && call(public void *.paint());
  after: afterPainting(figure f) {
    f.getCanvas.update();
  }
}

class figure {
  public Canvas getCanvas();
  public void paint() {
      . . .
  }
}

class figureA extends figure{
  public void paint() {
      . . .
  }
}

class figureB extends figure{
  public void paint() {
      . . .
  }
}
```

Figure 2.7 A basic code listing of the figure-painting program written in AspectJ

2.2.2 UML Use-Case Package

The Unified Modeling Language (UML) [6] is a set of notations for designing software system using the object-oriented approach. UML is the industrial standard and managed by the Object Management Group (OMG – see http://www.omg.org). UML consists of several groups of notations that could be used for describing software artifacts throughout the development cycle. Its *metamodel* separates UML itself into many packages depending on their different tasks. For the requirements phase, UML has the use-case package that contains a number of notations for using in the diagram called the use-case diagram. Use-case diagrams show the relationships of actors and use cases. This package helps capturing functional requirements of the system as use cases. The notations are as follows.

2.2.2.1 Use Case

A use case is a kind of classifier representing a functionality provided by the system. A use case usually captures a functional requirement. It is shown as an ellipse attaching with the name of the use case. An optional stereotype can be placed over the name. Behavioral semantics of a use case can be described in different ways. But it is normally in plain text format.

2.2.2.2 Actor

An actor defines a set of roles representing users of the system. It can interact with the system entities that usually are use cases. The default iconic representation of an actor is a “stick man” figure. Its name can be attached below the figure. An actor may also be displayed as a class notation with the stereotype keyword <<actor>> above its name.

2.2.2.3 Relationships

There are several standard relationships among use cases, and actors. This section discusses three kinds of relationship.

2.2.2.3.1 Association

An association indicates the participation of an actor in a use case. It is only relationship between actors and use cases.

2.2.2.3.2 Extend

An extend relationship from the use case U1 to the use case U2 indicates that the behavior of the use case U2 may be extendible by the behavior specified in the use case U1.

2.2.2.3.3 Include

An include relationship from the use case U1 to the use case U2 indicates that the behavior of the use case U1 will also contain the behavior specified in the use case U2.

The example of a use-case diagram is illustrated in Figure 2.8. There are five use cases, one actor. The relationships *include*, and *extend* are displayed as dashed lines.

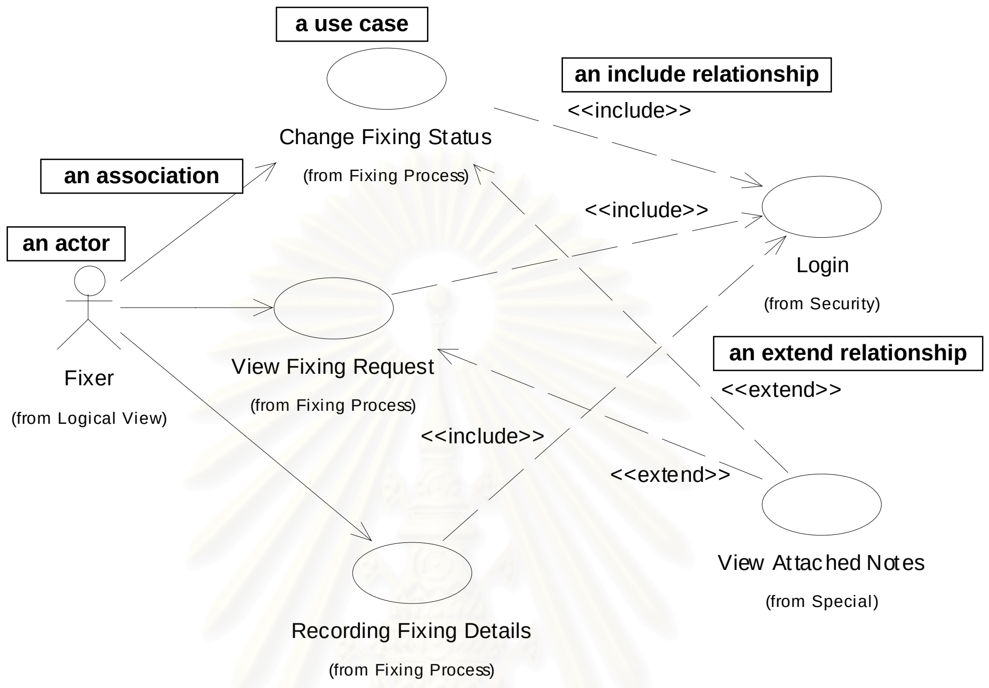


Figure 2.8 An example of the use-case diagram

2.2.3 UML Profile

There are several problem domains that are not directly supported by the original UML. Some special notations, keyword, or attributes may be needed to identify modeling artifacts in such domains. Fortunately, UML has its own standard mechanism called the UML profile [6] to extend its functionalities. Utilizing techniques offered by UML profile make several useful extensions of UML such as UML-RT, UML profile for CORBA [15], and WAE [16]. UML-RT is a profile for modeling real-time intensive applications. The profile introduces several the notations such as capsules, ports, connectors, protocols, and protocol role to UML. It is based on the modeling technique from [17]. UML profile for CORBA [15] provides a standard means for modeling CORBA IDL using UML notations. Conallen's WAE [16] for Web application modeling is another UML profile to model Web pages and HTML elements using UML artifacts.

UML profile standard offers three techniques to extend functionalities of UML as follows.

2.2.3.1 Stereotypes

Stereotypes are classification of an existing UML element. A stereotype keyword is usually labeled to the notation in <<keyword>> form.

2.2.3.2 Constraints

Constraints are restriction placed for controlling their associated stereotype.

2.2.3.3 Tagged values

Tagged values are the stereotype propertied. They contain additional information for the stereotype.

2.2.4 Use Case Maps

Use Case Maps (UCM) [18, 19] visually represent scenarios combined with structures. UCM illustrates use cases in a map-like diagram. Its notation is based on several concepts. A UCM diagram describes *causal relationships* between *responsibilities* that are bound to underlying structures of *components*. A causal path refers the execution path of the use case described using UCM. The paths are said to be causal because it involve ordering of activities that cause to effects. Responsibilities are generic actions, tasks to perform. Components represent generic software entities. Figure 2.9 describes the structural concept of UCM. The details of their elements are described in Table 2.4.

The UCM shares several common characteristics with the activity diagram of the UML [20], but UCMs have many features over the activity graph. UCMs combine structural view of the system with its behavioral activities. This feature advantages for describing architectural view of the system, while the UML activity diagrams emphasize on message sending between objects. This makes UCMs fit the need for modeling scenarios of use cases.

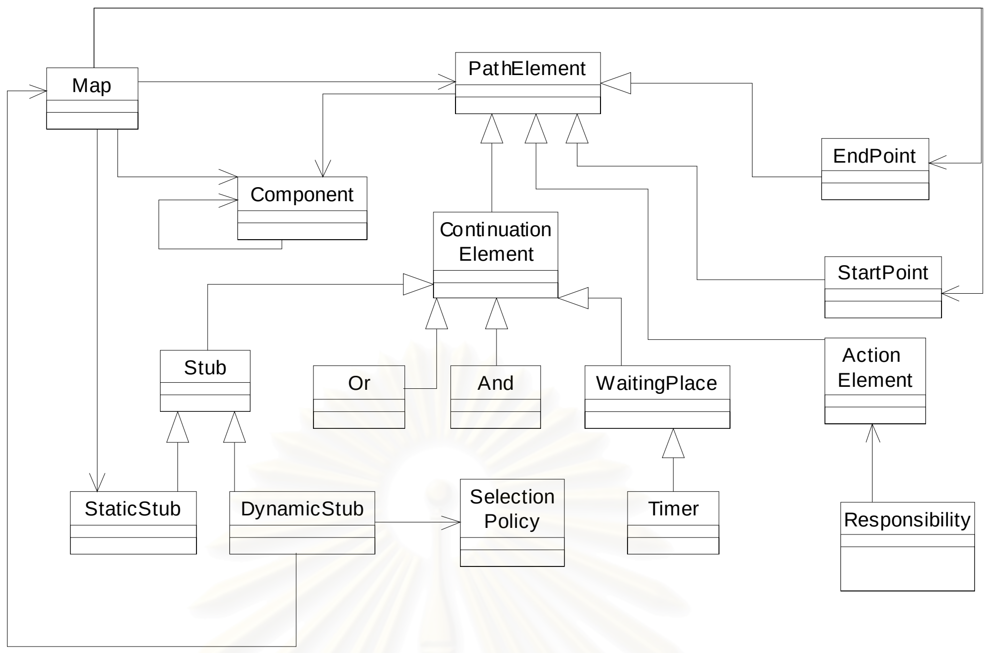


Figure 2.9 Class diagram describing the core concept of UCM

Table 2.4 Description of UCM concepts

Class Name	Description
Map	Composition of path elements and components
Path Element	Abstract class for an element over a causal path
Start Point	Beginning point of a scenario (possibly with preconditions)
End Point	End point of a scenario (possibly with postconditions)
Action Element	A path element on a causal path
Responsibility	An element to perform an action
Continuation Element	A super class representing a location where multiple path elements can connect together in a non-sequential way
OR	Composition of path as alternatives
AND	Composition of path as concurrent
Stub	A super class that represents a container of sub maps
Static Stub	A stub with a single sub maps with its relationship

2.2.5 Rational Unified Process

The Rational Unified Process (RUP) [3], and the unified process [1] are software engineering processes that employ the use-case driven approach and uses the UML as its modeling notation. The RUP is the enhanced and commercialized version of the unified process. Figure 2.10 shows the development lifecycle of the RUP, including its disciplines.

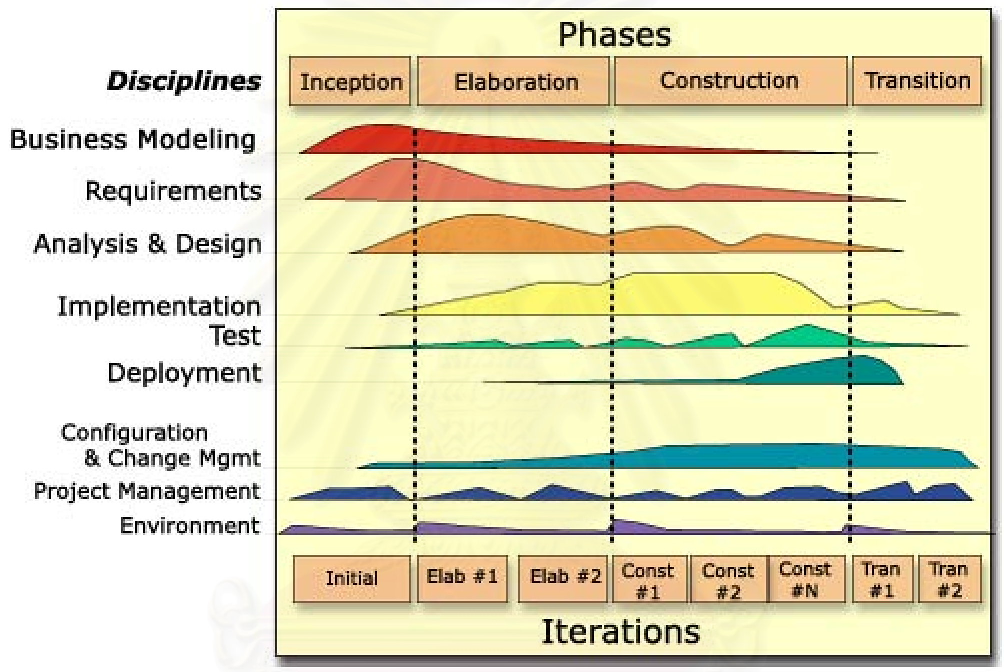


Figure 2.10 Development iterations defined in RUP [3]

Software development life cycle of RUP is the iterative controlled model [3]. Vertical dimension of the RUP process indicates the development disciplines while the horizontal dimension shows the development phases. This section reviews the requirements disciplines of RUP mainly located at the inception phase, and can be also found at the beginning of the elaboration phase. Figure 2.11 shows the overview of requirements discipline of RUP.

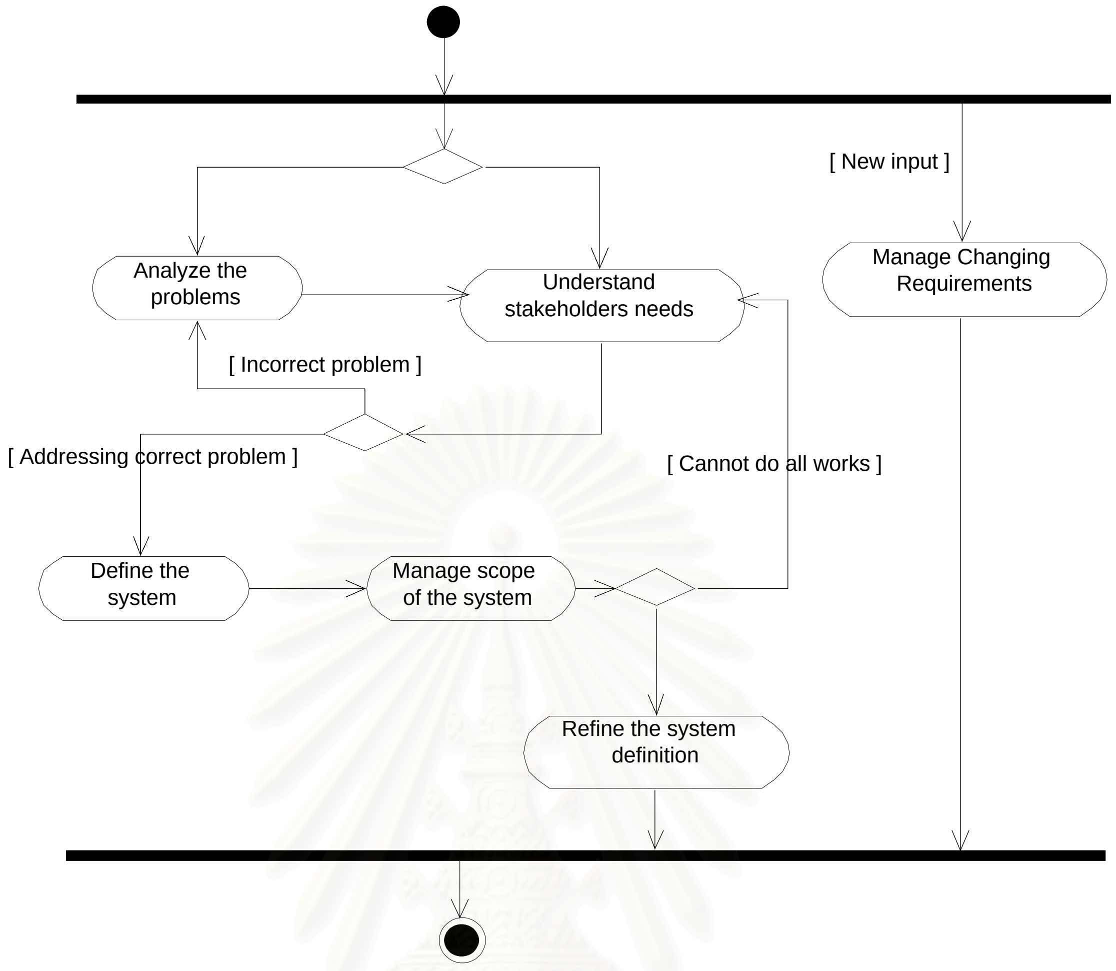

[New input]
Manage Changing Requirements
Analyze the problems
Understand stakeholders needs
[Incorrect problem]
[Addressing correct problem]
[Cannot do all works]
Define the system
Manage scope of the system
Refine the system definition

Figure 2.11 The RUP requirements workflows [3]

The requirements discipline contains six workflows. The summarized activities in each workflow are as follows.

2.2.5.1 Analyze the problems: the actors and preliminary use cases are identified in this workflow.

2.2.5.2 Understand stakeholders' needs: the actors and use cases are refined.

2.2.5.3 Define the system: the actors and use cases are refined.

2.2.5.4 Manage scope of the system: use cases are prioritized and organized their dependency.

2.2.5.5 Refine the system definition: this step is to specify the use cases details.

2.2.5.6 Manage changing requirements: the use-case model is restructured and managed their dependency.

The important artifacts using in this discipline are the software requirements specification (SRS) documents, the use-case model, the vision document, and etc. RUP also provides complete document templates for specifying the system details that are gathered from the stakeholders. There are two versions of the SRS document, the traditional, and the SRS for use-case driven approach.

The next chapter will describe our approach. It includes the new requirements model, the notations. Their mathematical perspective will also be presented.

CHAPTER 3
CONCEPTUALIZATION

3.1 Aspect-Oriented Requirements Model

The requirements model proposed in this thesis consists of a group of software processes that are designed to support the AO paradigm. These processes cover the early stages and the beginning of the analysis phase of the software development. To support the AOSD, this model has to be accomplished due to the reasons as follows.

- It is necessary to manage crosscutting concerns at the requirements level of the process.
