Transformation of UML 2.0 Models with Performance Annotations to Core Scenario Models

by

Toqeer A Israr

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

Master of Applied Sciences in Electrical Engineering

Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada

August 2005

©Copyright 2005, Toqeer A Israr
Abstract

The thesis proposes and implements a method for transforming a UML 2.0 model with performance annotations into an equivalent Core Scenario Model (CSM). The input to the transformation algorithm is a UML 2.0 model generated by a UML tool, either as an internal data structure, or as an XML file according to the XMI standard. The software specifications models in UML 2.0 are annotated using the UML Profile for Schedulability, Performance, and Time (SPT). The output of the transformation algorithm is the corresponding CSM description file in XML format, which can be transformed later to other performance models such as Queueing Networks, Layered Queuing Networks, Petri Nets, etc. The thesis describes the proposed transformation rules both at the UML 2.0 notation level, which are more intuitive, and at UML meta-model level, which corresponds to the actual implementation. A CSM model is generated from different types of UML diagrams that describe the software architecture, the deployment to hardware devices and the behaviour of key performance scenarios. The proposed transformation algorithm was implemented in Java, by using the open source platform Eclipse.
Acknowledgements

First and foremost, I would like to thank Allah, the Most Glorious One, for everything that I have. I am quite confident that my success can be credited to the prayers of my grandparents and parents.

To my parents, who have persistently given me love, hope and their infinite support and time to form me the person I am today, I love you from the bottom of my heart. Junaid Bhai & Shazima Bhabhi – thanks for your continuous struggle and belief in me. Tauseef bhai & Arooj Bhabhi – their everlasting love and such amazing devotion to assist me in the completion of this project. Moeed Bhai & Huma Bhabhi – their understanding, love and support have kept me grounded to complete this thesis, I would like to thank you all for everything that you all have done for me!

To my dearest Naima. Though u will never realize how important you are to me, I want you to know that I could not have completed this without you. Thank you for your unconditional love and support.

I would like to thank my supervisor, Dr. Dorina Petriu, for providing an interesting topic, constant direction, valuable advice, and most of all the opportunity to work with great minds.

Also thanks to Mr. Bran Selic and Dr. Murray Woodside for their time and effort. Their advice and encouragement considerably enhanced the quality of this thesis.

Finally I would like also like to thank my friends for all their help and support: RL, UF, NB, IA, AM, RW, SJ, SK, TY, TM, NS, AK, AK, MM, MB & BG
# Table Of Contents

CHAPTER 1. INTRODUCTION ......................................................................................... 1
   1.1 Thesis Motivation ...................................................................................... 1
   1.2 Scope of the Thesis Research .................................................................. 3
   1.3 Thesis Contributions ............................................................................... 5
   1.4 Thesis Content ......................................................................................... 6

CHAPTER 2. LITERATURE REVIEW ......................................................................... 7
   2.1 Unified Modeling Language 2.0 ................................................................. 7
   2.2 UML Meta-Model ..................................................................................... 9
   2.3 XML & XMI ............................................................................................ 10
      2.3.1 XML ................................................................................................. 10
      2.3.2 XMI ................................................................................................ 12
   2.4 UML Design Tools .................................................................................. 14
      2.4.1 Eclipse ............................................................................................. 14
      2.4.1.1 Plug-ins ....................................................................................... 15
      2.4.1.2 Workspace .................................................................................. 16
      2.4.1.3 Workbench and UI Toolkits ......................................................... 16
      2.4.1.4 Team Support ............................................................................. 18
      2.4.1.5 Help ............................................................................................ 18
   2.5 Rational Software Architect .................................................................... 18
   2.6 UML2 Plug-in .......................................................................................... 20
   2.7 UML Profile ............................................................................................ 20
      2.7.1 UML Schedulability Performance and Time Profile ......................... 21
   2.8 Performance Modeling Techniques ......................................................... 22
   2.9 Transformations of UML Models into Performance Models ............... 23

CHAPTER 3. INPUT AND OUTPUT MODELS ............................................................... 25
   3.1 Input Models ............................................................................................ 25
      3.1.1 Unified Modeling Language .................................................................. 25
         3.1.1.1 Discussion of Structural Diagram Meta-model ............................ 26
         3.1.1.1.1 Discussion of Class Diagram Meta-model ............................... 26
         3.1.1.1.2 Discussion of Deployment Diagram Meta-model ................... 27
         3.1.1.2 Discussion of Behavioural Diagram Meta-model ..................... 27
         3.1.1.2.1 Discussion of Sequence Diagram Meta-model ..................... 28
      3.1.2 Adapted Performance Profile ............................................................... 33
         3.1.2.1 PAclosedLoad Stereotype ......................................................... 34
         3.1.2.2 PAcontext Stereotype ............................................................... 35
         3.1.2.3 PAhost Stereotype .................................................................... 36
         3.1.2.4 PAopenLoad Stereotype ............................................................ 37
         3.1.2.5 PAcontext Stereotype ............................................................... 38
         3.1.2.6 PAstep Stereotype .................................................................... 39
      3.1.3 Use of Adapted Performance Profile in Rational Software Architect ... 40
   3.2 Output Models ......................................................................................... 44
CHAPTER 4. TRANSFORMATION FROM UML 2.0 TO CSM .............................................. 54

3.2.1 Core Scenario Model (CSM) ................................................................................. 44
3.2.1.1 Behavioural Part of CSM .................................................................................. 44
3.2.1.2 Structural Part of CSM .................................................................................... 45

3.3 UML 2 GRAPHICAL NOTATION AND THE UNDERLYING REPRESENTATION ...... 47

3.3.1 Object Model – UML 2.0 Behaviour ..................................................................... 47
3.3.1.1 Simple Interaction ............................................................................................. 47
3.3.1.2 Interaction Occurrence ...................................................................................... 49
3.3.1.3 Combined Fragment – Optional ....................................................................... 50

3.4 PROGRAMMATIC GENERATION OF UML MODELS ............................................. 52

CHAPTER 5. IMPLEMENTATION AND TESTING ........................................................ 106

5.1 IMPLEMENTATION ..................................................................................................... 106
5.1.1 PreProcessing to Produce Internal Data Structure .............................................. 106
5.1.1.1 PathConnector Class ....................................................................................... 106
5.1.1.2 PathFragment Class ......................................................................................... 107
5.1.1.3 Discussion of PreProcessing ............................................................................ 108

5.1.2 Transformation from Internal Data Structure to CSM ....................................... 110
5.2 Testing and Verification ................................................................. 111
  5.2.1 Testing with Various Test Scenarios ................................................ 113
  5.2.2 Verification .................................................................................. 115

CHAPTER 6. CASE STUDY ............................................................................. 116
  6.1 Description of the Building Security System ........................................... 116
  6.2 Use Cases ............................................................................................ 117
    6.2.1 Access Control Scenario ................................................................. 117
    6.2.2 Acquire / Store Video ...................................................................... 118
    6.2.3 Manage Rights ............................................................................... 118
  6.3 Deployment Diagram ............................................................................ 119
  6.4 Scenarios ............................................................................................... 120
    6.4.1 Acquire / Store .............................................................................. 120
    6.4.2 Access Control .............................................................................. 125
    6.4.3 Manage Rights ............................................................................... 130

CHAPTER 7. CONCLUSION ............................................................................. 135
  7.1 Conclusion ............................................................................................ 135
  7.2 Future Work .......................................................................................... 136

REFERENCES ................................................................................................. 138

APPENDIX A. TRANSFORMATION ALGORITHM ............................................ 143
APPENDIX B. CSM MODEL FOR BSS CASE STUDY ...................................... 149
List of Figures

Figure 1-1 UML 2.0 Interaction Model to CSM ................................................................. 4
Figure 2-1 4-layer Meta-model Architecture ................................................................. 10
Figure 2-2 - Eclipse Platform architecture ................................................................... 15
Figure 3-1 - Diagrams in UML 2.0 ................................................................................ 26
Figure 3-2 Partial Meta-model for Deployment ............................................................. 27
Figure 3-3 - Partial Meta-model for Model .................................................................... 31
Figure 3-4 - Partial Meta-model for Interaction Fragment ........................................... 31
Figure 3-5 - Partial Meta-model for Lifeline ................................................................. 32
Figure 3-6 - Partial Meta-model for Message ............................................................... 32
Figure 3-7 - Partial Meta-model for Combined Fragment ............................................ 33
Figure 3-8 - Partial Meta-model for Interaction Occurrence ....................................... 33
Figure 3-9 - Screenshot of RSA - Profiles ..................................................................... 40
Figure 3-10 - Screenshot of RSA - Selection of Profiles ............................................... 41
Figure 3-11 - Screenshot of RSA - Applying Stereotype .............................................. 42
Figure 3-12 - Screenshot of RSA - Element’s Properties ............................................. 43
Figure 3-13 - CSM Meta-model ...................................................................................... 46
Figure 3-14 - Graphical Notation of a Simple Interaction ............................................. 47
Figure 3-15 - Simple Interaction Object Containment ................................................ 48
Figure 3-16 - Graphical notation of Interaction Occurrence ....................................... 49
Figure 3-17 - Object Containment for Interaction Occurrence .................................... 50
Figure 3-18 - Graphical Notation of Combined Fragment ........................................... 50
Figure 3-20 - Example Code ......................................................................................... 52
Figure 4-1 - Deployment diagram relationship ............................................................ 60
Figure 4-2 - Structural Transformation ........................................................................ 61
Figure 4-3 - Beginning of Scenario Transformation .................................................... 62
Figure 4-4 - Ending of Scenario Transformation .......................................................... 63
Figure 4-5 - Simple Synchronous Message Transformation ....................................... 64
Figure 4-6 - Simple Asynchronous Message Transformation ..................................... 66
Figure 4-7a - Alternative Combined Fragment Transformation .................................. 67
Figure 4-7b - Alternative Combined Fragment Transformation .................................. 68
Figure 4-8a - Parallel Combined Fragment Transformation ....................................... 70
Figure 4-8b - Parallel Combined Fragment Transformation (continued) ...................... 71
Figure 4-9 - Optional Combined Fragment Transformation ......................................... 73
Figure 4-10 - Loop Combined Fragment Transformation ............................................ 75
Figure 4-11 - Simple Scenario ..................................................................................... 77
Figure 4-12 - Asynchronous Message ........................................................................... 78
Figure 4-13 - Interaction Occurrence Transformation ............................................... 80
Figure 4-14 - Optional Combined Fragment ............................................................... 82
Figure 4-15a - Parallel Combined Fragment ............................................................... 85
Figure 4-15b - Parallel Combined Fragment ............................................................... 86
Figure 4-16b - Alternate Combined Fragment .............................................................. 93
Figure 4-17a - Loop Combined Fragment .................................................................... 95
Figure 4-17b - Loop Combined Fragment ................................................................... 96
Figure 4-18 - Analyze Interaction Flowchart ............................................................... 99
List of Tables

Table 1 – PAclosedLoad Stereotype ..................................................................................... 34
Table 2 – PAcontext Stereotype ............................................................................................ 35
Table 3 – PAhost Stereotype .................................................................................................. 36
Table 4 – PAopenLoad Stereotype ....................................................................................... 37
Table 5 – PAcontext Stereotype ............................................................................................ 38
Table 6 – PAstep Stereotype .................................................................................................. 39
Table 7a – Mapping of Stereotypes to various CSM Elements .......................................... 55
Table 7b – Mapping of Stereotypes to various CSM Elements .......................................... 56
Table 8a – Mapping of Attributes from Performance Profile to CSM ............................... 57
Table 8b – Mapping of Attributes from Performance Profile to CSM (continues Table 8a) ......................................................................................................................................... 57
Table 9 – Mapping of Attributes from UML 2.0 to CSM ................................................... 58
Table 10 – TestCase Summary ............................................................................................ 114
Chapter 1. Introduction

1.1 Thesis Motivation

A need for an industry standard for the design and visualization of software systems has led to the emergence of Unified Modeling Language (UML) by Object Management Group (OMG). UML is the result of the convergence of notations from three main object-oriented methods, namely OMT (by James Rumbaugh), OOSE (by Ivar Jacobson) and Booch (by Grady Booch) [Booch+99]. UML provides a standard way of describing various systems by providing several kinds of diagrams, which supports multiple views of the same system, like structural and behavioral aspects, interactions amongst different system components and various actual physical implementation details.

Performance analysis is a very important issue, especially for real-time systems, which sometimes tends to get ignored till late into the software development process. It has been recognized that software analysis needs to be integrated throughout the software life-cycle [Smith90]. The overhead cost for performing performance analysis at the end of the development cycle can be considerable and could cause severe performance disasters in software projects.

To bridge the gap between software analysis and performance analysis, OMG introduced the UML Profile for Schedulability, Performance, and Time Specification [ProfileSPT1]. This profile adds light-weight extensions to the UML meta-model to express the basic abstractions used in performance analysis. Adding quantitative performance annotations to UML models enables their transformation into performance...
models, which can be analyzed with existing performance analysis tools. The performance analysis results can be fed back to the engineers as well as the customers at an earlier stage of the development allowing them to refine the models as necessary.

There are various performance models used in the research community and in the industry, such as Layered Queuing Networks (LQN) [Woodside+95], Extended Queuing Networks (EQN) [Williams+98], Stochastic Timed Petri Nets (STPN) [King+99], Stochastic Process Algebra (SPA) [Pooley99] and many others.

One of the current research efforts in software performance engineering is to automate the process of obtaining performance models from UML software specifications. One of the advantages of this research is to automate the construction of performance models from a UML model and to keep the two models consistent. Any changes in the original software specifications would be reflected in the performance models regardless of the stage in the systems’ life cycle. It would be very beneficial to let the software developers know of any potential performance problems that need to be resolved. Another advantage of the automatic transformation is that software developers would not need to know how to generate the performance models, but only how to add performance annotations to their UML models and have those transformed automatically into performance models.

Since performance is a dynamic property, in order to determine the performance characteristics of a system it is necessary to study the scenarios from the UML model. The SPT Profile [ProfileSPT1] defines a scenario as an ordered sequence of steps, each of which can be a sub-scenario. The UML 2.0 standard [UML2.0Super] describes a
scenario as the means of illustrating behavior by a specific sequence of actions. A
scenario can also be described to illustrate the execution of a use case instance providing
the means for the end user and its developers to state the expectations about the behavior
of the system.

Scenarios are often modeled using use case diagrams, interaction diagrams,
activity diagrams or state machines. These diagrams emphasize different aspects of the
system. Use cases can be used to describe the required usage of a system, to describe
what a system is supposed to do. State machines are usually used to specify the behavior
of a model element, and hence provide a localized view of an object; scenarios are
represented by the composed behaviour of a set of cooperating objects. Interaction
diagrams give the description of the behavioral aspect of a system from object to object
and method to method. Interaction diagrams, however, often describe a single scenario
or a part of a scenario. Another way to represent a scenario is by using activity diagrams,
which describe the overall flow of control. While interaction diagrams represent only
inter-object interactions, activity diagrams can represent behaviour at any granularity
level (both inter- and intra-object).

1.2 Scope of the Thesis Research

This research, being part of a bigger project, is responsible for generating
automatically Core Scenario Models from models given in UML 2.0 annotated with the
performance profile, as shown in Figure 1-1.
Part of this thesis responsibility is to accept an annotated UML model composed of class, deployment and the sequence diagrams as the input and transform it into the corresponding Core Scenario Model. This approach is described in Figure 1-1. The input describing the various UML 2.0 diagrams is created by the user either with the graphical editor tool such as Rational Software Architect [RationalRSA], or programmatically with an Eclipse plug-in named UML2 Plug-in [UML2.0Plug-in]. The graphical editor tool exports the model in an XMI format, which Eclipse XMI reader reads in and creates the internal data structure using UML Meta Model Library. Meanwhile, if the model is created programmatically, the internal data structure of the input UML models is created already in Java. The appropriate information is extracted from the internal data structure and transformed, as per the transformation rules, using the UML meta-model library into CSM. The transformation is performed in Java using Eclipse and the output is generated as CSM Java objects. Finally, the CSM output, which is in Java at this time, is rendered in XML using the Eclipse XML writer facility.
As mentioned, various UML graphical editors can be used to generate the input for the transformation engine. However, the transformation engine expects the XMI of the UML model. If only all those UML graphical editors produce the same XMI using the same schema, then can this transformation engine. The transformation engine would need to be revised if the XMI generated are from different schemas. On the other hand, if the assumption is made that a standard XMI is being implemented by all the tools, then there should be no problem using this transformation engine with the other UML graphical tools.

1.3 Thesis Contributions

The contributions of the thesis are as follows

- Propose a method for the automatic transformation of the UML 2.0 model annotated with performance information into Core Scenario Models. The transformation rules proposed here will take into account the characteristics of the interacting objects, and will create a CSM for every scenario defined as a sequence diagram.

- Design, implement and test a Java application running on top of Eclipse that achieves the above transformation from UML 2.0 to CSM.
1.4 Thesis Content

The thesis consists of six chapters described as follows:

The overview of the background literature for the thesis is described in Chapter 2. UML, MOF, XMI are discussed first, followed by UML model transformations. UML tools are discussed next along with the Performance Profile as well as Scenario Models.

Chapter 3 consists of the discussion of the internal structure of the UML models, focusing on the diagrams that will be transformed to CSM. UML 2.0 meta-objects are described and the programmatic generation of the UML 2.0 models is explained.

Chapter 4 discusses the transformation algorithm from UML 2.0 to CSM. This describes the generation of the CSM structure (software and hardware resources) as well as the generation of the CSM scenarios.

A Case Study is be presented in Chapter 5 which deals with a specific example. The internal structure of the UML 2.0 model, generated CSM and verification and testing are discussed.

Chapter 6 gives the conclusion of the thesis research and discusses possibilities of future work.
Chapter 2. Literature Review

The focus of this chapter is to provide background information of the thesis on the topics of Unified Modeling Language 2.0 (UML 2.0), the eXtensible Markup Language (XML), the XML Metadata Interchange (XMI), UML design tools, the UML Performance Profile, the Scenario Models such as Use Case Maps (UCM) and Core Scenario Model (CSM), and model transformations from UML to performance models.

2.1 Unified Modeling Language 2.0

Adopted by OMG as UML 1.1 in 1997, the Unified Modeling Language (UML) has become a standard notation for analysis, design and implementation of Object Oriented systems. This formalism arose from the combination of Booch’s, Jacobson’s and Rumbaugh’s methodologies [Booch94][Jacobson+92][Rumbaugh+91] and has almost completely not only these methodologies but also OORAM [Reenskaugh96], Syntropy [Cook+94] and many others. This standardized modeling language is used for the specification, visualization, architecture design, construction, simulation and testing as well as documentation of software systems.