- It is necessary to identify crosscutting concerns as aspects during the process of use cases capturing.
- It is necessary to realize aspects with scenario descriptive model during the process of use case realization to help capturing aspects as analysis and design artifacts.

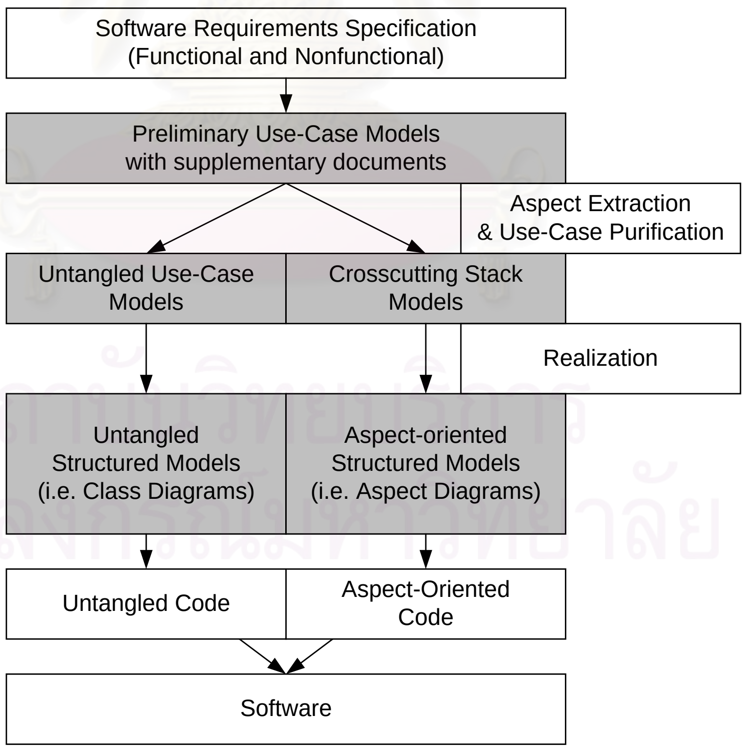


Figure 3.1 Overview workflow of the model

Figure 3.1 shows the overview of the model. Those processes are intended to support the unified process [1, 3]. This makes the unified process possible to use with AO paradigm. Table 3.1 shows the summary of input, and output artifacts, including roles that are related to this requirements model.

Table 3.1 The summarized artifacts and roles for this approach

Process	Input	Output	Role
Use-case purification and aspect extraction	1. Preliminary use-case model with supplementary documents	1. Purified use-case models 2. CSM diagrams	Aspect specifier
Aspect realization (parallel with use-case explanation process)	1. CSM diagrams 2. Purified use-case models	1. OCUM diagrams	Aspect engineer (with Use-case engineer)

3.1.1 Use-case purification and aspect extraction

This process gets the preliminary use-case model, and the supplementary documents as its input. It is to identify and specify crosscutting concerns and capture them as aspects. Extracted aspects will be put into the model called the crosscutting stack model (CSM). After capturing aspects, use cases that are identified to be a part of those aspects will be removed out of the use-case model. This activity is called the use-case purification. The purified use-case model should contain only the core services of the system. The overview of these processes is in Figure 3.2.

There are two sub processes as stated in Figure 3.2, the aspect extraction, and the use-case purification process. The main role involving with these processes is the aspect specifier. An aspect specifier is responsible to identify, specify crosscutting requirements as aspects.

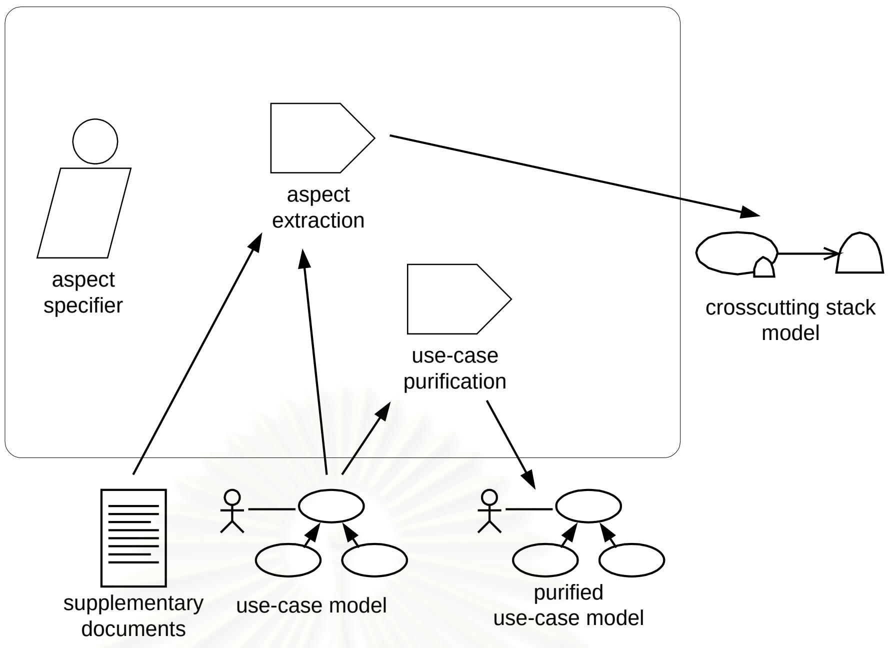


Figure 3.2 Use-case purification and aspect extraction activities

3.1.1.1 Aspect Extraction

This section describes activities, which are responsibilities of the aspect specifier, for identifying, specifying, and finally extracting the aspect from the use-case model supporting with the supplementary documents. A candidate aspect consists of, at least, a use-case selector, a pointcut association, and an advice case. The process starts firstly with identification of advice cases from the use-case models.

Preliminary Identification guidelines for aspects are as follows:

- Consider all use cases that do not associated with actors as an advice case.
- Consider the secondary requirements as an advice case.

Then, specify the details for the candidate crosscutting artifacts using the template. Inspired by the works [9, 10, 8], the template for specifying aspects in the use-case driven approach is shown in Table 3.2.

Figure 3.3 shows the activities for extracting aspects from the preliminary use cases. Several notations, advice cases, use-case selectors, and pointcut associations, from the crosscutting stack model are introduced here. The details of those notations will be discussed later in the next chapter.

Table 3.2 The template for use-case driven aspects

Name	<The name of the candidate advice case>
Concern	<Kind of concern (i.e. security, distributed, etc.)>
Description	<Executive description>
Source	<Source of information (i.e.use cases, stakeholders, documents)>
Crosscutting Type	<Functional or Nonfunctional>
Priority	<MAX, MED, MIN>
Precedence	<Precedence value for resolving conflictions (0…1500)>
Obligation	<Optional or Mandatory>
Influence	<Activities of software process affected by the aspect>
Models	<Related models (use-case models, scenario models)>
Requirements	<Related requirements>
Points	<Location in the scenario this aspect should be found>

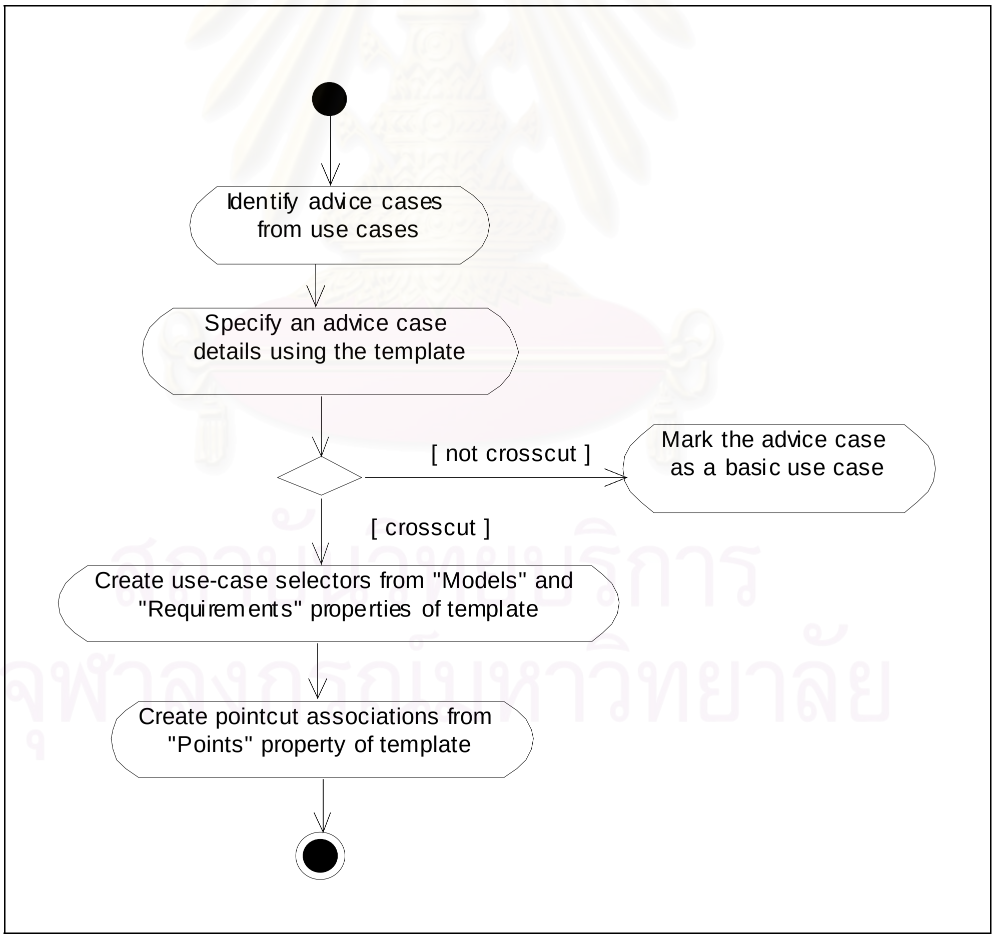

Identify advice cases from use cases
Specify an advice case details using the template
[not crosscut]
Mark the advice case as a basic use case
[crosscut]
Create use-case selectors from "Models" and "Requirements" properties of template
Create pointcut associations from "Points" property of template

Figure 3.3 Summarized activities for extracting aspects

In [9, 8], the authors suggested that considering candidate aspects using template can be done by looking at the models, and requirements path of the template. If they traverse several models or requirements, then they are candidate aspects.

3.1.1.2 Use-case purification

After identifying, specifying, and extracting aspects, the process to be done is the use-case purification. This process re-arranges the use-cases and removes all notations indicated as aspects out of the use-case model. Those aspects will be moved to the CSM diagram. Figure 3.4 shows the activities for the process of purifying use cases.

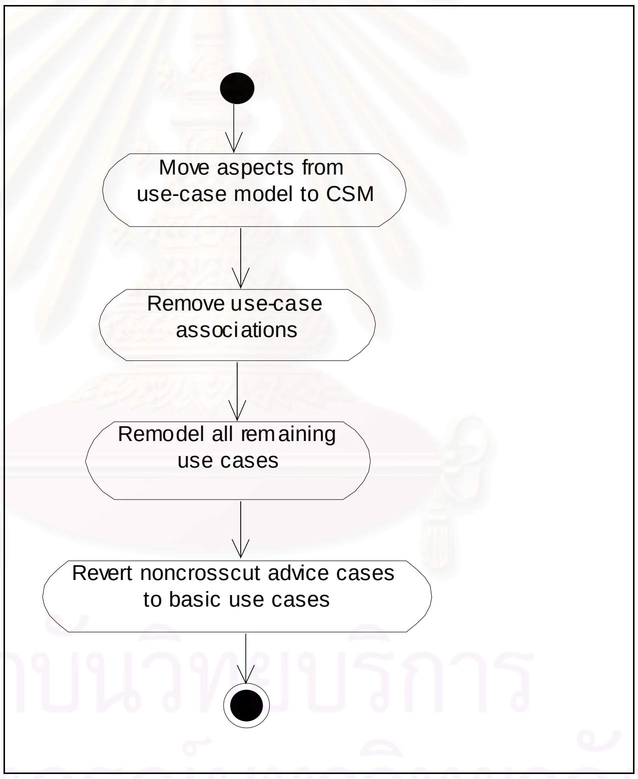

Move aspects from use-case model to CSM
Remove use-case associations
Remodel all remaining use cases
Revert noncrosscut advice cases to basic use cases

Figure 3.4 Use-case purification activities

3.1.2 Realization

There are four sub processes in the aspect realization process. This includes use-case explanation process because this approach employs the OCUM model for describing the scenario of use cases. The details of notations used in the OCUM model will be discussed in the later section.

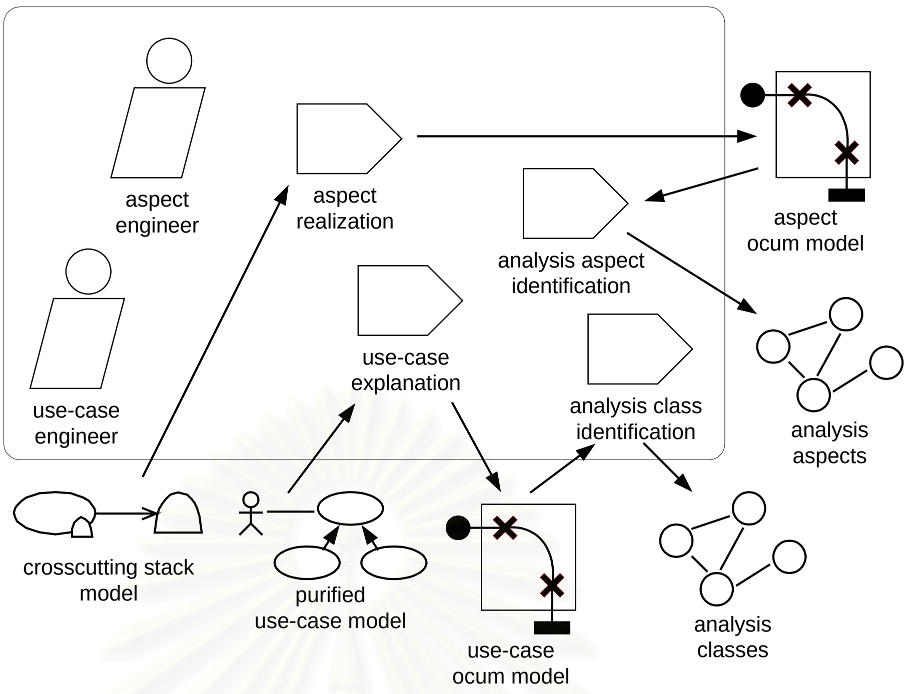


Figure 3.5 Aspect realization and use-case explanation activities

Figure 3.5 shows the overview of the activities for realizing aspects describing use cases. There are two roles involving this process. The aspect engineer is responsible for realizing aspects by describing them into scenario models. Then aspects for the analysis phase, called *analysis aspects*, should be captured from the scenario models. In the parallel activities, use cases are described and then used as input artifacts for capturing analysis classes from their scenarios. These parts are identical to the activities in the Unified Process [1, 3], but this approach employs OCUM, which is derived from the UCM [18, 19], as scenario diagrams. The use of OCUM provides several advantages beyond the scenario diagram of UML because an OCUM diagram models structural and behavioral artifacts in the same diagram. Moreover, OCUM is intended to support AO scenario description. The four sub processes are discusses in details here.

3.1.2.1 Aspect Realization

The aspect realization process is to describe an aspect from the CSM form into the OCUM scenario. This process provides mapping guideline to convert use-case selectors from CSM to the start-point providers with the dynamic start-points.

These notation specifications are described in details in the later chapters. The mapping guidelines are summarized in Table 3.3.

The advice cases will be described into the sequence of responsibilities attaching with Object Constraint Language (OCL) [6] expressions for specifying their behavior. These responsibilities are linked together with the causal path beginning with the start point. Stub stacks may be put along the causal path of other use cases, which will invoke the advice cases, depending on the information from the pointcut associations.

Table 3.3 The mapping guideline for the aspect realization process

Mapping From / To		Description
CSM notations	OCUM notations	
Use-case selectors	Start-point providers and dynamic start-points	The start-point providers and the dynamic start-points might be think of that they are specified version of the use-case selectors
Pointcut associations	Stub stacks	Placing the stub stacks along the path of other use cases is depended on the information “where” from the pointcut associations
Advice cases	Components, the causal path, responsibilities, OCL expression	Advice cases usually are explained as scenarios. Their behaviors are described by OCL expressions, and components.

3.1.2.2 Analysis Aspect Identification

This process identifies an aspect from the OCUM scenario model to the aspect notation for the analysis phase. During progression of this work, there is current no the standard notation for describing aspects in the analysis, and design phase. Recent work [21] proposed the design notation, called *aspect*, for the aspect-oriented design model (AODM). But it is only for AspectJ [7] language mapping, not generalized [21]. This process covers only the guideline for mapping the scenario artifacts found in OCUM to the general analysis aspect template. It does not follow the semantic of the AODM. However, mapping again from the template to the AODM artifact is not difficult to be done. Table 3.4 shows the template for specifying the analysis template captured from the scenario. The template consists of three rows as follows; the name of analysis aspect, list of pointcuts, and the advice code.

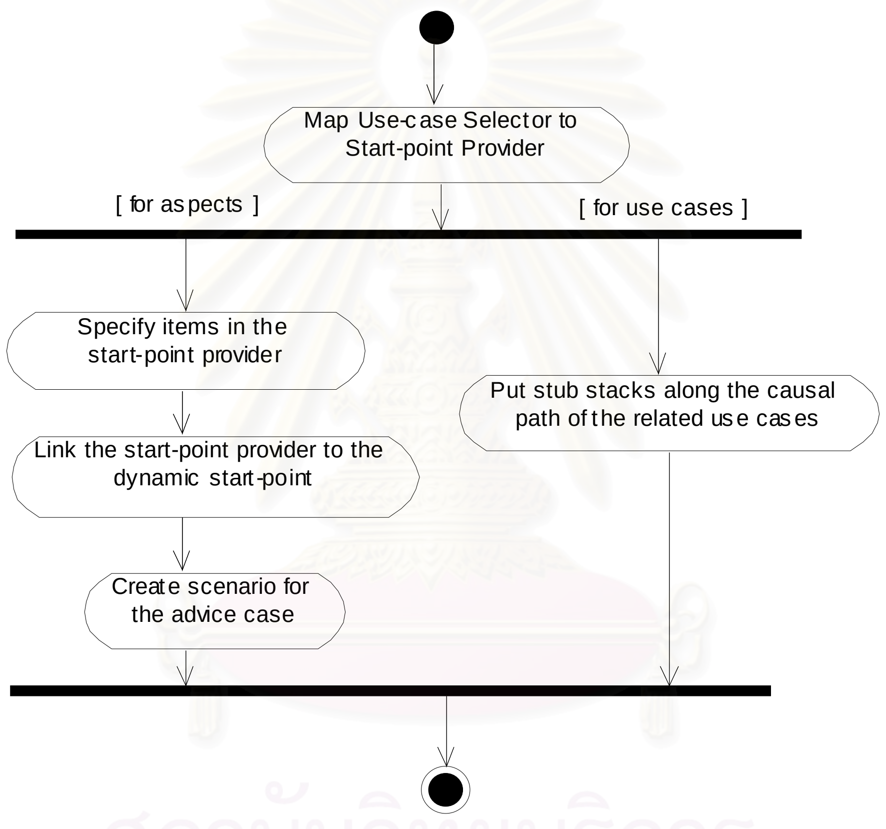

Map Use-case Selector to Start-point Provider
[for aspects]
[for use cases]
Specify items in the start-point provider
Link the start-point provider to the dynamic start-point
Create scenario for the advice case
Put stub stacks along the causal path of the related use cases

Figure 3.6 Activities for realizing aspects

Table 3.4 The simple aspect template for specifying analysis aspects

Name	<Name of the analysis aspect>
Pointcuts declaration	<List of pointcuts in regular language>
Advice code	<Pseudo code of the advice code>

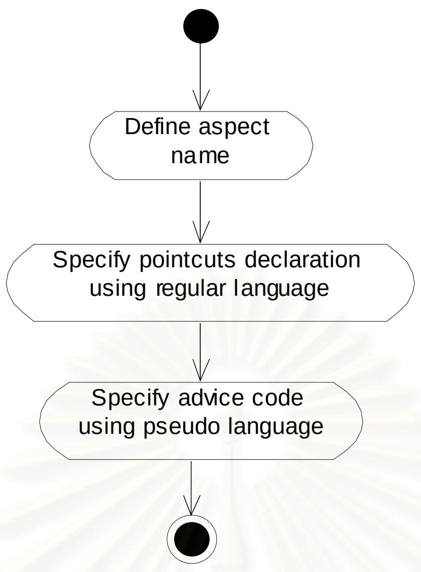

Define aspect name
Specify pointcuts declaration using regular language
Specify advice code using pseudo language

Figure 3.7 The activities for identify an analysis aspect

Figure 3.7 shows the activities to map the aspect from OCUM model to the template. Firstly, the name of the aspect is defined. It is usually from the name of the advice case. The second activity is to declare the pointcuts using regular language. The information for declaring pointcuts should be gained from the location of stub stacks from the scenario of the use cases that are related to the advice case, and the information from the start-point providers of the advice case. Finally, the scenario of the advice case is specified in pseudo, or formal language for describing as advice code.

3.1.2.3 Use-case Explanation

This process describes use cases from the use-case model into the OCUM scenario model. The use-case scenario can be specified as a sequence of responsibilities along the causal path of the OCUM model. Stub stacks may be placed on several points of the path to indicate that these points are crosscut, and it will invoke the relating advice cases.

In the UML, this kind of scenario can be described with the activity graph. But the OCUM model advantages over the activity diagram. The features derived from the UCM make the scenario, described using OCUM notations, modeling both structural artifacts and behavioral activities in the single view. This makes the OCUM model better describing the early architecture of the system than the activity graph [20].

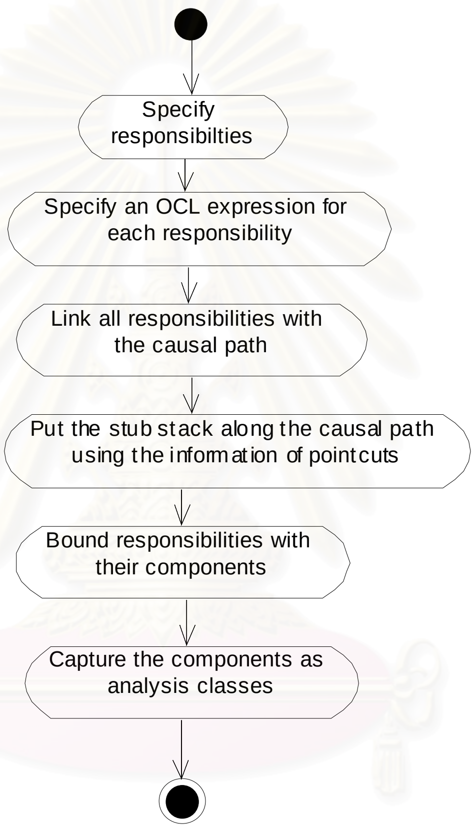

Specify responsibilties
Specify an OCL expression for each responsibility
Link all responsibilities with the causal path
Put the stub stack along the causal path using the information of pointcuts
Bound responsibilities with their components
Capture the components as analysis classes

Figure 3.8 The activities for describing use cases, and capturing classes

3.1.2.4 Analysis Class Identification

This process is to capture the analysis UML classes from the OCUM components in use-case scenarios. The activity is simple because the OCUM defines the concept of its components corresponding to the class concept of the UML. Figure 3.8 shows the activities of both the use-case explanation, and the analysis class identification in the same workflow.

3.2 Crosscutting Stack Model

This section describes the crosscutting stack model (CSM), and its notations. This model is a combination of the AO paradigm with the use-case model. The CSM is extended from the preliminary work proposed in [22].

Several defining extensions are based on the use-case package of the metamodel of the UML [6]. The use-case package is a sub package of the behavioral package of the metamodel. The key elements of use-case model are use cases and actors. To extend its functionality for capturing crosscutting requirements with the concept of aspect-oriented, a crosscutting stack model, an advice case, a use-case selector, and a pointcut association, are introduced here.

3.2.1 A Diagram of CSM

3.2.1.1 Semantics

The diagram of the CSM shows use-case selectors and advice cases together with their relationships. The advice cases represent system functionality or properties that cut across other functionalities, which are use cases, of the system.

3.2.1.2 Notations

A diagram of the CSM is a graph of use-case selectors and a set of advice cases, and the relationships between these elements. The relationships are special kind of associations called the pointcut associations. The example of the CSM diagram is illustrated in Figure 3.9.

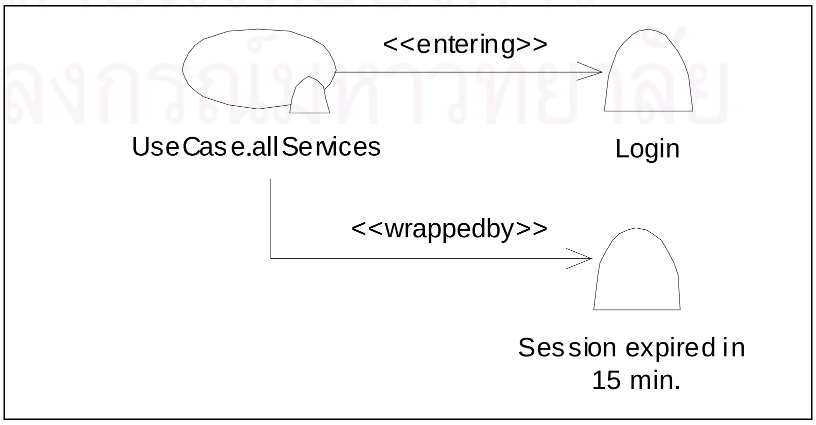


Figure 3.9 The Example of Crosscutting Stack Model

3.2.2 Advice Case

3.2.2.1 Semantics

An advice case is defined as a specialization of the classifier from the UML metaclass. It represents a functionality or property of the system that cut across other use cases in the use-case model. It also defines a sequence of actions, but cannot be performed directly by the actor. Triggering from an instance of use-case selector will perform an advice case. A concept of an advice case follows the concept of the *advice* in the AOP [5].

3.2.2.2 Notations

An advice case is shown in the crosscutting stack diagram using a notation of use case with attaching stereotype keyword <<advice case>>. It can also be modeled using a vertical-half-ellipse, the A-like shape. It contains the name of advice case below the icon. Graphical representations of an advice case are displayed in Figure 3.10.

3.2.2.3 Presentation Options

The name of the advice case may be placed below its icon. The name of an abstract advice case may be shown in italics.

3.2.2.4 Style Guidelines

Advice case names should follow style guidelines stated in the UML specification [6].

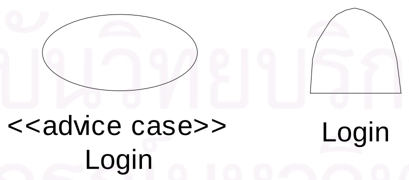


Figure 3.10 Graphical representations of an advice case

3.2.3. Use-case Selector

3.2.3.1 Semantics

A use-case selector is a kind of classifier representing a group of functionality, or use cases of the system. Use-case selectors follow the concept of pointcut designators in AOP [5]. According to AOP, a pointcut is a set of selected join points of the system [5]. A pointcut defines *what* will be crosscut, and when. This similar semantic is defined using a use-case selector incorporating with a pointcut association. This approach uses a use-case selector to define what the advice case crosscuts.

3.2.3.2 Notations

A use-case selector is displayed as a use-case notation attaching with <<use-cases selector>> stereotype in the CSM. It is also represented as a use-case with a little vertical-half-ellipse attaching at the right corner of it. Figure 3.11 shows both stereotype style, and iconic representations of a use-cases selector.

A use-case selector contains an OCL expression below its icon. A use-case selector uses the expression to find a group of use cases. This notation is to represent what to be crosscut, not where. In this model, the AOP pointcut concept is separated into a use-case selector, and a pointcut association. A pointcut association defines when to cut across. Separating a use-case selector from a pointcut association enables reusing the same selector with many pointcut associations.

3.2.3.3 Presentation Options

The name of the use-case selector may be placed below the icon instead of the OCL expression for describing the group of use cases that will be selected using the natural language.

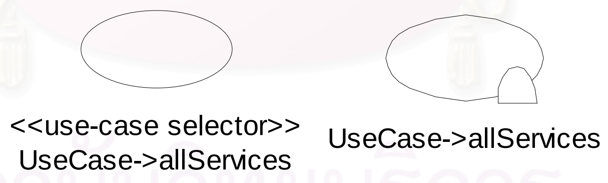


Figure 3.11 The use-case selector, and its iconic representation

3.2.4 Pointcut Association

3.2.4.1 Semantics

A pointcut is a kind of association that links between use-case selectors and advice cases in the CSM diagram. A pointcut association must be labeled with a stereotype to indicate *where* the use cases grouped by the use-case selector should perform the appropriating advice case. Combining pointcut associations with use-case selectors provides the AO concept of pointcut designator in the CSM diagram.

3.2.4.2 Notations

A pointcut association is a relationship attaching with stereotype keyword. Figure 3.12 shows the use of the *entering* pointcut association incorporating with the use-case selector and the advice case *Login*. This is a complete aspect notation for using in the CSM. The pointcut association labeled with <<entering>> forces the system to perform the advice case "Login" *before* performance of all use cases in the current model.

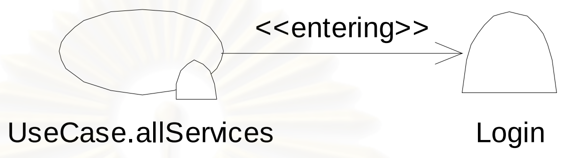


Figure 3.12 An aspect – the combination of a use-case selector, a pointcut association, and an advice case

Predefined set of stereotypes that can be attached to a pointcut association is in Table 3.5. The set of stereotypes defined in [9, 10] are revised and extended here. They are also listed in Table 3.5.

Table 3.5 A set of pre-defined pointcut associations

Pointcut Association Stereotype	Description
Entering	A pointcut before actor performing the use case (overlapping activities at the beginning of the use case).
Leaving	A pointcut after actor performing the use case (overlapping activities at the end of the use case).
Exception Raising	A pointcut when actor performing the use case with error handling.
Wrapped By	Entering + Leaving.
Overlapping	Partially replacing activities of the use case.
Overriding	Replacing all activities of the use case.

3.2.5 Additional OCL Properties

In order to make a use-case selector having semantics in itself, we define additional OCL [6] properties to use with the use-case selector. Two additional properties are defined as the following: *UseCase.allServices* is the OCL property for UseCase type returns its result as a set of all use cases that are performed by every actor, and *UseCase.servicesOfActor* is the OCL property that returns its result as a set of use cases specified by Actor. The implementation of both properties are as follows.