The version 2.0 [UML2.0Super] is used throughout the thesis. This version, even though has been in works for a while, is not yet completely stabilized and hence revisions keep on updating this standard. UML 2.0 standard is described in two documents – UML 2.0 Infrastructure [UML2.0Infra] and UML 2.0 Superstructure [UML2.0Super].

UML Infrastructure: Intended as a comprehensive and a precise specification of the UML’s semantic constructs, this document provides the semantics for all modeling
notations describing the UML Notation Guide. This document also includes the definition of the UML 'abstract syntax' in the form of a set of UML packages. These packages contain a set of meta-classes to define the constructs and their relationships

**UML Superstructure:** The superstructure document is focused on the thirteen official diagrams and several semiofficial diagrams. This document describes the meta-classes, the methods and the relationships between all these diagrams and their respective meta-classes. The diagrams are defined in two categories - Structural and Behavioural. Structural diagrams are used to show the building blocks of a system while behavioural diagrams are used to show how the system responds to requests or otherwise evolve over time.

Even though UML offers a rich set of modeling concepts and notations to meet the needs of a typical software modeling project, users may wish to attach non-semantic information to the models or may wish to have additional features and/or notations beyond those defined in the standards. To accomplish this, UML defines the following built-in extensions:

**Stereotypes:** Stereotypes defines how an existing meta-class can be extended and allows the usage of the platform or domain specific terminology or notation in addition to the ones used for the extended meta-class. They group constraints and tagged values so they could be given descriptive names and be applied to model elements.
Tagged Values: Stereotypes, like classes, may have properties. These properties are referred to as tag definitions and the values of these properties are called tagged values. They are used to attach arbitrary information to any model elements.

Constraints: Constraints places conditions or restrictions on natural language or in OCL for the purpose of declaring some of the semantics of a particular design element.

The above mechanisms help us extend the concepts of UML and introduce some new ones. The tagged values are added to extend the meta-classes to express the quantitative performance information to the UML diagrams. The extension mechanisms are used to define the UML profiles which specializes the UML for different application domains. The one used in this thesis is described in Chapter 2.4.3.

2.2 UML Meta-Model

A four-layer meta-model architecture defines the UML meta-model as illustrated in Figure 2-1. The meta-model architecture is described as follows:

M0: is an instance of a model and is responsible for describing domain-specific information.

M1: is the instance of the meta-model. This is a model of the domain-specific information (ex. in UML)

M2: is the instance of the meta-meta-model. This is the definition of the UML in its meta-classes and is responsible for defining a language specifying model.
M3: is the meta-meta-model. It is responsible for defining a language for specifying a meta-model. This is the definition of the way that UML is defined.

<table>
<thead>
<tr>
<th>M3: Meta-meta-model (MOF)</th>
<th>The primary responsibility of the meta-model layer is to define a language for specifying models</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2: Meta-model (UML Meta-model)</td>
<td></td>
</tr>
<tr>
<td>M1: Model</td>
<td></td>
</tr>
<tr>
<td>M0: user objects (data)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2-1 4-layer Meta-model Architecture

The relationship between these layers is a "instance-of" relationship. The M3 layer, Meta Object Facility (MOF), describes the basic concepts from which specific meta-models are created at the meta(M2) level. One of the meta-model which is the instance-of the MOF meta-meta-model is the UML meta-model. In Chapter 3, more details about the parts of the UML meta-model related to interaction, class and deployment diagrams are discussed.

2.3 XML & XMI

2.3.1 XML

XML is the eXtensible Markup Language. This new standard adopted by World Wide Web Consortium (W3C) is designed to improve the functionality of the Web by providing more flexible and adaptable information identification. This fast becoming
standard for data interchange on the Web was originally designed to complement HTML. XML is a way to work with information in a structured form.

XML is called extensible due to its nature of not having a fixed format like HTML (a single predefined markup language). XML, a ‘meta-language’ can actually and is used for describing other languages which gives you the ability to design your own customized markup language for unlimited different types of documents.

Rather than presentation, XML places the emphasis onto describing the content rather than the presentation. The following are some of the features why XML is widely accepted:

Data Identification: XML describes the type of data rather than how to display the data using tags. These tags may be defined at will. XML allows users to define new tags to indicate the structure of their documents.

Plain Text: Since XML is in plain text, anything from a plain notepad editor to a visual development environment can be used to create and modify XML files. This feature makes it very user friendly and easy to debug the programs as well as for storing small amounts of data. However, due to XML’s front end to the database makes it a great choice to store large amounts of XML data.

Free Style: XML, a free style language, can be processed by various stylesheets to produce outputs in different formats such as PDF, TEX, or some other format.
**Well-Formed:** Because XML document needs to be well-formed due to regular and consistent notation, it is fairly easy to build a program to process XML data. XML is a vendor-neutral standard which gives us the option to choose among several XML parsers depending on the simplification process of the XML data.

### 2.3.2 XMI

XMI, an abbreviation for “XML Metadata Interchange”, is a widely used interchange format for sharing objects using XML. XMI is used to enable easy interchange of meta-data between different modeling tools and meta-data repositories. XMI integrates the following three key industry standards:

- **XML** – eXtensible Markup Language, a W3C standard [W3C]
- **UML** – Unified Modeling Language, an OMG Modeling Standard [UML2.0Super]
- **MOF** – Meta Object Facility, an OMG meta-modeling and meta-data repository standard [MOF]

XMI, an integration of these 3 standards, combines the best of OMG and W3C meta-data and modeling technologies, making it possible for developers to share object models and other meta-data over the internet.

XMI describes various aspects involving the description of objects in XML:

- The foundation based on the representation of objects in terms of XML elements and attributes.
- To relate objects within the same file or across files, XMI includes standard mechanisms to link objects. IDS and UUIDS makes it possible to identify objects and to reference them from other objects

- Using rules described in DTDs and Schemas to validate XMI documents
- Production of XML DTDs/Schemas/Documents from an object model

The XMI describes two sets of rules to provide the open interchange and leverage the capabilities of XML, DTD generation and the document generation. The DTD generation is used to specify the certain rules used for the validation of an XMI document, while the Document Generation actually generates a document based on a given DTD.

XMI DTDs cannot express the semantic meaning appropriately for a model. These are aided by the standards such as UML, MOF, and other OMG standards. For example, the UML-based DTDs allow interchange of object oriented UML models. This gives the ability to interchange at both the data level (XML) as well as the semantic level (OMG).

An information model is used to describe the elements of an XMI DTD. For example, a UML model, plus a fixed set of element declarations that may be used by all XMI documents. This is used as a physical mechanism for the interchanging UML models conforming to the UML meta-model.
2.4 UML Design Tools

2.4.1 Eclipse

IBM's Eclipse Platform is an extensible Integrated Development Environment for anything and nothing in particular. This open source tool is designed to facilitate tool providers with mechanisms and rules to follow which would lead to seamlessly integrated tools. These mechanisms and rules are described by well-defined API interfaces, classes and methods.

Eclipse provides support for the construction of a variety of tools for application development. This tool supports arbitrary content types such as HTML, Java, C, JSP, etc. With its ability to work with GUI and non-GUI based application development environments, Eclipse facilitates seamlessly, under a variety of operating systems, including Windows and Linux, the integration of tools across different content types and tool providers.
As illustrated in Figure 2-2, Eclipse is composed of a workbench, a workspace, help module, team module and various plug-ins. All these reside on top of a Platform Runtime.

2.4.1.1 Plug-ins

Plug-in, the smallest unit of Eclipse Platform, can be developed and delivered separately. These are coded in Java and usually consists of Java code in a JAR library, some read only files, and other resources such as images, web templates, GUI components, etc. A plug-in does not necessarily need to contain code at all, such as the plug-ins which contributes to online help in the form of HTML pages. For interoperability between other plug-ins, each plug-in can declare any number of named
extension points and any number of extensions to one or more extensions points in other plug-ins. The Platform Runtime, on start-up, determines the set of available plug-ins, leading it to build an in-memory plug-in registry. The extension declarations by name are matched with their corresponding extension point declarations, resulting in a plug-in registry which is available through the Platform API.

2.4.1.2 Workspace

The users' workspace hosts regular files used by the user on which the various tools plugged in to the Eclipse Platform operate upon. This consists of top-level projects, which generally are contained within sibling subdirectories of a single workspace but are not necessarily restricted to. These projects, containing the user created files, are directly accessible to the standard programs and tools of the underlying operating systems. Workspace keeps a history mechanism to keep track of the previous changes for any kinds of accidental data loss. It also provides a marker mechanism for annotating resources. This includes annotations such as compiler error messages, to-do list items, bookmarks, search hits, and debugger breakpoints.

2.4.1.3 Workbench and UI Toolkits

The User Interface(UI) of the Eclipse platform is built around a workbench to provide an overall structure with the ability for extensions to the user. The two toolkits composing the workbench API and implementation are Standard Widget Toolkit (SWT) and JFace.
The SWT with its operating system (OS) independent API provides a widget set and a graphics library which is tightly integrated with the native window system. SWT is used for the entire Eclipse Platform UI and any tools that plug into it.

Implemented using SWT, JFace is a user interface toolkit to simplify common user interface programming tasks. This toolkit consists of OS independent APIs and implementations and is designed to work with SWT. Jface includes components from a usual UI toolkit such as image and font registries, dialog, preference, wizard frameworks, and progress reporting for long running operations.