```
UseCase.servicesOfActor(a: Actor): Set(UseCase)
Context
  UseCase::servicesOfActor(A: Actor): Set(UseCase)
pre:
  true
post:
  result = a.allConnections->select(r |r.type.OclIsKindOf(UseCase))
```

```
UseCase.allServices(): Set(UseCase)
Context
  UseCase::allServices(): Set(UseCase)
pre:
  true
post:
  Actor.allInstances->forAll( a |
    result.union(UseCase.servicesOfActor(a)))
```

3.2.6 Use Case to Advice Case Converting Rules

Rules defined here are to convert use cases from the traditional use-case model to advice cases in CSM. Mathematical analysis of these rules will be further discussed later in this chapter. A number of definitions will also be introduced to support the following rules. Two rules are described as follows.

3.2.6.1 Every tangled use case (discarded use case) that is removed out of the use-case diagram must be replaced by an effective advice case instead in the CSM diagram.

3.2.6.2 All tangled associations linked with the discarded use case must be removed out of the use-case diagram, and the effective advice case must have one equivalence pointcut association to those association in the CSM diagram.

3.2.7 Tool support

The described notations above are all supported by the ASREM add-in. ASREM is an add-in of Rational Rose, the UML modeling tool.

Figure 3.13 shows ASREM in action. The technical details of the implementation of ASREM, including its user's guideline are discussed later in the appendices. ASREM consists of two parts as follows:

3.2.7.1 Notations and Association

ASREM offers a set of notations supporting the CSM diagram in Rational Rose. The notations include the use-case selector, the advice case. Their base notation is the use case attaching with their stereotypes. ASREM offers modeling of the pointcut association based on the association relationship of Rational Rose.

3.2.7.2 Stereotypes and Tagged Values

All pre-defined pointcut associations that can be found in Table 3.5 are implemented as a set of association stereotypes in ASREM. The properties of the template of the use-case driven aspect that are stated in Table 3.2 is also implemented as tagged values for the advice case.

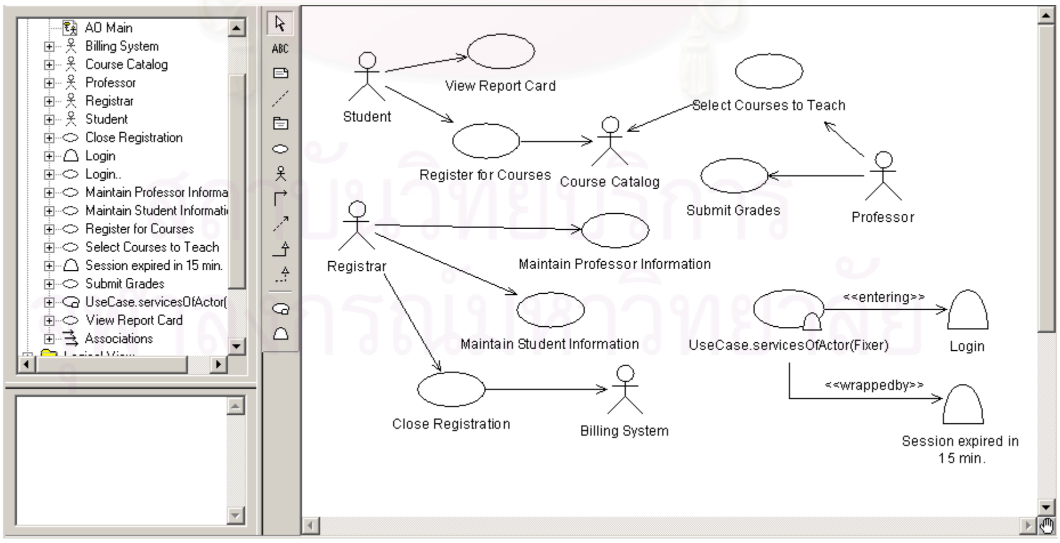


Figure 3.13 Modeling CSM notations in the use-case diagram of Rational Rose

3.3 Object/Crosscutting/Use Case Maps

The UCM [18, 19] is a semi-formal, and map-like diagram for describing use-case scenarios. A UCM diagram features an architectural explanation of the system. It visually integrates structural components and scenarios of use cases in single view [20]. This is very useful for modeling the interactive, such as applications in Web-based domain [23], systems.

UCM basic notations consist of the following elements: a component, a causal path, a stub, and a responsibility [18, 19]. Several UCM notations are improved to support semantics of aspect-oriented, and also more object-oriented here. Moreover, the integration of OCL [6], a formal language proposed as a part of the UML, with the UCM is also suggested. The detail of the improvements, which are implemented in The Object/Crosscutting/Use Case Maps (OCUM), is described.

The enhancements of OCUM beyond the UCM are divided into two groups for supporting both AO and OO paradigm. The enhancements to support AO are stub stacks, start-point providers, and dynamic start-points. The enhancements to better support OO concept are embedded OCL expressions, object context abstraction, parameter symbols, object constructors, destructors, and type information. Table 3.6 summarizes these improvements. Figure 3.14 shows the core structure of the OCUM. The core structure is illustrated as a UML class diagram. Shaded elements are the enhanced notations.

3.3.1 Stub Stack

3.3.1.1 Semantics

In UCMs, the stubs can be replaced by sub-maps. This concept is considered partial supporting AO's joint points. Unfortunately, one stub can be replaces by one plug-in at a time, although the stub is the dynamic kind. But in AO, a joint point links many advice codes depending on its pointcuts designator. When the join point is reached, all advice codes are performed. Both traditional stubs concept, which are static and dynamic, are not adequate to support the kind of AO semantic. To describe the scenario that supports such semantic, the stub stack is introduced here. A stub stack is to support AO concept in OCUM diagrams. It contains several static stubs that link to their sub maps.

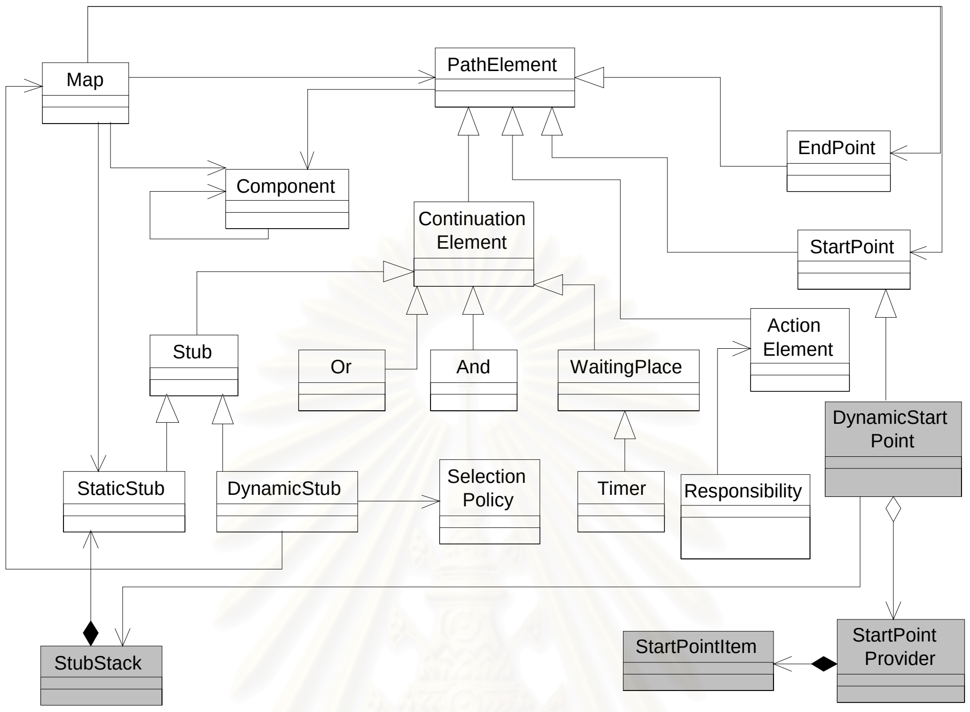


Figure 3.14. The OCUM core concept

3.3.1.2 Notations

A stub stack is illustrated by stacking diamonds. The name is optionally attached below the icon. Figure 3.15 shows the graphical representation of a stub stack. This notation is designed as the stacking diamonds because a single diamond represents a stub in UCM, and this notation is to represent all possible ways to invoke their related advice cases, thus it is designed to be a stack of UCM stubs. A stub in UCM can be thought of that it is an advice case in AO concept. This notation represents that this point will invoke multiple advice cases at a time.

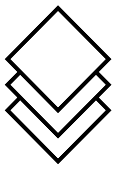


Figure 3.15 The stub stack notation

Table 3.6 Summarized enhancement of OCUM model

Paradigm Support	Feature Name	Type	Description
Aspect-oriented	Stub stacks	Notation	To support the join point semantic on the causal path
	Start-point providers	Notation	To support the mapping of use-case selector form CSM model
	Dynamic start-point	Notation	To enhance the concept of static start point. This makes this dynamic start-point possible to get information from the start-point provider
Object-oriented	Embedded OCL expression	Text	This enables responsibilities contain an action semantic
	Object context abstraction	Abstraction	This concept helps make the OCL expression contains the meaning relating to the component.
	Parameter symbols	Text	This is to support the argument concept and the dataflow analysis concept though the causal path.
	Object constructor, destructor	Text	This is to support constructor/destructor concept for optionally indicating the object state.
	Type information	Text	This adds stereotype concept of UML to the component block. It makes possible to provide more information of the component.

3.3.2 Dynamic Start-point

3.3.2.1 Semantics

A dynamic start-point retrieves information from the start-point provider to start the scenario. A dynamic start-point is a derivative of a start point. This new notation is to support the AO paradigm.

3.3.2.2 Notations

Dynamic start-points are indicated by dashed circle attaching its name below the icon. Figure 3.16 shows the dynamic start-point with the name 'login'.

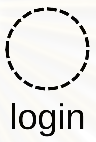


Figure 3.16 The graphical notation of the dynamic start-point

3.3.3 Start-point provider

3.3.3.1 Semantics

A start-point provider contains a number of points indicating where its associated start point can be used. The concept of the start-point provider is designed to support AO scenario description. In AO, the advice code can be performed at several join points depending on its pointcut designators [5]. This means the advice code may has more than one start point. The traditional UCM notation did not explicitly support this concept [20, 18, 19]. To realize this concept in the OCUM scenario, the start-point provider is introduced to serve the starting information to it start point. The start-point provider contains a number of start-point items. Each item is labeled with the name of use case that the scenario will start from. In fact, the start-point provider is the specified version of the use-case selector from CSM.

3.3.3.2 Notation

The graphical representation of the start-point provider is a stack of rectangles. Each rectangle is a start-point item. Figure 3.17 shows connecting a start-point provider to a login dynamic start-point. The start-point provider contains three start-point items. This enables substitution of those start-point items to the dynamic start-point depending on its selection policy. Each item may contain an expression for indicating where the start-point relates. The related stub stack name is expressed after the @ sign. The start-point item “Change Fixing Status@cfs01” means it is a start-point invoking from the stub stack named *cfs01*.

3.3.3.3 Guideline

The start-point provider must link to the dynamic start-point. Each item in the start-point provider contains its related use-case name. The use case name optionally follows by the @ sign and the stub stack’s name to express its location on the scenario path.

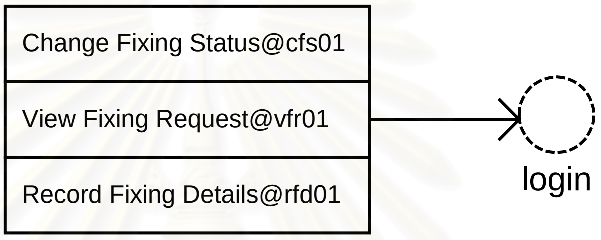

Change Fixing Status@cfs01
View Fixing Request@vfr01
Record Fixing Details@rfd01
login

Figure 3.17 The start-point provider with the dynamic start point

3.3.4 Embedded OCL Expression

Embedding OCL expression to a responsibility enables formal specification to the scenario of the use case. This provides additional using of parameter symbols to help better understanding of the dataflow through out the causal path. An OCL can be expressed by specifying it below the responsibility, instead of its name. The *object context* of the expression is provided by the components bounding the responsibility. This object context abstraction is a useful enhancement of the OCUM that helps embedding OCL expression possible. For example, the object *self* specifying in the OCL expression refers to an instance of the component.

3.3.5 Parameter symbol

An operational parameter symbol is a dollar sign with a positive integer i.e. $1, $2, $3. A symbol that holds a result from the last action is indicated by a dollar sign with an underscore ($_), called the result-buffering symbol (RBS). This symbol advantages the dataflow analysis possibilities through the scenario. Both kinds of symbol are proposed to use within an embedded OCL expression.

3.3.6 Object constructor and destructor

To offer the more object-oriented semantics to the diagram, optional object constructor and destructor are proposed as keywords *new* and *destroy* respectively. They can be placed in front of an object context in a component block. The keyword new is indicated that the object will be created, and then it performs OCL expressions. The keyword destroy is indicated that the object perform their OCL expressions, and then it will be destroyed.

3.3.7 Type information

Meta type or other information can be attached to the component block, as a stereotype, above the component name. This concept is as same as the stereotype concept of the UML.

To show the capability that able to support the AO paradigm of the OCUM model, the scenario explanation of the advice case is illustrated in Figure 3.18.

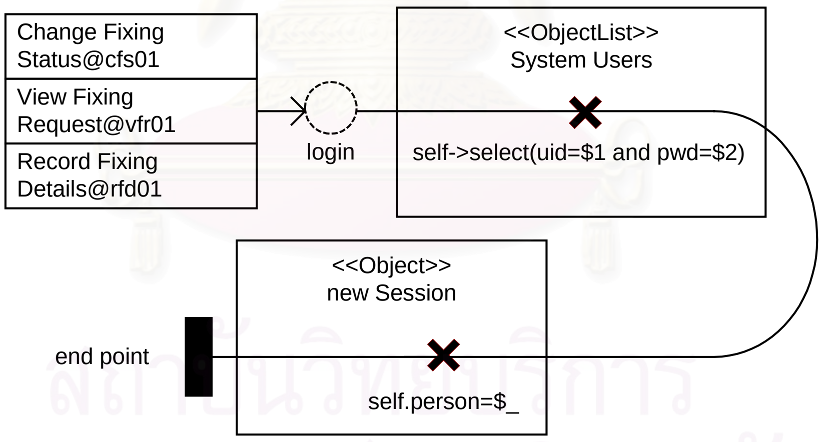

Change Fixing Status@cfs01
View Fixing Request@vfr01
Record Fixing Details@rfd01
login
<<ObjectList>> System Users
self->select(uid=$1 and pwd=$2)
<<Object>> new Session
end point
self.person=$_

Figure 3.18 An OCUM diagram of the Login advice case

From Figure 3.18, the Login advice case is described as an OCUM diagram. It contains a dynamic start-point named *login*. This start point is linked from the start-point providers, which contains a number of start points indicating that the *login* start point will be invoked from them. The component block *System Users* is attached with the stereotype <<ObjectList>>. This makes one can think of that the component should be a list of objects. This information is configurable, and depends on the problem domain. The causal path of the diagram runs through two responsibilities embedding their OCL expressions. The responsibility in the first block contains two parameter symbols. This means that the use case may be implemented as a kind of function in the later phase, and should have two inputs.

The second component contains the responsibility appearing the RBS in its expression. The RBS symbol holds the result from the first OCL expression. Its value will be assigned to the *person* attribute of the object in this component. The concept of including RBS in the diagram makes the dataflow analysis, and consistency checking possible. Type checking can be applied to the diagram to ensure that type of the *person* attribute of the *Session* class conforms to the data retrieved from the first action, which returns *System User* type. This implies some relationships between the *Person* class, and the *System User* class, for example.

3.3.8 Tool support

The concept of OCUM is implemented in the modeling tool called OCUM Vectra. OCUM Vectra includes all above notations for modeling an OCUM scenario. Its summarized features are as follows.

3.3.8.1 It supports OCUM diagram, including OO and AO scenario modeling.

3.3.8.2 It is designed as an add-in of Rational Rose. It can export OCUM components to UML class diagram in Rational Rose.

3.3.8.3 It saves its output as an XML format. This enables other tools to read and process an OCUM diagram.

3.4 Mathematical Perspective

This section discusses the requirements model from the mathematical perspective. A number of definitions of basic elements are introduced. The complexity index is defined. This section ends with a mathematically proof that the crosscutting stack model's complexities are always lesser or equal than the complexities of the preliminary use-case model.

3.4.1 Basic Definitions

Definition 1:

A software system is a tuple of finite services, and finite join points.

Example1:

We define the software system $Z = (\mathbf{S}, \mathbf{J})$,

where **S** is a finite set of system services, and **J** is a finite set of join points.

Definition 2:

Services of the system are either a set of use cases union with a set of advice cases.

Definition 3:

User cases of the system are a finite set of use cases. It is a subset of the services of the system.

Definition 4:

Advice cases of the system are a finite set of advice cases. It is a subset of the services of the system.

Example 2:

Given the system $Z = (\mathbf{S}, \mathbf{J})$,

$\mathbf{S} = \mathbf{U} \cup \mathbf{A},$

$\mathbf{U} = \{u_1, u_2, u_3\},$

$\mathbf{A} = \{a_1, a_2\}$

Definition 5:

A use case of the system is a sequence of activities that are executed through some join points.

Given the system $Z = (\mathbf{S}, \mathbf{J})$,

$\mathbf{S} = \mathbf{U} \cup \mathbf{A}$

$\mathbf{J} = \{j_1, j_2, \ldots, j_n\}$

A use case $U = (Z, \mathbf{U}, \mathbf{j}, \mathbf{\Omega})$ is a use case of the system Z , where

$Z = (\mathbf{S}, \mathbf{J})$ is the software system,

$\mathbf{\Omega}$ is a set of executable path of U over $\mathbf{J}$, and $\mathbf{\Omega} \subseteq \mathbf{J} \times \mathbf{J}$,

$\mathbf{j}$ is a set of single start-point of U, and $\mathbf{j} \subseteq \mathbf{J}, |\mathbf{j}| = 1$.

Definition 6:

An advice case of the system is a sequence of activities. The advice case has one or more start points.

Given the system $Z = (\mathbf{S}, \mathbf{J})$,

$\mathbf{S} = \mathbf{U} \cup \mathbf{A}$

$\mathbf{J} = \{j_1, j_2, \ldots, j_n\}$

An advice case $A = (Z, \mathbf{A}, \mathbf{\Theta}, \mathbf{\Omega})$ is an advice case of the system Z,

where

$Z = (\mathbf{S}, \mathbf{J})$ is the software system,

$\mathbf{\Omega}$ is a set of executing path of A over $\mathbf{J}$, and $\mathbf{\Omega} \subseteq \mathbf{J} \times \mathbf{J}$,

$\mathbf{\Theta}$ is a set of start points of A, and $\mathbf{\Theta} \subseteq \mathbf{J}$

3.4.2 Complexity Index

This section illustrates a proof to show that it always reduces the complexity of the use-case model when applying the technique to the use-case model.

Definition 7:

The complexity index (*CI*) over a use-case model is as follows:

$$CI = \frac{\sum a_i^u + \sum a_i^p - \sum a_i^t}{\sum U_i + \sum A_i^e - \sum U_i^d} \qquad \textbf{(1)}$$

where	a^u	is an association in the use-case diagram,
	a^p	is a pointcut association in the use-case diagram,
	a^t	is a association considered to be tangled,
	U	is a use case,
	A^e	is an effective advice case,
	U^d	is a discarded use case.

In the traditional use-case diagram, no tangled association in the diagram is considered to exist, thus

$$\sum a_i^p - \sum a_i^t = 0 \qquad \textbf{(2)}$$

Now substitute (2) into (1). Thus, we have

$$CI_0 = \frac{\sum a_i^u}{\sum U_i} \qquad \textbf{(3)}$$

We call CI_0 the traditional complexity index of the use-case diagram. Recall (1), we rearrange the equation (1) as follows:

$$CI = \frac{\sum a_i^u}{\sum U_i} + \frac{(\sum a_i^p - \sum a_i^t)\sum U_i - (\sum A_i^e - \sum U_i^d)\sum a_i^u}{(\sum U_i)^2 + (\sum A_i^e - \sum U_i^d)\sum U_i} \qquad \textbf{(4)}$$

Then, we substitute (3) into (4)

$$CI = CI_0 + \frac{(\sum a_i^p - \sum a_i^t)\sum U_i - (\sum A_i^e - \sum U_i^d)\sum a_i^u}{(\sum U_i)^2 + (\sum A_i^e - \sum U_i^d)\sum U_i} \qquad \textbf{(5)}$$

In this thesis, we consider $\sum A_i^e = \sum U_i^d$, by our rule no. 1. Substitution $\sum A_i^e - \sum U_i^d = 0$ into (5), we have

$$CI = CI_0 + \frac{\sum a_i^p - \sum a_i^t}{\sum U_i} \qquad \textbf{(6)}$$

We now simply proof that for all use-case diagram applied our rules into it, we will always have $\frac{\sum a_i^p - \sum a_i^t}{\sum U_i} \leq 0$. Thus, we have $\sum a_i^t \geq \sum a_i^p$ to satisfy our second rule.

The next chapter will illustrate a case study to show how the model is applied to the real problem domain.

CHAPTER 4
IMPLEMENTATION

4.1 Overview

In this chapter, the case study is illustrated to present how the aspect-oriented requirements model can be applied to the real problem. The case study is the maintenance management system for the factory. The complete use-case model of the system and its class diagram are described in the appendix III. In the complete diagram of the system, there are twenty use cases, three actors. Seven associations links between those actors to seven use cases. The thirteen remaining use cases are linked with <<include>>, and <<extend>> relationships to those seven use cases. Case study illustrated here is a part of the system.

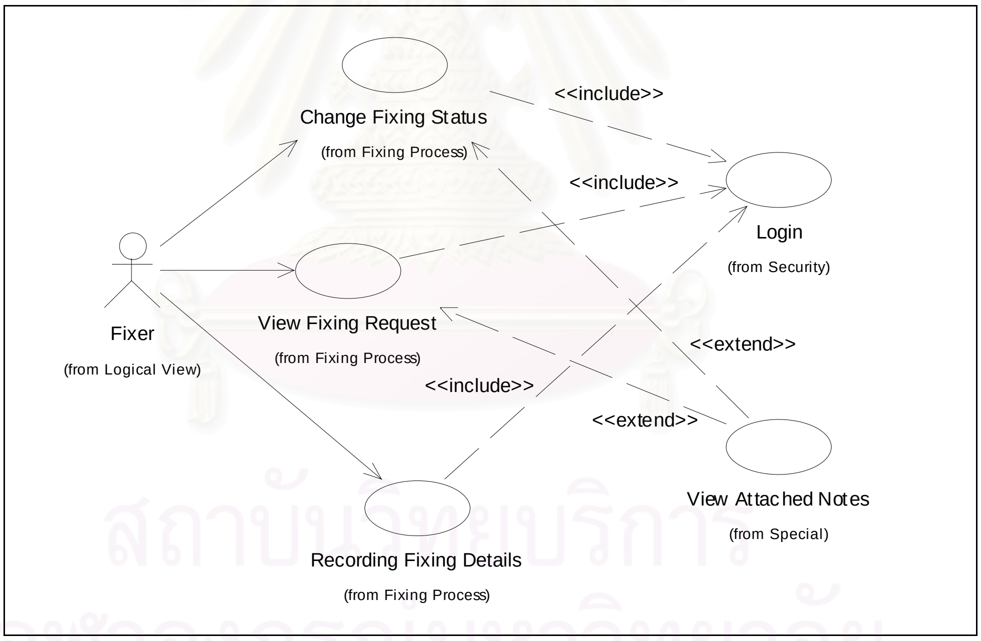


Figure 4.1 The use case diagram of the actor *Fixer*

There are three services served by the maintenance system for the Fixer: the service for changing fixing status of the machine, the service for viewing the fixing request from other employees, and the service for recording the fixing details to the maintenance database. All fixing staffs that are granted to use this system have their own user name and password for logging into the system before performing their tasks. If the logged user does not perform any task for 15 minutes, the user session will be expired. This makes the user to re-login. One can leave notes for any documents displaying in the screen to other users. Figure 4.1 shows use cases diagram for the actor Fixer.

The diagram contains three use cases that are associated with the Fixer. The remaining use cases, Login, and View Attached Notes, are linked with those use cases via <<include>>, and <<extend>> relationships respectively. The Login use cases are included in all three services while the View Attached Notes use case extends two services.

4.2 Modeling Steps

The use case diagram in Figure 4.1 is assumed to be the preliminary use cases diagram. It will be taken as input of the process described in chapter 3. The activities of the process are applied to the case study as follows.

4.2.1 Aspect extraction

4.2.1.1 Identify advice cases

Two advice cases are identified from the use-case model: the Login, and the View Attached Notes use cases. They are identified as advice cases because they links to other use cases via an <<include>>, and an <<extend>> relationships. The Login use case links with three use cases via <<include>>, and the View Attached Notes use case links with two use cases via <<extend>>.

4.2.1.2 Specify advice cases in the template

In this step, the advice cases details are described using the template from Table 3.2. This results the templates shown in Table 4.7 and Table 4.8. Table 4.7 is the template for Login advice case, and Table 4.8 is the template for the View Attached Notes advice case. Figure 4.2 shows the notations of both captured advice cases.

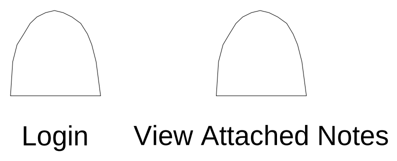


Figure 4.2 The Login and the View Attached Notes advice cases

4.2.1.3 Consider the advice cases are whether crosscut or not

The template properties Models, and Requirements are used for considering the advice case is crosscutting or not. In this case study, the Login, and the View Attached Notes advice cases are both considered crosscutting.

4.2.1.4 Construct a use-case selector from the template information

Use-case selectors can be created from the information of Models, and Requirements properties of the templates. For the Login advice case, the use-case selector will select all three services, Change Fixing Status, View Fixing Request, and Record Fixing Details. The use-case selector for this advice case could be specified as follows.

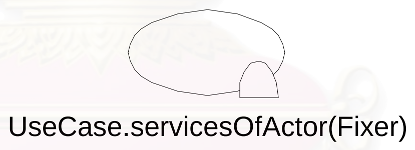


Figure 4.3 The use-case selector selecting all services of Fixer

The advice case View Attached Notes extends two services, Change Fixing Status, and View Fixing Request use cases. Thus, its use-case selector could be specified as follows.

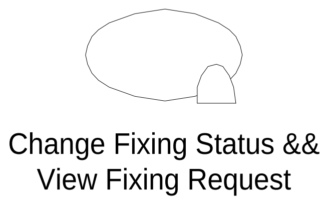


Figure 4.4 The use-case selector selecting two use cases

4.2.1.5 Linking with pointcut association

Referring to the templates of both Login, and View Attached Notes advice cases, pointcut associations <<entering>> could be used to link between these advice cases and their use-case selectors.

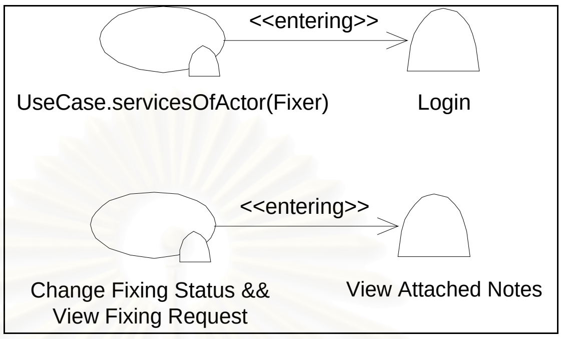


Figure 4.5 The CSM model for *Login*, and *View Attached Notes* aspects

Table 4.7 The specified template for Login

Name	Login
Concern	Security
Description	Restricts the access to the important services of the system
Source	Use cases
Crosscutting Type	Functional
Priority	MAX
Precedence Value	750
Obligation	Mandatory
Influence	Architectural, Design, Implementation
Models	Use cases: 1. Change Fixing Status 2. View Fixing Request 3. Recording Fixing Details
Requirements	Requirements: 1. The user should be logged in before using every service of the system
Points	Before every services

Table 4.8 The specified template for the View Attached Notes advice case

Name	View Attached Notes
Concern	Communication
Description	Enable user to see attached notes if exists
Source	Use cases
Crosscutting Type	Functional
Priority	MED
Precedence Value	750
Obligation	Mandatory
Influence	Implementation
Models	Use cases: 1. Change Fixing Status 2. View Fixing Request
Requirements	Requirements: 1. The user should be able to read the attachment of fixing requests
Points	Before

4.2.2 Use-case purification process

This process is to purify the preliminary use-case model by removing the use cases that are identified as advice cases out of the model. The advice cases are now modeled in the CSM diagram, as in Figure 4.5, instead of the use case diagram. Figure 4.6 shows the diagram after purifying the use-case model.

After purification, there are only three use cases that are the main services of the system, in the use-case model. This provides several advantages for further analysis. The purification reduces the overall complexity of the use-case model.

The complexity analysis from the mathematical perspective has been discussed in the previous chapter. The comparison of the complexity of the preliminary model with the purified model will be discussed later in this chapter.

4.2.3. Aspect Realization

4.2.3.1 Map use-case selectors to start-point providers

This process is to describe the scenario of the advice cases. The results of this process are a set of OCUM diagrams. This process maps the use-case selectors of the aspects to the start-point providers. Figure 4.7 shows the start-point provider linking with the dynamic start-point "login."

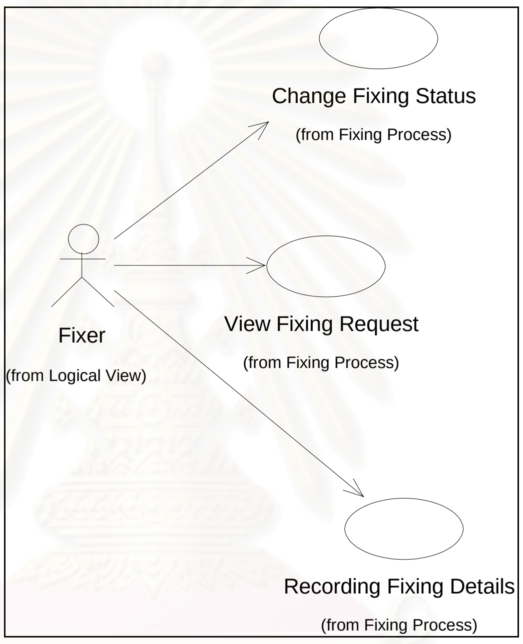


Figure 4.6. The purified use-case model for the case study

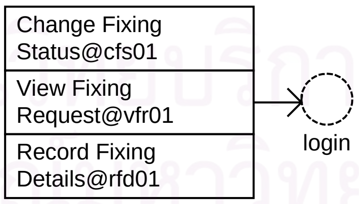

Change Fixing Status@cfs01
View Fixing Request@vfr01
Record Fixing Details@rfd01
login

Figure 4.7 The start-point provider for Login aspect

4.2.3.2 Create scenario for the advice case

The advice case Login is described in OCUM scenario. It contains two responsibilities for retrieving System User object, and assigning it to the person attribute of the new login session. Stub stacks will be placed on the paths of the services that will be modeled in the use-case explanation process. Figure 4.8 illustrates the scenario of the Login advice case.

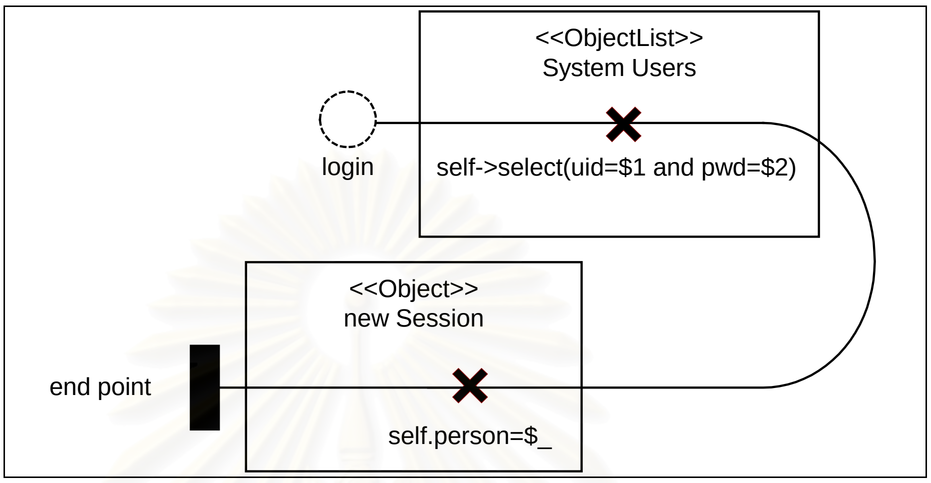


Figure 4.8 The scenario of Login advice case

4.2.4 Use-case explanation

Use-cases of the system that has been purified will be specified into the OCUM scenario in this step. The stub stacks will also be placed according to the description of related start-point providers, and the information from the advice case templates.

4.2.5 Analysis Aspect Identification

This step is to identify analysis aspects from the scenarios. The analysis aspect template is used for specifying details of analysis aspect. From the OCUM diagram in Figure 4.8, the aspect Login is described as follows.

4.2.5.1 Analysis Class Identification

Analysis classes usually are identified from the component blocks appearing in the scenario. According to the unified process [3], classes can be classified into three categories for the further detail design. Three categories are entity, control, and boundary. From the Login scenario, a number of classes can be captured as follows.

1. Class SystemUser: this class represents a user of the system. The class should be an entity class.
2. Class SystemUsers: this class represents a list of SystemUser. This class should be an entity class.
3. Class Session: this class is for maintain the login session of the user. This class should be a control class.
4. Class Person: this class is a specialization of the SystemUser. This class should be an entity class.

Table 4.9. The analysis-level aspect Login

Name	Login
Pointcuts declaration	before public void *.Perform()

Advice code	try {
	  su := SystemUsers->select(uid=$1 and pwd=$2);
	  session := new Session();
	  session.person := su;
	} catch (EUserNotFound e);

The steps of applying the requirements model to the case study have been illustrated. It can be clearly observed that crosscutting artifacts can be extracted from the main services of the system, and are modeled separately. This makes the analysis of the main services simpler. The above steps show how to handle functional crosscutting artifacts. Not only functional, but the nonfunctional artifacts, such as some kinds of system properties or quality attributes, are also modeled using this approach. Table 4.10 shows an example of specified nonfunctional crosscutting captured by the same activities.