The Eclipse Platform workbench provides the UI characteristics and supplies the structures in which tools interact with the user. With it having such a central function, the workbench is also known as the Eclipse Platform UI as a whole with the main window displaying the Eclipse Platform. This workbench is dependent on the SWT API as well as on the JFace API. The workbench is based on editors, views and perspectives. Editors give the ability to open, edit and save objects. Like a file system based tools, these editors follow an open-save-close lifecycle, but with tight integration in the workbench. Views describe the characteristics of the current object a user is working on within a workbench. Several perspectives can be available in a workbench window, but one of which is visible at any given time. Each perspective comes with its own views and editors, arranged accordingly. Several various types of views and editors can be open at the same time within a perspective.
2.4.1.4 Team Support

A team repository can be associated with a project in the workspace which is to be placed under version and configuration management. The functionality of team repositories can affect the user's workflow in cases such as committing / retrieving files to / from repositories and for comparing different file versions. Since the repository has its own affects on the user's workflow, Eclipse allows each team repository to define its own workflow so if the user is already familiar with an existing workflow with a team repository, they can quickly learn it form within Eclipse.

2.4.1.5 Help

The Help mechanism within Eclipse platform allows tools to define and contribute documentation to one or more online books. A tool usually contributes help style documentation to a user guide and API documentation to a separate programmer guide. Help also incorporates raw content in html files. The facilities arrange the raw content into online books with suitable navigation structures expressed separately in XML files. This separation allows pre-existing html documents to be incorporated as books without the need for rewriting them.

2.5 Rational Software Architect

IBM Rational Software Architect (RSA) is a design and development tool which allows software architects and senior developers within a development team to specify and maintain all aspects of an application's software architecture [RationalRSA]. This powerful and configurable tool leverages model-driven development with the UML for
creating well-architected applications and services. RSA combines all aspects of software design and development into one powerful and easy to use tool. Built on top of the open sourced Eclipse Platform, this tool is influenced by several open industry standards.

RSA is a complete design and a development toolset. This tool includes all the features of the IBM Rational Application Developer for WebSphere Software for building scalable Web services, Java, J2EE and other portable applications. This software has the capacity for the visualization and editing of J2EE, Java and C++ structure and behaviour through the UML diagrams. It has support for UML 2.0 diagrams for analysis and design using Use Case, Class, Sequence, Activity, Composite Structure, State Machine, communication, Component and Deployment Diagrams. This allows the user to capture and communicate all aspects of an application architecture using an industry standard. With the UML 2.0 sequence diagrams, RSA helps understand the flow of Java applications. The diagrams can in turn used for the generation of code from design model. This transformation can be customized to tailor code generation pattern to meet the needs of an organization.

RSA can generate HTML, PDF and XML reports from UML designs. These documents are not only useful in the process of documentation but also the interconnectivity with other tools. RSA can export user created UML 2.0 models in UML2 format which in turn can be imported into the Eclipse Platform (with the UML2 plug-in) for further analysis and transformation. This powerful feature is extremely
useful since this can produce the meta-objects of any given diagrams for further transformations.

2.6 UML2 Plug-in

UML2 project is an open source project based on Eclipse Modeling Framework from IBM. This is a useable implementation of the UML 2.0 meta-model to support the development of the modeling tools. The UML 2 plug-in also provides test cases as a means of validating the specification, as well as to specify the validation rules for defining and enforcing levels of compliance. The plug-in provides a common XMI schema to assist the interchange of semantic models. This feature becomes extremely useful when importing models from various tools such as RSA. Not only, models can be imported, but they can also be created programmatically in Eclipse. The UML2 plug-in provides application program interfaces (API) for the UML 2.0 meta-model it implements. This API can then be used to create user defined models as well as to parse in already created models. This in memory model can then be transformed as per the requirements of the users. Although, this is all a Java based implementation, the files area all stored as a XML format for easier interoperability.

2.7 UML Profile

The UML meta-model is not necessarily designed to facilitate every designer’s requirements. A profile extends meta-classes of an existing meta-model to satisfy various purposes. This extension ability allows the user to tailor a meta-model for
various platforms such as J2EE or .NET as well as various domains (such as real-time or business model domain).

This thesis is about the transformation of Interaction Diagrams of UML 2.0 to Core Scenario Models. For the successful transformation to CSM, these interaction diagrams require performance parameters, which are given with the help of the UML profile for "Schedulability, Performance and Time (SPT)", described next.

2.7.1 UML Schedulability Performance and Time Profile

OMG, realizing the need for quantifiable notion of time and resources in UML, issued a request for proposal (RFP) asking for a UML profile for "Schedulability, Performance and Time." Thereafter, in June 2001, OMG released UML profile for Schedulability, Performance and Time. This profile defines standard paradigms of use for the modeling of time, schedulability, and performance related aspects of real time systems. This profile is meant to assist in the creation of models from which quantitative predictions can be made regarding the mentioned characteristics. This profile also standardizes the communications between the contents, a developer would have designed, and the various analysis and design tools.

The UML profile for "Schedulability, Performance and Time" was released for the previous versions UML 1.X, but not for UML 2.0 itself. The profile in this thesis has been adapted for UML 2.0.
2.8 Performance Modeling Techniques

The Profile for Schedulability, Performance and Time enables the following extensions of UML models:

- Expressing the performance requirements within the design context
- Extending selected elements of UML model by the association of performance related Quality of Service (QoS) characteristics
- Specifying or describing various quantitative execution parameters which can then be used for generating performance models and for performance analysis.
- Displaying in the UML model context performance results obtained from performance analysis.

Therefore, SPT helps to realize two things: estimating performance parameters by using some kinds of modeling techniques and secondly to detect bottlenecks or critical resources a system might have. Amongst the various modeling techniques used in common practice, described below are the three most common ones:

- **Queueing Models**: In various scenarios, this modeling technique defines the system as a set of service centers that execute different workloads. Efficient analytical analysis methods have been developed for different kinds of queueing models. The traditional queueing models have been extended to form, for example, Extended Queueing Networks (EQN) and Layered Queueing Network (LQN).
- **Simulation Models**: This modeling technique may capture more details of the system under study than a queueing model, as it mimics the behaviour of the system.
under study. However, it has the disadvantage that it takes longer to evaluate than analytical models. Using the execution time distribution for the operations of each step, this technique may follow a detailed scenario structure.

- **Discrete State Models:** Discrete State Models encompasses modeling techniques such as Petri Nets. This technique defines tokens to execute the software, again following a detailed scenario structure.

As per the SPE methodology proposed in [Smith90], software performance models are created from the frequently executed scenarios annotated with performance parameters. There are various ways of describing these scenarios in UML, two of which are Interaction Diagrams and Activity Diagrams. This thesis focuses on Interaction Diagrams. Although the SPT Profile is not the focus of this thesis it does play a significant part in the transformation of Interaction Diagrams to Core Scenario Models, which is the objective of this thesis.

### 2.9 Transformations of UML Models into Performance Models

In the software performance engineering field there have been significant efforts to integrate performance analysis into the software development process by using different performance modeling paradigms: queueing networks, Petri nets, stochastic process algebras, simulation. A very good survey of the techniques for deriving performance models from UML models is given in [Balsamo04]. The technique from [Cortellessa+00] follows the SPE methodology very closely, generating the same kind of models as in [Smith90]. The work from [KahkipuroOl] introduces an UML-based notation and framework for describing performance models. In [Lopez04] UML models
are transformed into Petri Nets, but the contention for hardware resources is not considered yet. [Cavenet03] presents a transformation from UML to Stochastic Process Algebra.

The performance research group from Carleton University has implemented so far UML-to-LQN transformations in different ways. A solution described in [Amer01] uses an existing graph-rewriting tool PROGRES [Schurr90]. Another solution presented in [Petriu+02] implements in Java an ad-hoc graph transformation at the UML 1.4 metamodel level; the third solution given in [GU+02] uses Extensible Stylesheet Language Transformation. A two-phase transformation from UML to LQN using XML algebra concepts and implemented in XSLT was presented in [GU+05].

Another on going research project at Carleton University, named PUMA [PUMA05] proposes to use a unified intermediate model named Core Scenario Model (CSM), between different kinds of design specifications (e.g., different UML versions) and different kinds of performance models (e.g., queueing-based, Petri nets, simulation). CSM captures the essence of performance specification as expressed in the SPT Profile, strips away the design details irrelevant to that analysis. The advantage of CSM is evident when we consider adding a new transformation from UML to another performance formalism, as it is much simpler to translate from CSM to a performance model, than directly from UML. The present thesis is a part of the larger project PUMA.
Chapter 3. Input and Output Models

3.1 Input Models

3.1.1 Unified Modeling Language

The Unified Modeling Language (UML) 2.0 specification was adopted in August, 2004, but this is still not the final version of the standard. The UML specifications used in this thesis is [UML2.0Inffa] for the Infrastructure and [UML2.0Super] for the Superstructure.

UML 2.0 is divided into three major parts – structural, behavioural and supplement parts. The structural part discusses the meta-model regarding the static, structural constructs such as classes, components, etc used in various structural diagrams. The behavioural part goes onto describing dynamic, behavioural constructs such as activities, interactions etc used in behavioural diagrams. The supplement part discusses the auxiliary constructs, such as information flows, models, templates, primitives, and profiles, which give UML the ability to be customized for various domains, platforms and methods.

The structural part of UML 2.0 is described by the following diagrams as illustrated in Figure 3.1: Class Diagram, Composite Structure Diagram, Component Diagram, Deployment Diagram, Object Diagram, and Package Diagram. The behavioural aspect is described by Activity Diagram, Use Case Diagram, State Machine Diagram, and Interaction Diagrams. Interaction Diagrams can get further refined into Sequence Diagram, Collaboration Diagram, Interaction Overview Diagram, and Timing Diagram.
The diagrams used in this thesis are: Class Diagram, Deployment Diagram, and Sequence Diagrams.