4.3. Summary

From the case study, two use cases are identified as crosscutting artifacts. Thus, they are converted to be advice cases. The Numbers of tangled associations are reduced significantly. The complete purified use-case model of the system is also illustrated in the appendix III. Table 4.11 shows the comparison of purified use-case model with the preliminary use-case model using the complexity index.

Table 4.10 Nonfunctional Crosscutting Example

Name	Automatic Session Expire
Concern	Security
Description	Prevents other users using the logging session
Source	Supplementary documents
Crosscutting Type	Nonfunctional
Priority	MIN
Precedence Value	1000
Obligation	Optional
Influence	Design, Implementation
Models	Use cases: 1. Change Fixing Status 2. View Fixing Request 3. Recording Fixing Details Advice cases: 1. Login
Requirements	Requirements: 1. The user session should be expired in 2 minutes
Points	Before every services

Table 4.11 Complexity comparison of the preliminary and the purified model

	Number of Use Cases	Number of Advice Cases	Number of Associations	Complexity Index
Preliminary Model	20	0	27	1.35
Purified Model	18	2	20	1.00

From Table 4.11, it is clearly observed that the purifed model that are applied our approach has the smaller complexity value that the preliminary model. Comparing the complete system to the case study, Table 4.11 shows that our approach works better when using it in the more complex use-case model.

The next chapter will give conclusion, and finally end with the discussion in several open questions.

CHAPTER 5
CONCLUSIONS AND DISCUSSIONS

This thesis has presented the model of aspect-oriented requirements that support the unified software development process [1, 3]. This works applies the concept of the AOP, including join points abstraction, pointcut designators, and advice codes. The model proposed a number of activities to helps requirements engineers capture crosscutting concern during the early software development phase. These activities are the aspect extraction process, the use-case purification process, and the aspect realization process. These processes are designed to be parallel processes of the unified process.

The aspect extraction process is to extract crosscutting requirements, both functional and nonfunctional, out of the preliminary use-case model. A set of notations is introduced to help capturing these crosscutting artifacts. The notations are the use-case selector, the advice case, and the pointcut association. These notations are used to represent aspects in the CSM diagram.

The use-case purification process is to separate the use cases that are identified as advice cases out of the use-case model. This process is intended to simplify the use-case model. The purified use cases could be modeled using the traditional process of the unified process.

The aspect realization is the process for specifying the details of the advice cases, including its scenario, start points. This process describes aspects with the OCUM diagram. An OCUM diagram combines view of structural and behavioral into the single map. It is based on the Use Case Maps. OCUM has been enhanced to support AO paradigm, and better support OO paradigm. It introduced several concepts, such as the start-point provider, the dynamic start-point, the stub stack, the parameter symbols, and etc. The use of OCUM makes possible to model both AO, and OO scenario in the same diagram.

The use-case explanation process described in this thesis also employs OCUM for describing the scenario of use cases. Although, the scenario of use cases can be modeled using the UML notation, such as the activity graph, but the UML itself does not support the AO concept. With OCUM, describing use cases that are related to the AO paradigm is much better. The processes for capturing analysis artifacts from the OCUM are also proposed. These result the more complete software development process for AO paradigm. The case study is illustrated to show that this AO model can be solved the crosscutting modeling found in the real problem domain. It is clearly observed that the purified use-case model is easier to understand and analyzed that the preliminary model. This makes further analysis of the main services of the system much more simpler.

Moreover, two software tools are built to support the CSM, and OCUM model. Both software packages are created to be the Rational Rose add-ins. This makes better integrating the AO approach to the unified process.

There are some limitations in this approach. Although, the software tools can better support the change of requirements, the analysis of the impact of requirements changes is not covered here. This model is intended to support only the unified process. The generalized model of the early aspects management should further be proposed. The software tools support only Rational Rose. This may extend to support more modeling tools. The XMI specification proposed by OMG [24] could be considered for using as the file format. This makes possible for other tools to process the output of the software tools in this thesis.

Several open questions are induced by this work. Formalism of the CSM notations could be further invented. Improvement, and refinement of the OCUM model could be done in many ways. Consistency checking between two kinds of model, the CSM and its equivalent scenario, the OCUM model, should be considered. Identifying and specifying the crosscutting requirements using the natural language processing are possible, since the number of templates have been defined. Conflictions resolving among the aspects should be considered. There are some works in progression [9, 8] that investigate this approach. However, an automatic process is still needed. The standard crosscutting notations should be proposed to be part of the UML.

REFERENCES

1. Jacobson, I., G. Booch, and J. Rumbaugh, The Unified Software Development Process. The Addison-Wesley Object Technology Series. 1999: Addison-Wesley.
2. Kotonya, G. and I. Sommerville, Requirements Engineering: Processes and Techniques. 1997, Chicester: John Wiley & Sons Ltd. 282.
3. Rational, The Rational Unified Process. 2002, Rational Software.
4. AOSD.NET, AOSD.NET Homepage. http://www.aosd.net, AOSD.NET.
5. Kiczales, G., et al. Aspect-Oriented Programming. Proceedings European Conference on Object-Oriented Programming. 1997: Springer-Verlag.
6. OMG, The Unified Modeling Language Specification version 1.5. 2003, Object Management Group. http://www.omg.org/uml.
7. Xerox, AspectJ Homepage. http://www.aspectj.org/, Xerox Parc.
8. Rashid, A., et al. Early Aspects: A Model for Aspect-Oriented Requirements Engineering. IEEE Joint Conference on Requirements Engineering. 2002. Essen, Germany: IEEE Computer Society Press.
9. Araujo, J., et al. Aspect-Oriented Requirements with UML. UML 2002. 2002.
10. Moreira, A., J. Araujo, and I. Brito. Crosscutting Quality Attributes for Requirements Engineering. 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002). 2002. Italy: ACM Press.
11. Chung, L., et al., Non-Functional Requirements in Software Engineering. 2000: Kluwer Academic Publishers.
12. Malan, R. and D. Bredemeyer, Defining Non-Functional Requirements, http://www.bredmeyer.com/papers.htm.
13. Clark, R. and A. Moreira. Constructing Formal Specifications from Informal Requirements. Software Technology and Engineering Practice. 1997: IEEE Computer Society Press.
14. JavaSoft, Java 2 Platform, Enterprise Edition http://www.javasoft.com/j2ee, Sun Microsystems.
15. OMG, The UML Profile for CORBA, v 1.0. 2001, Object Management Group. http://www.omg.org/technology/documents/formal/profile_corba.htm.

16. Conallen, J., Building Web Applications with UML. The Addison-Wesley Object Technology Series. 1999: Addison Wesley.

17. Slic, B., G. Gullekson, and P. War, Real-Time OO Modeling. 1995: John Wiley & sons.

18. Buhr, R.J.A., Use Case Maps as Architectural Entities for Complex Systems. IEEE Transactions on Software Engineering, 1998. **24**(12): p. 1131-1155.

19. Buhr, R.J.A. and R.S. Casselman, Use Case Maps for Object-oriented Systems. 1995: Prentice Hall.

20. Amyot, D. and G. Mussbacher. On the Extension of UML with Use Case Maps Comcepts. <<UML>> 2000, 3rd International Conference on the Unified Modeling Language. 2000. York, UK.

21. Stein, D., S. Hanenberg, and R. Unland. A UML-based Aspect-Oriented Design Notation for AspectJ. Conference of Aspect-Oriented Software Development. 2002. Easchede, The Netherlands: ACM.

22. Kaewkasi, C. and W. Rivepiboon. Aspect-Oriented Extension for Capturing Requirements in Use-Case Model. The International Conference CAiSE'03 Forum. 2003. Austria: CAiSE'03.

23. Kaewkasi, C. and W. Rivepiboon. WWM: A Practical Methodology for Web Application Modeling. The 26th Annual International Computer Software and Applications Conference. 2002. Oxford: IEEE Computer Society.

24. OMG, XML Metadata Interchange (XMI) Specification, version 1.2. 2002, Object Management Group. http://www.omg.org/technology/documents/formal/xmi.htm.

APPENDICES

APPENDIX I
SOFTWARE TOOLS

Software tools

This appendix introduces software tools for using the aspect-oriented requirements model with the UML modeling tool, Rational Rose. There are two software packages. The first one is for modeling crosscutting requirements by employing the concept of CSM model. The second tool is for describing scenarios following the concept of OCUM model.

1. ASREM Add-in for Rational Rose

After installation of the ASREM add-in into Rational Rose, the new notations will appear in the Available toolbar buttons listbox. Two new buttons are the button labeled Create a use-case selector, and Create an advice case. Adding these buttons to the current toolbar by clicking on the "Add ->" button. Figure 1 shows the dialog for customizing the toolbar.

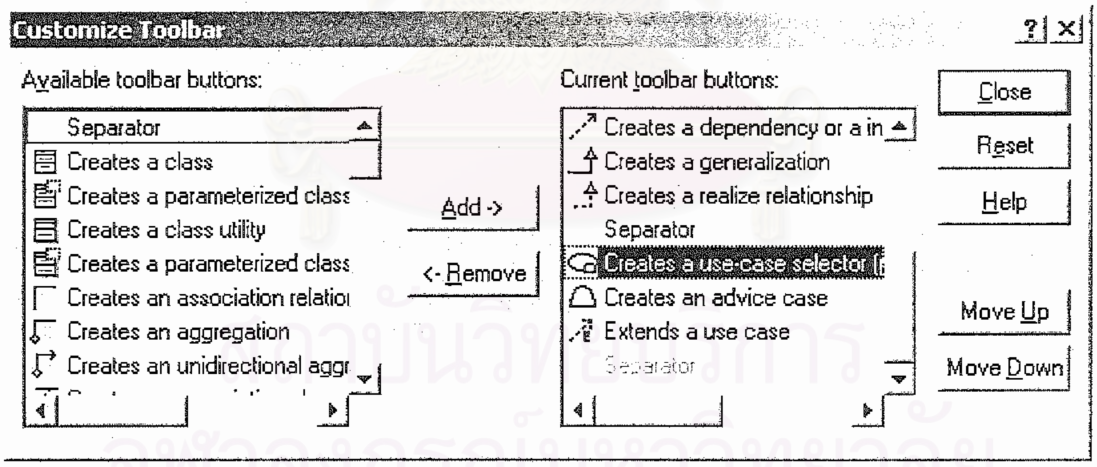


Figure I.1 Customizing the new notations in Rational Rose

The icons of the use-case selector, and the advice case now appear in the toolbar. The requirement engineers can use these icons to create an aspect by clicking on the use-case selector icon, then clicking on the diagram. Clicking the button of then advice case, then clicking on the diagram will create an advice case. The aspect can be completely defined by linking the association from the use-case selector to the advice case. Figure 2 shows the buttons and the example of the aspect creating with them.

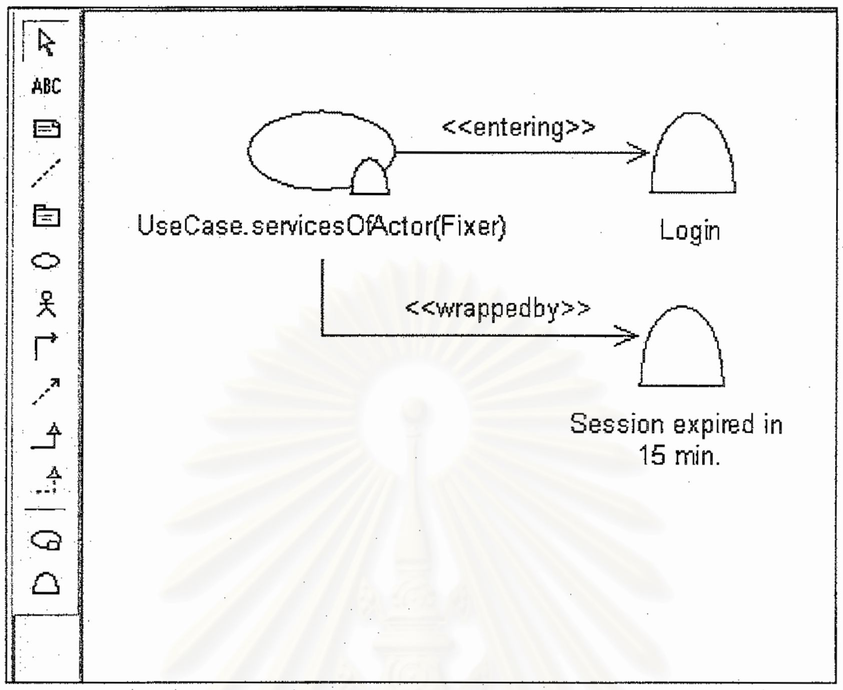
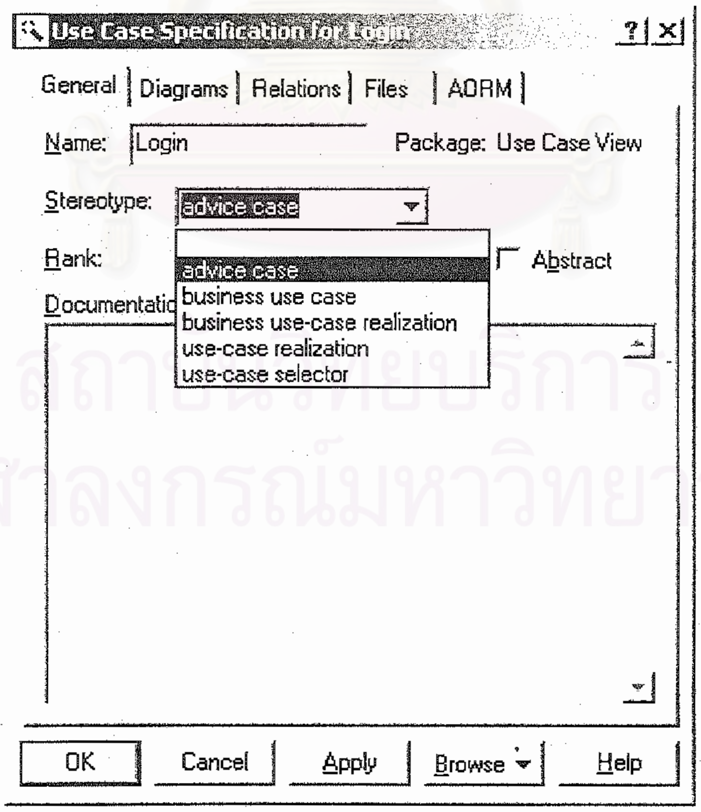


Figure I.2 The example of aspects modeled in Rose


Use Case Specification for Login

General | Diagrams | Relations | Files | AORM

Name: Login Package: Use Case View

Stereotype: advice case

Rank:

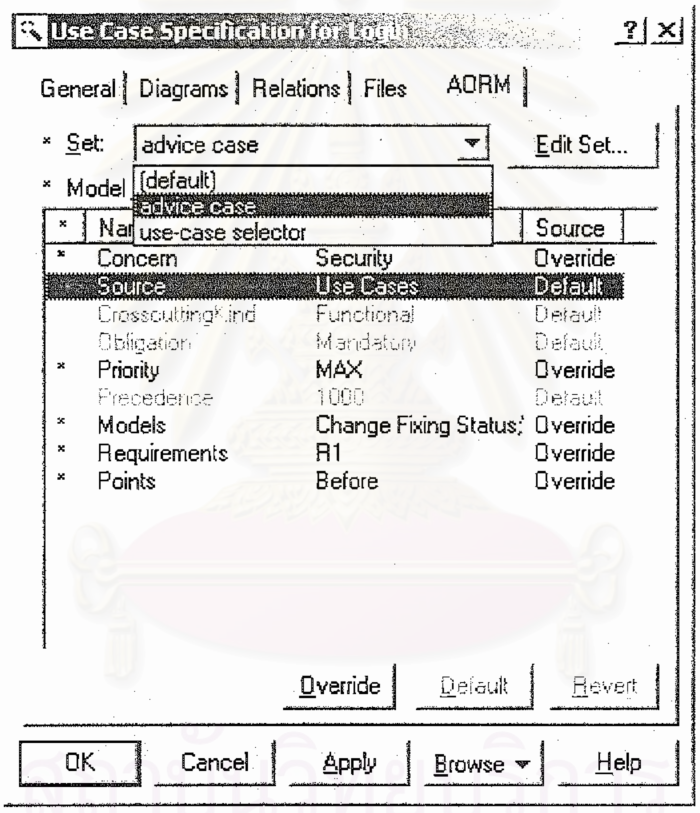
Abstract

Documentatio

Figure I.3 The specification dialog for the Login advice case

Double clicking an advice case in the diagram will show the specification dialog. Figure 3 shows the specification dialog for the Login advice case. Changing its stereotype is possible by selecting another from the stereotype dropdown.

Additional property page AORM is for specifying the template details for the advice case. These tagged values are Concern, Source, CrosscuttingKind, Obligation, Priority, Precedence, Models, Requirements, and Points. They are identical to the template properties from Table 3.2.



* Set: advice case

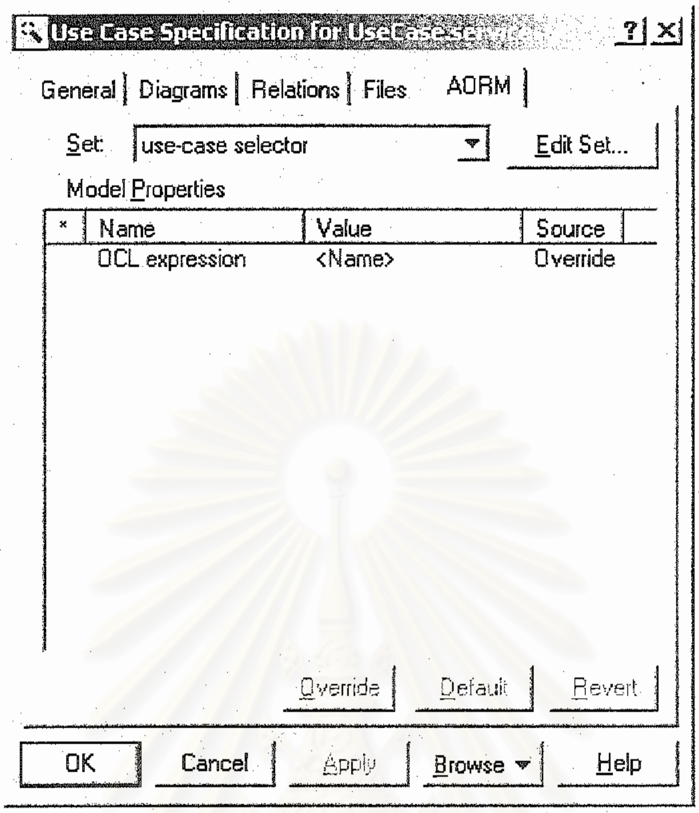
* Model

(default)
advice case
use-case selector

*	Name	Value	Source
*	Concern	Security	Override
	Source	Use Cases	Default
	CrosscuttingKind	Functional	Default
	Obligation	Mandatory	Default
*	Priority	MAX	Override
	Precedence	1000	Default
*	Models	Change Fixing Status;	Override
*	Requirements	R1	Override
*	Points	Before	Override

Figure I.4 The tagged values for specifying additional details to the advice case

Double clicking on the use-case selector will display the specification dialog for it. There is only one tagged value property for the use-case selector. This tagged value is for specify the OCL expression. By default, the OCL expression is the same value to the use-case selector's name. Figure 5 shows the specification dialog for the use-case selector.



Set: use-case selector

Model Properties

*	Name	Value	Source
	OCL expression	<Name>	Override

Figure I.5 The specification dialog for a use-case selector

2. OCUM Vectra

OCUM Vectra is a modeling tool for OCUM models. It supports all OCUM notations described in chapter 3. It is designed to be able to exchange the model with Rational Rose. The structural part of OCUM model can be exported to the class diagram in Rational Rose and vise versa.

Figure 6 shows a window of the OCUM Vectra. OCUM Vectra workspace consists of the toolbar, the object tree view, the notation bar, the object documentation box, and the drawing area. The Toolbar contains a number of buttons for common tasks such as loading a diagram, saving a diagram, exporting a diagram to a Rose model, and etc. OCUM Vectra saves the scenario in an XML format. This makes other tools possible to process an OCUM model. Figure 7 shows the diagram in an XML output.

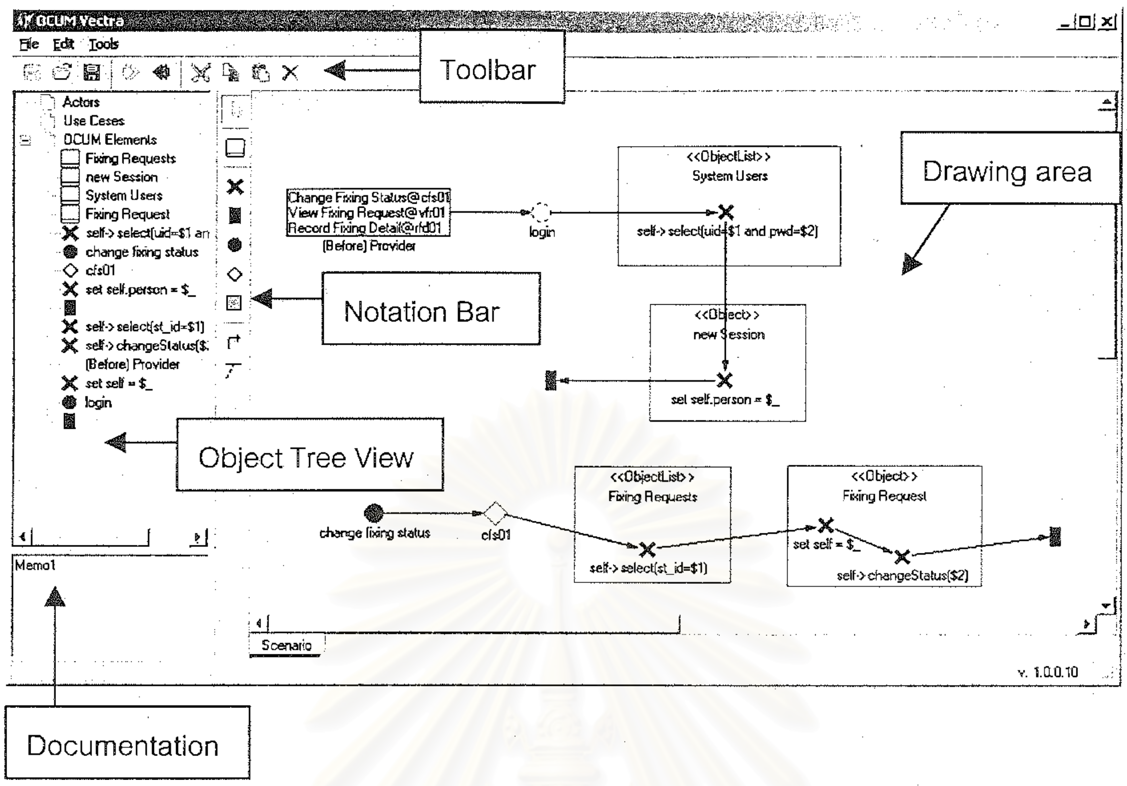
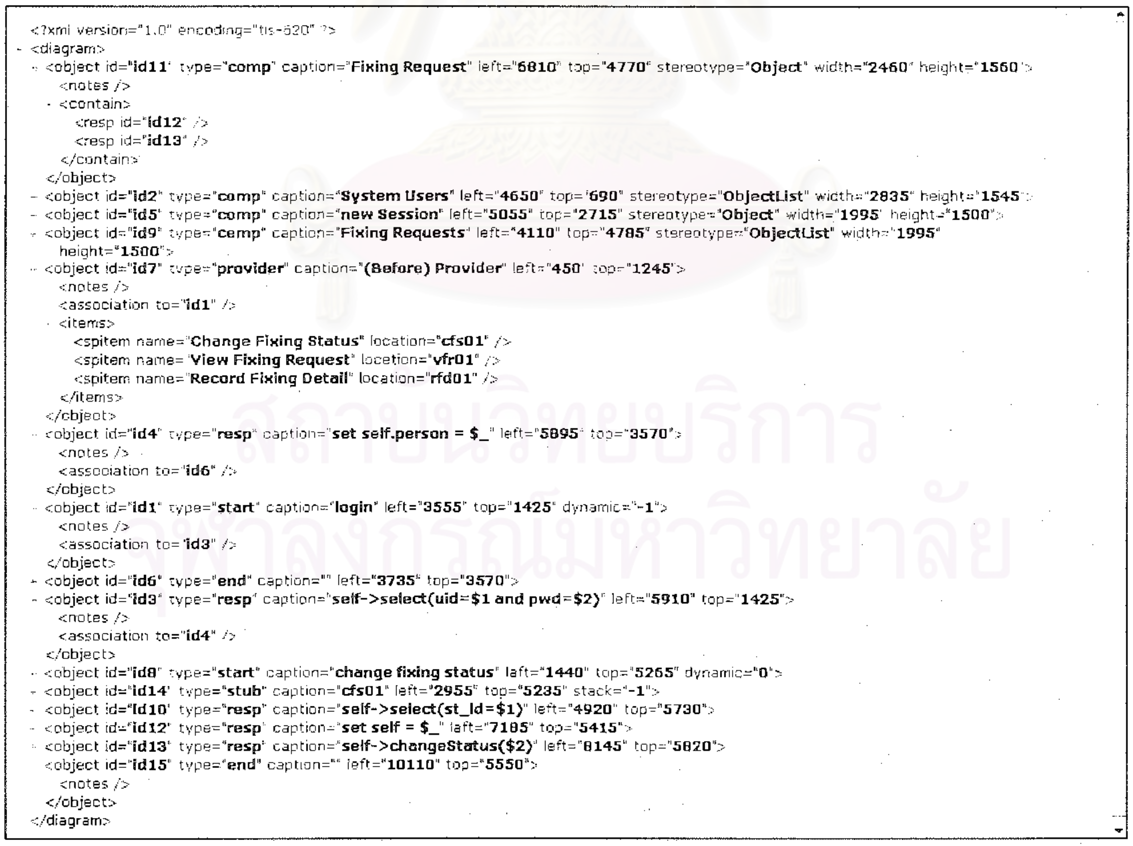


Figure I.6 The OCUM Vectra window