3.1.1.1 Discussion of Structural Diagram Meta-model

The structural diagram will be discussed very briefly and on a very abstract level as this is not related that significantly to this thesis.

3.1.1.1.1 Discussion of Class Diagram Meta-model

A class is a family of objects which have similar structure, behaviour and meaning. An object can be any item that has identity, structure and behaviour. A class is a kind of a classifier whose features are attributes and operations. Amongst other things, classes own Properties. Instances of the Property meta-class represent the attributes of a class.
3.1.1.1.2 Discussion of Deployment Diagram Meta-model

The deployment diagram describes the layout of software artifacts on the various nodes in a system. An artifact represents a physical piece of information used or produced by a software development process of a system. A node is a computation resource upon which artifacts are deployed on. Nodes are connected through communication paths, which symbolize internetworking of those connected nodes.

![Diagram of Deployment Meta-model]

Figure 3-2 Partial Meta-model for Deployment

3.1.1.2 Discussion of Behavioural Diagram Meta-model

The Sequence Diagram (also called Interaction Diagram) describes the sequence of events happening in a scenario at a given point in time. These are the realizations of use cases. These are often the most useful diagrams for representing scenarios.

In this thesis, from the Behavioral Part of the UML 2.0 Specifications, only the Sequence Diagram is used for the transformation to Core Scenario Model.
3.1.1.2.1 Discussion of Sequence Diagram Meta-model

For a complete discussion of the Sequence Diagram meta-model, please refer to the specifications. This section will briefly highlight and summarize the important elements used in the transformation of the Sequence Diagram meta-model.

The meta-model best can be described by describing the meta-element “Interaction.” An interaction is a behavioural unit which focuses on the passing of the information with Messages between Connectable Elements of the Classifier.

An interaction in a Sequence Diagram is illustrated by a solid outline rectangle. A pentagon in the upper left corner of the rectangle contains the keyword “sd” followed by the interaction name and parameters. An interaction diagram may also have local attributes defined with the same syntax as attributes in general as shown within class symbol compartments. These attribute definitions may appear near the top of the diagram frame or within note symbols at other places in the diagram.

Interaction is an interaction fragment and has interaction fragments. Interaction fragment, a meta-class, is an abstract notion of the most general interaction unit. This abstract class is a specialization of NamedElement. An InteractionFragment is covered by a lifeline.

An event occurrence is a subclass of an interaction fragment and of MessageEnd and hence inherits all of their properties. These EventOccurrences represent the point in
time when an action occurs. The ordering of these basic semantic unit of interactions are
the meanings of Interactions. EventOccurrences are syntactically are at MessageEnds or
the beginning or ending of an ExecutionOccurrence.

Within a lifeline, an ExecutionOccurrence is an instantiation of a unit of
behaviour. Since the execution occurrence really represents some action being executed,
it has some duration which is represent by a start and a finish event occurrence. An
execution occurrence is again a subclass of an InteractionFragment and hence inherits all
the properties of the InteractionFragment.

As seen, Lifeline is contained within an interaction. Lifelines cover
InteractionFragments. They represent an individual participating in an Interaction.
These can only represent one interacting entity. The order of EventOccurrences on a
lifeline matters in the sense that is denotes the ordering in which these EventOccurrences
will occur. Lifeline is again a subclass of a NamedElement.

A message can be contained within only one interaction. It defines a particular
communication between two lifelines of an interaction. For this, it normally is associated
with two EventOccurrences – one sending EventOccurrence and one receiving
EventOccurrence. A message not only specifies the sender and the receiver, but also
specifies the kind of communication given by the dispatching ExecutionOccurrence.
Synchronous messages expect a reply message before the continuation of the execution
while an asynchronous message does not. Asynchronous Messages are denoted by an
open arrowhead. Synchronous messages typically represent method calls and are shown
with a filled arrowhead. The reply messages are denoted by a dashed line.
MessageEnd is an abstract NamedElement which denotes what can happen at the end of a message. MessageEnd is associated with SendMessage, which contains the information about the sending event, as well as ReceiveMessage which contains the information regarding the receiving event.

A Combined Fragment is a subclass of an Interaction Fragment. This meta-class defines an expression of interaction fragments. A combined fragment is described by an interaction operator and the corresponding interaction operands. Some of the values, an interaction operator can have are:

Seq: to represent a sequential combined fragment

Alt: to represent a combined fragment with alternate flows. This alternate flow will be taken if the conditions specified in the InteractionConstraints will be true.

Opt: to represent a combined fragment with an optional flow. This optional flow is taken if the conditions are satisfied in the InteractionConstraints.

Par: to represent a combined fragment with parallel execution. This can illustrate multiple parallel executions occurring at the same time

Loop: to represent a combined fragment to illustrate if there are some actions which are being repeated. The Guard may contain the minimum and the maximum number of repetitions required.

An Interaction Occurrence refers to an Interaction that is defined elsewhere. Instead of showing a selected set of interactions, those selected set of interactions can be
placed in another interaction diagram. An interaction occurrence is then used to refer to another interaction diagram. This is shorthand for representing interactions when an abstract view is sufficient enough and the need to show the detailed is not necessary. This is also quite handy in reusing common interactions amongst different interactions. InteractionOccurrence is a specialization of an InteractionFragment.

![Partial Meta-model for Model](image1)

**Figure 3-3 - Partial Meta-model for Model**

![Partial Meta-model for Interaction Fragment](image2)

**Figure 3-4 - Partial Meta-model for Interaction Fragment**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 3-5 - Partial Meta-model for Lifeline

Figure 3-6 - Partial Meta-model for Message
3.1.2 Adapted Performance Profile

The performance profile described in section 2.4.2 is for UML 1.4. Since this thesis is using input models based on UML 2.0, parts of the performance profile was modified. Table 1-1 illustrates the SPT Stereotypes and their tagged values, and the meta-classes they are applied to in UML 1.4 and UML 2.0. This section discusses the domain concepts of the SPT Profile. Not only that, but it also goes ahead and describes
which meta classes were being used originally in the SPT profile for UML 1.4 and the ones which were are now being used for UML 2.0

3.1.2.1 PAclosedLoad Stereotype

Table 1 - PAclosedLoad Stereotype

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UML 1.4</td>
<td>UML 2.0</td>
</tr>
<tr>
<td>&lt;&lt;PAclosedLoad&gt;&gt;</td>
<td>Message</td>
<td>Message</td>
</tr>
<tr>
<td></td>
<td>Stimulus</td>
<td>Signal</td>
</tr>
<tr>
<td></td>
<td>Action State</td>
<td>Activity</td>
</tr>
<tr>
<td></td>
<td>Subactivity State</td>
<td>Activity</td>
</tr>
<tr>
<td></td>
<td>Action</td>
<td>Action</td>
</tr>
<tr>
<td></td>
<td>Action Execution</td>
<td>Execution</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>Operation</td>
</tr>
<tr>
<td></td>
<td>Method</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>Reception</td>
<td>Reception</td>
</tr>
</tbody>
</table>

For the stereotype PAclosedLoad, some of the meta-classes that required a change was Stimulus, Action State, Subactivity State, Action Execution, and Method. These are the same of the base classes for UML 1.4 which are not applicable to UML 2.0. As shown in table 1-1, the equivalent classes are Signal, Activity, Activity, and Execution.
Occurrence respectively. Method doesn’t have an equivalent meta-class in UML 2.0. The remaining meta-classes remained unchanged.

### 3.1.2.2 PAcontext Stereotype

Table 2 – PAcontext Stereotype

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>UML 1.4</strong></td>
</tr>
<tr>
<td>&lt;&lt;PAcontext&gt;&gt;</td>
<td>Collaboration</td>
</tr>
<tr>
<td>CollaborationInstanceSet</td>
<td></td>
</tr>
<tr>
<td>ActivityGraph</td>
<td>Activity</td>
</tr>
</tbody>
</table>

A performance context describes one or more scenarios. These scenarios are used to express dynamic situations involving a specific set of resources. For the stereotype PAcontext, which models a performance context, the Collaboration meta-class remain untouched and is still a base class in UML 2.0. However, for collaborationInstanceSet, there is no equivalent meta-class, and the activity graph meta class has activity as its equivalent meta-class in UML 2.0.
3.1.2.3 PAhost Stereotype

Table 3 – PAhost Stereotype

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;PAhost&gt;&gt;</td>
<td></td>
<td>PAutilization</td>
</tr>
<tr>
<td></td>
<td>Classifier</td>
<td>PAschdPolicy</td>
</tr>
<tr>
<td></td>
<td>Node</td>
<td>PArate</td>
</tr>
<tr>
<td></td>
<td>ClassifierRole</td>
<td>PActxtSwt</td>
</tr>
<tr>
<td></td>
<td>Instance</td>
<td>PAprioRange</td>
</tr>
<tr>
<td></td>
<td>Partition</td>
<td>PApreemptable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAthroughput</td>
</tr>
</tbody>
</table>

PAHost annotates the processing resource utilized by a component (instance) for its execution. As described in Table 1-3, this stereotype is applied to the classifier, node and partition meta-class which hasn’t changed in UML 2.0. ConnectableElement generalizes the concept of classifier role from UML 1.X, while Instance meta-class’s equivalent is InstanceValue in UML 2.0.
3.1.2.4 PAopenLoad Stereotype

Table 4 – PAopenLoad Stereotype

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>UML 1.4</th>
<th>UML 2.0</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;PAopenLoad&gt;&gt;</td>
<td>Message</td>
<td>Message</td>
<td></td>
<td>PArespTime</td>
</tr>
<tr>
<td></td>
<td>Stimulus</td>
<td>Signal</td>
<td></td>
<td>PApriority</td>
</tr>
<tr>
<td></td>
<td>Action State</td>
<td>Activity</td>
<td></td>
<td>PApopulation</td>
</tr>
<tr>
<td></td>
<td>Subactivity State</td>
<td>Activity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Action</td>
<td>Action</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ActionExecution</td>
<td>Execution Occurrence</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>Operation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Method</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reception</td>
<td>Reception</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PAopenLoad stereotype is used to model a stream of requests which arrive at a given rate in some prearranged pattern. As the Table 1-4 illustrates, meta classes Message, Action, Operation, and Reception are still applicable in UML 2.0. In UML 1.4, the meta classes Stimulus, and Action Execution map to UML 2.0’s meta classes Signal and Execution occurrence respectively. Action State and Subactivity State from UML 1.4, both map to Activity in UML 2.0.
3.1.2.5 PAcontext Stereotype

Table 5 – PAcontext Stereotype

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;PAcontext&gt;&gt;</td>
<td>Classifier</td>
<td>PAutilization</td>
</tr>
<tr>
<td>Node</td>
<td>Node</td>
<td></td>
</tr>
<tr>
<td>ClassifierRole</td>
<td>ConnectableElement</td>
<td></td>
</tr>
<tr>
<td>Instance</td>
<td>Instance</td>
<td></td>
</tr>
<tr>
<td>Partition</td>
<td>ActivityPartition</td>
<td></td>
</tr>
</tbody>
</table>

This stereotype models a passive resource. As illustrated in Table 1-5, not much has changed from UML 1.4 to UML 2.0 for the meta-classes this stereotype is applied upon. Only the Partition meta-class for the purposes required for this stereotype has changed to ActivityPartition, a meta-class in UML 2.0 and Connectable Element, an element in UML 2.0, generalizes the concept of Classifier Role from UML 1.X.
### 3.1.2.6 PAstep Stereotype

Table 6 - PAstep Stereotype

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;PAstep&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td>Message</td>
</tr>
<tr>
<td></td>
<td>Stimulus</td>
<td>Signal</td>
</tr>
<tr>
<td></td>
<td>Action State</td>
<td>Activity</td>
</tr>
<tr>
<td></td>
<td>SubactivityState</td>
<td>Activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAdemand</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAreptime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAprob</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PArep</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAdelay</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAextOp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAinterval</td>
</tr>
</tbody>
</table>

An execution of some action is represented by a step. The PAstep stereotype, as described in Table 1-6, is applied to Message in both UML 1.4 and UML 2.0. However, this stereotype was being applied to Stimulus, Action State and SubActivity State in UML 1.4 which is now have equivalent meta classes in UML 2.0 such as Signal for Stimulus and Activity for Action State and Subactivity State.
3.1.3 Use of Adapted Performance Profile in Rational Software Architect

Rational Software Architect provides a very friendly way to implement and apply the performance profile to models. Profiles can be created in the tool and then can be applied to a model. To apply a given profile, the profile is added to the model itself as shown in the Figure 3-9. Add Profile button leads to another window where a profile can be selected and added as shown in Figure 3-10.

Figure 3-9 – Screenshot of RSA - Profiles
Once the profile has been added, the profile stereotypes needs to be applied to the desired meta-elements. As seen in the Figure 3.11, when a meta-element is selected, its properties windows in the bottom give an option to add stereotypes. As seen, the add stereotype button brings up a dialog box which gives the option of applying the appropriate stereotypes.
Figure 3-11 – Screenshot of RSA – Applying Stereotype
A screen capture is shown in Figure 3.12 for RSA. As can be seen, stereotypes properties tab is shown when a meta-element is selected in the model viewer. As seen, the applied stereotypes are listed for that meta-element, in this case Execution Occurrence. For each stereotype, the attributes are listed below with a “Value” column for assignment of the values for each attribute.

Once the model has the profile applied, the stereotypes are available for appropriate elements. These stereotypes’s attributes for each applied element is visible and the values can be set accordingly then. The model is saved then retaining all this information for export and future use.
3.2 Output Models

3.2.1 Core Scenario Model (CSM)

Core Scenario Models (CSM)[Petriu+05] are closely based on the domain model of the UML Profile for Schedulability, Performance and Time. The CSM metal-model starts with a CSM meta-element as the root node and everything is contained within the CSM meta-element. CSM meta-model has two parts – behavioural and structural.

3.2.1.1 Behavioural Part of CSM

CSM meta-class contains Scenarios meta-class. Scenarios represent behaviour of one kind of a system response. A scenario has an ordered sequence of steps each of which are related with each other by some kind of a PathConnection. A Step is loosely defined as an execution of an action.

CSM provides explicit flow of events described by path connections. A Path Connection can be a Sequence, Branch, Merge, Fork, Join, Start and End. Branch and merge meta-class are of nature of OR-fork and OR-join respectively while fork and join meta-classes are of nature of AND-fork and AND-join respectively. The attributes of the successor steps include the probabilities for an OR-fork. Start meta-class represents the start of a complete scenario while the End meta-class represents a possible ending to a scenario. Each path connection can take a different number of predecessor and successor steps.

Each scenario has a workload which defines the intensity of its execution. The workload can be an “open workload” with arrival coming from the environment or they
can be a "closed workload" where there is a fixed number of potential arrivals which are either already in the system or are waiting to arrive again. This workload information is described in the meta-class Start.

A step has a Refinement meta-class. A Refinement meta-class is used to describe a sub-scenario for the possible purposes of abstraction. Refinements hold a reference to a scenario for which it is describing.

### 3.2.1.2 Structural Part of CSM

General Resource is the top meta-class in the structural side of CSM. This is contained within the CSM meta-class. Resources get further specialized into Passive and Active Resources. Active resources, such as a user, spontaneously generate events while passive resources respond to requests. Both of these types of resources can either be protected (in which case a client gets exclusive rights to one or more of units of resources) or they can be unprotected (in which case they can be shared without control). Active Resources can further be specialized into External Operation and ProcessingResource. ProcessingResources are devices which can actively process resource themselves such as CPU. Meanwhile, ExternalOperations represent external operations requested by the system but the external operations are executed on their own external processing resources. Every step has a host processing resource or CPU which actually executes the step. Passive Resources can be further specialized into Components which described resources such as buffers, tasks or semaphores.
Figure 3-13 - CSM Meta-model
3.3 UML 2.0 Graphical Notation and the Underlying Representation

The previous sections describe the UML 2 meta-classes and how they are related and all their relevant information pertaining to this thesis. To fully understand these concepts, a discussion on the realization of these meta-classes is presented below.

3.3.1 Object Model – UML 2.0 Behaviour

3.3.1.1 Simple Interaction

![Graphical Notation of a Simple Interaction](image)

Figure 3-14 – Graphical Notation of a Simple Interaction

Figure 3.14 illustrates a simple interaction diagram. This describes a synchronous message being passed and a reply to it (login:Message and replyLogin:Message) and also an asynchronous message (notify:Message). The internal data structure created by a
UML tool for the interaction diagram shown in Figure 3.14 contains a meta object initialize which is an instance of meta class Interaction. Initialize object contains client, and server, both objects of lifeline meta class. The login, replyLogin and notify, all three are objects of meta class Message. However, notify object is an asynchronous message and hence requires no reply. Initialize object of course contains all the event occurrences and execution occurrences described in this notation. All this containment of objects is illustrated in Figure 3.15.

![Diagram](image)

**Figure 3-15 – Simple Interaction Object Containment**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
3.3.1.2 Interaction Occurrence

Figure 3-16 – Graphical notation of Interaction Occurrence

Figure 3.16 illustrates an interaction diagram with an interaction occurrence. The above notation renders the object modifyUserList. modifyUserList has client1, server1 and retrieveUserList. This retrieveUserList, which is an object of Interaction Occurrence, refers another interaction named RetrieveUserList. The containment is shown in Figure 3.17.
3.3.1.3 Combined Fragment – Optional

The Figure 3.18 involves a simple interaction between client and server as well as shows an alternative combined fragment. The root object of this notation is retrieveUserList. retrieveUserList contains client1, server1, ev1 (ev = event occurrence), ev2, ev3, ev4, ev5, ev6, msg1, msg2, msg3, msg4, ex1 (ex = executionOccurrence), ex2 and combFrag1. The interactionOperator for this combined fragment, combFrag1, is opt and thus describing the combined fragment as an optional combined fragment with
guard1 as its guarding condition to enter this combined fragment. Now combFrag1 contains an operand which is labeled as oper1. Oper1 contains all the interaction fragments such as oplev1, oplev2, oplev3, oplev4, oplev5, oplex1, and oplex2. This is all illustrated in Figure 3.19

Figure 3-19 – Graphical Notation of Combined Fragment
3.4 Programmatic Generation of UML Models

During the initial phase of this thesis, there were no tools to support drawing UML 2.0 models by using the standard graphical notation. IBM’s Eclipse was used in this area with the plug-in called “UML2 Plug-in”. This led to programmatic creation of the UML 2.0 models including the class diagrams, deployment, and interaction diagrams. All the metaobjects were defined programmatically in JAVA. Various relationships between different objects were realized and implemented. This all was just the basis of the input model and not the transformation itself. Further on, the UML Profile for Schedulability, Performance and Time was also implemented and applied to the input models. The programmatic generated models annotated with the performance parameters were then used as the input of the transformation engine.