```
<?xml version="1.0" encoding="tis-620" ?>
- <diagram>
  - <object id="id11" type="comp" caption="Fixing Request" left="6810" top="4770" stereotype="Object" width="2460" height="1560">
      <notes />
    - <contain>
        <resp id="id12" />
        <resp id="id13" />
      </contain>
    </object>
  - <object id="id2" type="comp" caption="System Users" left="4650" top="690" stereotype="ObjectList" width="2835" height="1545">
  - <object id="id5" type="comp" caption="new Session" left="5055" top="2715" stereotype="Object" width="1995" height="1500">
  - <object id="id9" type="cemp" caption="Fixing Requests" left="4110" top="4785" stereotype="ObjectList" width="1995"
      height="1500">
  - <object id="id7" type="provider" caption="(Before) Provider" left="450" top="1245">
      <notes />
      <association to="id1" />
    - <items>
        <spitem name="Change Fixing Status" location="cfs01" />
        <spitem name="View Fixing Request" location="vfr01" />
        <spitem name="Record Fixing Detail" location="rfd01" />
      </items>
    </cbject>
  - <object id="id4" type="resp" caption="set self.person = $_" left="5895" top="3570">
      <notes />
      <association to="id6" />
    </object>
  - <object id="id1" type="start" caption="login" left="3555" top="1425" dynamic="-1">
      <notes />
      <association to="id3" />
    </object>
  + <object id="id6" type="end" caption="" left="3735" top="3570">
  - <object id="id3" type="resp" caption="self->select(uid=$1 and pwd=$2)" left="5910" top="1425">
      <notes />
      <association to="id4" />
    </object>
  - <object id="id8" type="start" caption="change fixing status" laft="1440" top="5265" dynamic="0">
  - <object id="id14" type="stub" caption="cfs01" left="2955" top="5235" stack="-1">
  - <object id="Id10" type="resp" caption="self->select(st_Id=$1)" left="4920" top="5730">
  - <object id="id12" type="resp" caption="set self = $_" laft="7185" top="5415">
  - <object id="id13" type="resp" caption="self->changeStatus($2)" left="8145" top="5820">
    <object id="id15" type="end" caption="" left="10110" top="5550">
      <notes />
    </object>
</diagram>
```

Figure I.7 The XML output from OCUM Vectra

APPENDIX II

ADD-IN IMPLEMENTATION FOR RATIONAL ROSE

Overview

This appendix discusses the techniques used for implementing the software tools ASREM, and OCUM Vectra. ASREM is implemented as Rational Rose add-in notations, while OCUM Vectra is implemented as a Rational Rose tool. ASREM provides new stereotypes, and tagged values using Rose stereotype, and tagged values definition. OCUM Vectra provides exporting its objects to Rose through Rose document object model using Rose COM object. All techniques are discussed below.

Stereotype Definition

The way to create user-defined notations in Rose can be done by defining new stereotype in the INI file named *defaultstereotypes.ini*. This file locates in installation path of Rose. In the section *[Stereotype Items]* in that file, our new stereotypes are defined. Figure II.1 shows their definition.