The programmatic generation involves a Factory Class. This factory class is used for the generation of the CSM elements. So to create any CSM elements, methods within this Factory class are invoked as shown in Figure 3-20.

```
Scenario scenariol = Csm23Factory.eINSTANCE.createScenario();
scenariol.setId(CSMObjectsIds.getScenarioId());
CSM.getCSMElement().add(scenariol);
```

Figure 3-20 - Example Code
As shown in the example code, scenarios and every other CSM element is created using the Factory class and invoking the appropriate methods. The id of a scenario elements is retrieved by an id generator and is set. Then this scenario element is added to the CSM element itself.

As noticed in this example above, the code not only requires a very thorough knowledge of the CSM meta-model, but also the UML 2.0 meta-model. Deep understanding of the meta-models is required to generate all the model elements that should be created by a UML editor. This deep understanding of the meta models is a very time consuming as well as a tedious job.

The UML2 plug-in is a helpful program to create models if no other UML 2.0 editor is available. However, the whole programmatic generation is very error prone, as it lacks features for checking the completeness or correctness within a given model. Models can be quite incomplete or incorrect, as objects may be missing or incorrect associations may be created and no errors would be generated.
Chapter 4. Transformation from UML 2.0 to CSM

4.1 Introduction

This chapter describes the proposed transformation from UML 2.0 models containing interaction and deployment diagrams to CSM.

4.2 Overview of the Transformation Approach

The transformation is composed of a set of transformation rules (described in sections 4.3). A transformation algorithm invokes these rules (described in section 4.6) to complete the whole transformation from UML 2.0 to CSM. An assumption is made that the correct inputs are created according to the cases described in section 4.3 and 4.4. If the input does not follow the cases that were considered in these conditions, the transformation does not guarantee the correctness of the transformation.

The UML meta-model has been expanded by applying the adapted Performance Profile. Stereotypes are applied to some of the meta-elements in UML 2.0 meta-model. With the profile at play here, those meta-elements now have more attributes than originally defined in UML 2.0 Super Structure Specification [UML2.0Super]. These new attributes are used to add useful information in the model. Not all of the stereotypes and attributes are going to be discussed as that could be quite extensive. As described in the Table 1-7, the first column gives the UML 2.0 meta-model element that is being examined. The stereotype and stereotype attributes columns list some of the stereotypes that are being applied to this UML 2.0 element mentioned in the first column. The CSM Element to which a UML 2.0 Element maps to is also given. The last column, CSM
Element Attributes, lists the attributes that are being set for the given CSM Element. The values of these attributes are obtained from the values of the attributes of the stereotypes which have been applied to the particular UML 2.0 meta-model element.

### 4.2.1 Mapping from UML 2.0 to CSM

Table 7a – Mapping of Stereotypes to various CSM Elements

<table>
<thead>
<tr>
<th>SPT Stereotype</th>
<th>UML 2.0 Elements</th>
<th>CSM Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAStep</td>
<td>Execution Occurrence</td>
<td>Step</td>
</tr>
<tr>
<td></td>
<td>Combined Fragment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction Occurrence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Signal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Activity</td>
<td></td>
</tr>
<tr>
<td>PAResource</td>
<td>Classifier</td>
<td>Resource</td>
</tr>
<tr>
<td></td>
<td>Node</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConnectableElement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ActivityPartition</td>
<td></td>
</tr>
<tr>
<td>PAHost</td>
<td>Node</td>
<td>Processing Resource</td>
</tr>
<tr>
<td></td>
<td>Classifier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConnectableElement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>InstanceValue</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Partition</td>
<td></td>
</tr>
</tbody>
</table>
The mapping of UML 2.0 to CSM is based on the SPT stereotypes. The SPT profile is by definition very close to CSM and hence it is reflected when we compare the SPT stereotypes and the CSM Elements. Most of the relevant stereotypes of SPT are covered in the mapping tables in Table 7a and Table 7b. We make the assumption that the relevant UML elements are stereotyped by the designers. If a stereotype is missing, the algorithm will skip the element and continue on with the transformation. However, the algorithm does not identify all the possible errors in the input.

PAStep stereotype is applied to the UML 2.0 meta-model element Execution Occurrence. This means that the Execution Occurrence has all the attributes which the PAStep stereotype did. The CSM element Step has the attributes host Demand,
Table 8a – Mapping of Attributes from Performance Profile to CSM

<table>
<thead>
<tr>
<th>Profile Attribute</th>
<th>CSM Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAprobability</td>
<td>probability</td>
</tr>
<tr>
<td>PArepetition</td>
<td>Rep_count</td>
</tr>
<tr>
<td>PADemand</td>
<td>host_demand</td>
</tr>
<tr>
<td>PADelay</td>
<td>delay</td>
</tr>
<tr>
<td>PAinterval</td>
<td>interval</td>
</tr>
<tr>
<td>PArespTime</td>
<td>responseTime</td>
</tr>
<tr>
<td>PApriority</td>
<td>Priority</td>
</tr>
<tr>
<td>PAoccurrence</td>
<td>maxOccurs / minOccurrs</td>
</tr>
</tbody>
</table>

<< PAstep >> => Step

<< PAopenLoad / PAClosedLoad >> => Step

Table 8b – Mapping of Attributes from Performance Profile to CSM (continues Table 8a)

<table>
<thead>
<tr>
<th>Profile Attribute</th>
<th>CSM Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAresource</td>
<td>Resource</td>
</tr>
<tr>
<td>PAHost</td>
<td>Resource</td>
</tr>
<tr>
<td>PAutilization</td>
<td>utilization</td>
</tr>
<tr>
<td>PASchedulingPolicy</td>
<td>sched_policy</td>
</tr>
<tr>
<td>PAthroughput</td>
<td>Throughput</td>
</tr>
<tr>
<td>PARate</td>
<td>1 / OpTime</td>
</tr>
</tbody>
</table>
Table 9 – Mapping of Attributes from UML 2.0 to CSM

<table>
<thead>
<tr>
<th>Special Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UML 2.0</strong></td>
</tr>
<tr>
<td>InteractionConstraint</td>
</tr>
<tr>
<td>InteractionConstraint</td>
</tr>
</tbody>
</table>

probability and repCount. The value of the hostDemand is obtained from PADemand, value of the probability is obtained from PAProb, and the value of the repCount is obtained from PAREp. These PADemand, PAProb and repCounts are all attributes of the stereotype PAStep which has been applied to the execution occurrence and hence are the attributes of the execution occurrence itself.

Similarly, PAResource stereotype is applied to a Node, Classifier and a ConnectableElement. These UML 2.0 elements map to a general element in CSM called Resource. The capacity variable of that CSM Resource element gets assigned the value of the PACapacity attribute from the PAResource stereotype.

PAHost stereotype is applied to the UML 2.0 element Node, Classifier and a ConnectableElement. These elements can map to a CSM element called ProcessingResource. Inverse of the PARate value is assigned to the variable OptTime attribute of the ProcessingResource.
PAOpenLoad stereotype is applied to the Execution Occurrence element in UML 2.0. The equivalent of Execution Occurrence in CSM is a Step. Basically, that means that the attributes of the Steps are set by the attributes of this stereotype. PArespTime maps to responseTime attribute of a Step. PApriority maps to the priority attribute of a Step and PApopulation maps to the population attribute of a Step.

Attributes of PAClosedLoad stereotypes are the same as PAOpenLoad with the addition of PAExternalDelay. PAClosedLoad attributes map similarly to those of the PAOpenLoad. PArespTime maps to responseTime attribute of a Step, PApriority maps to the priority attribute of a Step and PApopulation maps to the population attribute of a Step. PAExternalDelay attribute maps to the externalDelay attribute of the CSM element Step.

Most of the values of the attributes of CSM's elements are being obtained from the SPT profile. However, there are some values being obtained from UML 2.0 elements. As shown in Table 10, the repetition count for the CSM element Step is obtaining its value from the maxInt / minInt attributes of the InteractionConstraint. These values are used especially when a loop combined fragment is being transformed. Similarly, the pre / post condition CSM elements are obtaining their values from the UML 2.0 InteractionConstraint. This is seen when the optional / alternative combined fragment is being transformed in CSM.
4.3 Structural Aspect

4.3.1 Deployment Diagram

![Deployment diagram relationship](image)

In an interaction diagram, a lifeline represents a property, which corresponds to a component in the CSM meta class diagram. A property, that represents a lifeline, has an element "type" in UML 2.0. As can be seen in Figure 4.1, in a deployment diagram, an artifact is deployed on a node. A node yields a processing resource. An artifact manifests an instance specification. That instance specification has a type. If that type element is the same type element as the one which property has and also if name of the property is the same name as the instance specification, then that component which was yielded from the property of the lifeline is hosted on the processing resource yielded by the node.

In Figure 4.2, the client and the server both transforms into CSM components. However, only the server gets a processing resource, WebServer, on which the server is hosted on. This is because in the deployment diagram, there is actually an instance specification with the name server which is manifested by the artifact serverFiles which is deployed on the WebServer node.
Figure 4-2 – Structural Transformation
4.4 Behavioural Aspect

4.4.1 Transformation Rules

The atomic transformation rules are described in the following sections. Since these are atomic transformations, they may occur in the middle of a scenario.