```
[Stereotyped Items]

. . .

;======= EARLY ASPECT =======
;=(c) 2002 Chanwit Kaewkasi =

Use Case:use-case selector
Use Case:advice case
Association:entering
Association:leaving
Association:exception
```

Figure II.1 Stereotype definitions in "defaultstereotypes.ini" file

From Figure II.1, the use-case selector, and the advice case stereotype are available on the notation of the Use Case. The entering, leaving, and exception stereotype are available on the notation of the Association.

Icons of these stereotypes are also defined in this INI file. Each stereotype has its own section to define its icons. The icon of the stereotype that will appear on the drawing canvas of Rose must be in an EMF, or a WMF file format. Both file formats, which are formally known as Meta file format, store picture in a vector form. The stereotype's palette icons consist of three bitmaps with different kind for the medium scale, small scale, and list type. Each bitmap icon must be in a BMP format. Different bitmaps are used in different places in Rose, such as in the object tree view, or the toolbar. A tool tip label of each stereotype can also be specified in each stereotype section. Figure II.2 shows an example of stereotype specification detail of pointcut association *Entering*, and the use-case selector.

```
. . .

[Association:entering]
Item=Association
Stereotype=entering

. . .

[Use Case:use-case selector]
Item=Use Case
Stereotype=use-case selector
Metafile=&\stereotypes\normal\usecase_selector.emf
SmallPaletteImages=&\stereotypes\small\usecase_selector_s.bmp
SmallPaletteIndex=1
MediumPaletteImages=&\stereotypes\medium\usecase_selector_m.bmp
MediumPaletteIndex=1
ListImages=&\stereotypes\list\usecase_selector_l.bmp
ListIndex=1
ToolTip=Creates a use-case selector (pointcut)\nAspectOriented use-case selector

. . .
```

Figure II.2 Stereotype specification details

Tagged Value Definition

Rose supports tagged values definition in its own format called property file with PTY extension. Tagged values are specially defined for a stereotype. New tagged values will appear in new tab in the specification dialog. Figure II.3 shows tagged value definition of the advice case.

```
(object  Petal version 42)
(list Attribute_Set
  (object Attribute tool "AORM" name "propertyId" value "9317696821")
  (object Attribute tool "AORM" name "advice case__UseCase" value
    (list Attribute_Set
      (object Attribute tool "AORM" name "Concern" value "<Not Specified>")
      (object Attribute tool "AORM" name "Source" value "Use Cases")
      (object Attribute tool "AORM" name "CrosscuttingKind" value "Functional")
      (object Attribute tool "AORM" name "Obligation" value "Mandatory")
      (object Attribute tool "AORM" name "Priority" value "")
      (object Attribute tool "AORM" name "Precedence" value 1000)
      (object Attribute tool "AORM" name "Models" value "")
      (object Attribute tool "AORM" name "Requirements" value "")
      (object Attribute tool "AORM" name "Points" value "")
    )
  )
)
```

Figure II.3 Tagged values definitions for the advice case stereotype

Rose Document Object Model

Add-in program can access Rose document object model (DOM) via Rose COM object. Rose COM object provides rich interfaces to control its element. Its root interface is IRoseApplication. To get access to Rose application, the add-in tool must create a Rose application object, then access to the Rose application via IRoseApplication interface. Figure II.4 shows code snippet for exporting OCUM components to a class diagram in Rose.

```
var
  i: integer;
  RoseClass: IRoseClass;
  Comp: TOCUMComp;
  RoseCat: IRoseCategory;
  RoseClassDiagram: IRoseClassDiagram;
begin
  if RoseApp = nil then begin
    RoseApp := CoRoseApplication.Create;
    RoseApp.Visible := true;
  end;

  if OpenDialog2.Execute then begin
    ModelFilename := OpenDialog2.FileName;
  end;

  if RoseModel = nil then begin
    RoseApp.OpenModel(ModelFilename);
    RoseModel := RoseApp.CurrentModel;
  end;

  RoseCat := RoseModel.GetAllCategories.GetFirst('Logical View');
  RoseClassDiagram := RoseCat.ClassDiagrams.GetFirst('Main');

  for i := 1 to AddFlow1.Nodes.Count do begin
    if AddFlow1.Nodes.Item(i).Tag = 'comp' then begin
      Comp := TShapeFactory.getNodeAsComp(AddFlow1.Nodes.Item(i));
      RoseClass := RoseCat.AddClass(Comp.Caption);
      if RoseClass = nil then begin
        RoseClass := RoseCat.GetAllClasses.GetFirst(Comp.Caption);
      end;
      Comp.Uid := RoseClass.GetUniqueID;
      RoseClass.Documentation := Comp.Notes;
      RoseClass.Stereotype := Comp.StereoType;
      RoseClassDiagram.AddClass(RoseClass);
    end;
  end;
end;
```

Figure II.4 Code of classes exporting in OCUM Vectra

IRoseApplication has IRoseModel interface member. IRoseModel is responsible to manipulate the UML model in Rose. The model contains object categories that normally are for use-case diagrams, class diagrams, and deployment diagrams. The category containing class diagrams are called 'Logical View'. Code in Figure II.4 retrieves the *Main* class diagram from the *Logical View* category, and then traverses all OCUM diagram to add OCUM components to the Logical View category via the

AddClass method of IRoseCategory interface. After adding classes to the Logical View category, all classes are displayed in the Main class diagram using the AddClass method of IRoseClassDiagram interface.

APPENDIX III
PROBLEM DOMAIN FOR CASE STUDY

Preliminary Use-Case Model

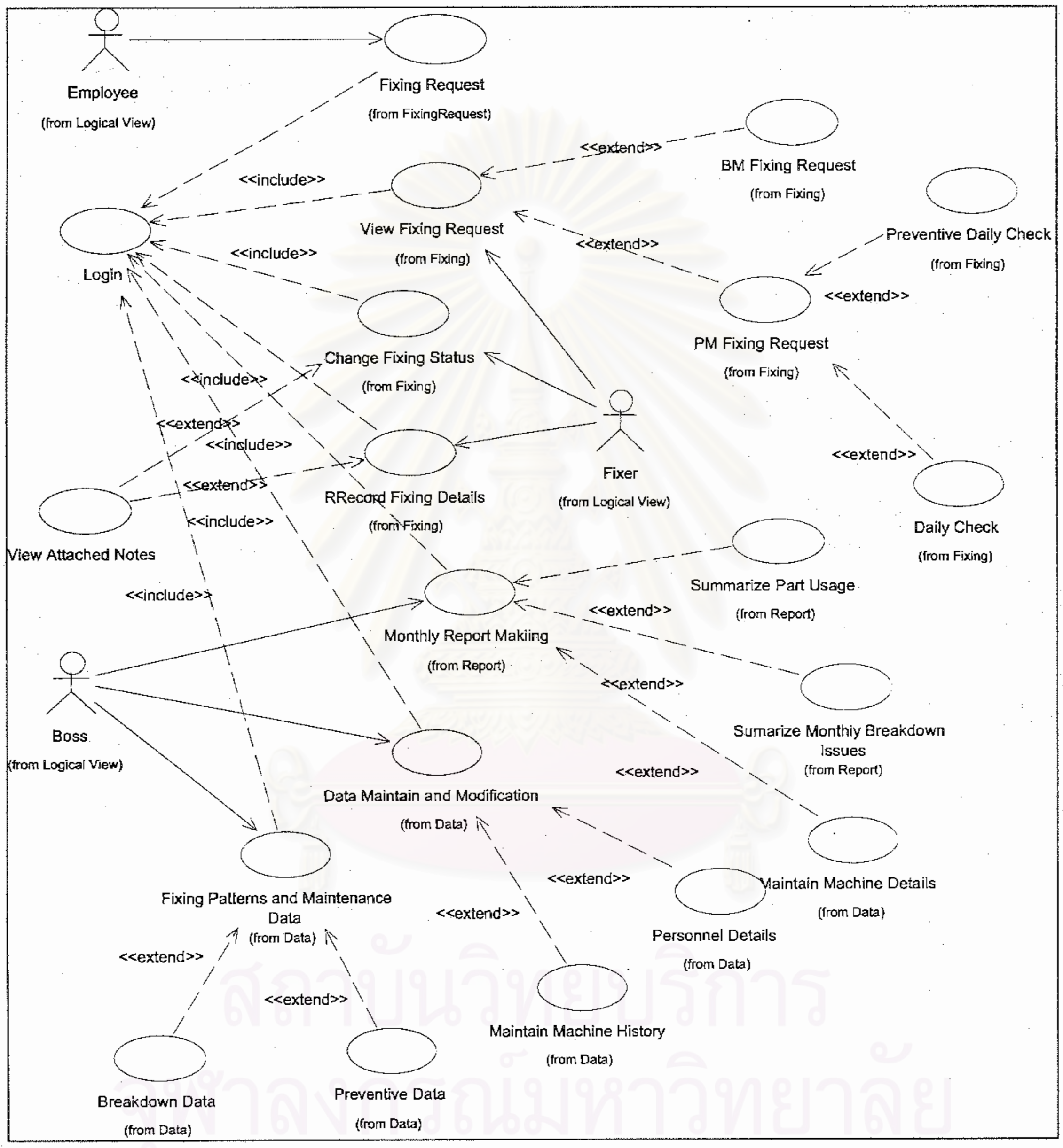


Figure III.1 Preliminary use-case diagram for the maintenance management system

Purified Use-Case Model

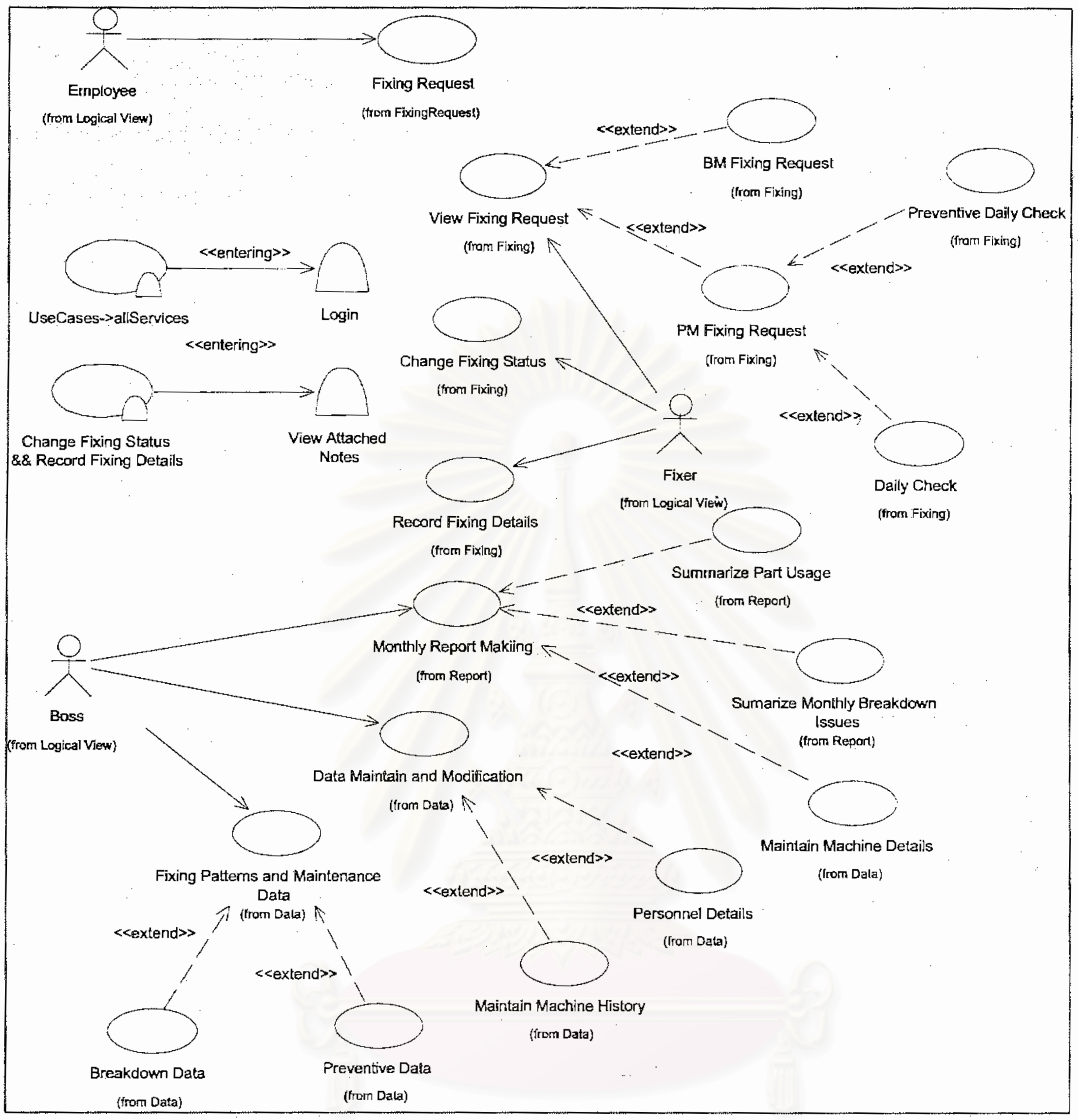


Figure III.2 Purified use-case diagram with CSM notations

BIOGRAPHY

Mr. Chanwit Kaewkasi was born on July 29, 1978 at Suratthani, and got a Bachelor Degree in Computer Engineering (First Class Hon.) at Suranaree University of Technology, Nakorn Ratchasima, in 2000.