4.4.1.1 Beginning of a Scenario

A scenario begins with an execution occurrence on the first lifeline. This execution occurrence is annotated with performance information regarding this scenario such as population, the interval for the user entry in the scenario as well if the scenario has an openLoad or a closedLoad.

Figure 4-3 – Beginning of Scenario Transformation
This simple initial setup of each scenario gets transformed into the following CSM elements: Start, ResourceAcquire acquiring Component S on the CPU Processing Resource, and Initial Step (labeled here as x1). The attributes of the element Initial Step, x1, are assigned all the values of attributes that are used to annotate the execution occurrence.

As this describes the beginning of a scenario, the transformation here is just described for the initial execution occurrence.

4.4.1.2 Ending of a Scenario

<table>
<thead>
<tr>
<th>UML 2.0</th>
<th>CSM</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Figure 4-4 – Ending of Scenario Transformation

This represents the end of a scenario. Basically a last message or the last reply of a message denotes the end of a scenario.

When the last message or the last reply of a message is found, that means that there is nothing more on at least this thread of the executions. Hence, the CSM....
element End is created to reflect the end of the scenario. Of course, there is no telling what CSM elements would have been created before this CSM End element as that would depend on the interactions contained in the UML scenario.

4.4.1.3 Simple Synchronous Message

![Diagram of Simple Synchronous Message Transformation]

Figure 4-5 – Simple Synchronous Message Transformation

A synchronous message is passed between two lifelines. This involves, amongst other UML elements, a message and couple of execution occurrences as labeled as x1 and x2.

The transformation for this synchronous simple message passing between the two lifelines gets very easily transformed into CSM. The execution occurrences labeled as x1
and x2 gets transformed into Step x1 and Step x2. We assume that the component represented by lifeline s has already been acquired previously. Hence, there is no ResourceAcquire before Step x1. However, for Step x2, a ResourceAcquire on Component t is shown with the Processing Resource CPU. The Step x2 is shown to represent the execution occurrence on the lifeline t in the Interaction Diagram. There are no more executions on that lifeline, and the reply of the message is shown. This will cause a ResourceRelease as there is no more use of holding on to the Component t. Then further execution will continue.

The execution occurrence, x1, is not split during the transformation before and after the invocation of the m1. This is because the CSM is very close to SPT profile and the transformation is heavily based on the elements which are stereotyped. The execution occurrence, in its completeness, can and is stereotype by the SPT. However, there is no way of stereotyping partial elements and hence the complete execution occurrence gets stereotyped and gets transformed into CSM. The other UML 2.0 elements involved that could potentially get stereotyped would be event occurrences but since event occurrences, by definition, do not have a time for its execution, event occurrences would not meet the requirements for the stereotyping. If the designers require that x1 should be split to model two different execution times, one for before and one for after message, then nested execution occurrences should be used as shown in Figure 4-6.
4.4.1.4 Simple Asynchronous Message

An asynchronous message (m1) is shown followed by another message (m2). The m1 message causes some execution to happen on lifeline t while the m2 causes some execution to happen on the lifeline s. There is no waiting for reply for m1. A second thread starts off and executes the second message, m2, concurrently. This starts off the execution x2 which is executing in parallel with x3.

There is an assumption made that this is part of an existing scenario and the execution occurrence x1 is really coming from before. Hence, we assume that the lifeline representing component s is already acquired. The first execution occurrence x1 yields...

Figure 4-6 – Simple Asynchronous Message Transformation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Step x1. The asynchronous nature of message m1 causes a Fork element to be created. After the Fork element, there are two paths to represent each thread. Step x2 represents the execution occurrence represented by the x2 execution occurrence (associated with message msg2). The other thread associated with message msg1 requires that the resource representing lifeline s will be released. For this, there is Resource Released created for this resource. Due to the execution occurrence x3, a ResourceAcquire for the component t is created followed by Step x3.

4.4.1.5 Alternative Combined Fragment

Figure 4-7a – Alternative Combined Fragment Transformation
These figures illustrate a segment of a scenario when a choice is given. The alternative combined fragment handles this kind of a section where the number of
operands would be equal to the number of choices, in this case 2 operands. Each operand is governed by a guard condition which will decide which operand to be followed.

The alternate combined fragment yields a branch. Since there are two operands in this alternative combined fragment, there are two complex Steps branched off, operand1 and operand2. Then these two branches are merged in again by a Merge element.

Each complex step refers to a scenario which is looked more in detail. The operand starts off with a Start element. Component t resource is acquired. Then for operand 1, Step x2 is created while for operand 2, Step x3 is created. Resources are released which are represented by ResourceRelease elements and finally an End element to show that this scenario has come to an end.
### 4.4.1.6 Parallel Combined Fragment

<table>
<thead>
<tr>
<th>UML 2.0</th>
<th>CSM</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Component $s$ was acquired before $x1$

Figure 4-8a – Parallel Combined Fragment Transformation
<table>
<thead>
<tr>
<th>UML 2.0</th>
<th>CSM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>operand1</strong></td>
<td><strong>Operand 2</strong></td>
</tr>
<tr>
<td>Start:operand1</td>
<td>Start:operand2</td>
</tr>
<tr>
<td>ResAcq::</td>
<td>ResAcq::</td>
</tr>
<tr>
<td>Step::x2</td>
<td>Step::x3</td>
</tr>
<tr>
<td>ResRel::</td>
<td>ResRel::</td>
</tr>
<tr>
<td>End</td>
<td>End</td>
</tr>
</tbody>
</table>

Figure 4-8b – Parallel Combined Fragment Transformation (continued)
This figure illustrates a segment of a scenario when there needs to be multiple scenarios running in parallel. The parallel combined fragment is very similar to the alternative combined fragment. The number of operands equals to the number of “scenarios” which are required to be run in parallel.

The parallel combined fragment yields a fork. Since there are two operands in this parallel combined fragment, there are two complex Steps forked off, operand1 and operand2. Then these two branches are joined in again by a Join element.

Each complex step refers to a scenario which is looked more in detail. The operand starts off with a Start element. Component t resource is acquired. Then for operand 1, Step x2 is created while for operand 2, Step x3 is created. Resources are released which are represented by ResourceRelease elements and finally an End element to show that this scenario has come to an end.
### 4.4.1.7 Optional Combined Fragment

<table>
<thead>
<tr>
<th>UML 2.0</th>
<th>CSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Diagram]</td>
<td>Component s was acquired before x1</td>
</tr>
<tr>
<td>s:CA</td>
<td>Step::x1</td>
</tr>
<tr>
<td>t:CB</td>
<td>Step::operand1</td>
</tr>
</tbody>
</table>

**Figure 4-9 – Optional Combined Fragment Transformation**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
An optional combined fragment is very much like the alternative combined
fragment except this one has only one operand and does not give more than one optional
path to take.

This combined fragment yields a Step x1 and then a complex Step Operand1. The
precondition of the Operand1 Step will have the guard condition of the Operand1
operand of the combined fragment. The operand1 scenario has the usual Start CSM
element, ResourceAcquire CSM element on the component t, Step x2 to represent the
execution occurrence x2, then a ResourceRelease CSM element on the component t and
finally the End CSM element.

Again, since this is considered an atomic transformation, this transformation can
be plugged in anywhere given the appropriate assumptions are made.
4.4.1.8 Loop Combined Fragment

Figure 4-10 – Loop Combined Fragment Transformation
A loop combined fragment is very much like the optional combined fragment except this is used for the repetition of a given set of executions.

This combined fragment yields a Step x1 and then a complex Step Operand1. The number of repetition in the Operand1 Step will have the number of repetition as specified in the loop combined fragment’s guard.

Operand1 complex step refers to a scenario which is suppose to be a representation of the operand 1 of the loop combined fragment. The operand1 scenario has the usual Start CSM element, ResourceAcquire CSM element on the component t, Step x2 to represent the execution occurrence x2, then a ResourceRelease CSM element on the component t and finally the End CSM element.

Again, since this is considered an atomic transformation, this transformation can be plugged in anywhere given the appropriate assumptions are made.

4.4.2 Transformation Rules Example

4.4.2.1 Simple Interaction

A plain simple interaction with a couple of lifelines and a synchronous message exchange interaction between the two lifelines are represented in Figure 4.1:
Figure 4-11 – Simple Scenario

The CSM equivalent model is shown on the right hand side of the figure.

This simple example invokes the following transformation rules:

a) generating the CSM *Start* element;

b) transforming the first execution occurrence by the Client into an unnamed *Step* preceded by *R_Acquire: client* and ended by *R_Release: client*;

c) transforming the synchronous call into *Step: login* preceded by *R_Acquire: server* and ended by *R_Release: server*;

d) generating the CSM *End* step.

The Initialize scenario first produces, the CSM meta element *Start* named Initialize. Since the first execution occurrence is on the client lifeline, CSM acquires the
resource client and attaches an initial step with it. The login request is being invoked at
the server side so the CSM acquires the server resource and attaches a login Step. This is
a synchronous message which the CSM transformation engine recognizes and hence
doesn’t create any steps relating to the return of the login request. As there are ends on
the execution occurrence on the server life line and then on the client life line, CSM
transformation engine reflects these ends by creating Resource Releases for server and
client respectively. Since there is nothing more in this interaction, a CSM meta element
End is attached.

4.4.2.2 Simple Interaction with Asynchronous Messages

![Figure 4-12 - Asynchronous Message](image)

Figure 4-12 – Asynchronous Message
This simple example with asynchronous message invokes the following transformation rules:

a) generating the CSM Start element;

b) transforming the first execution occurrence by the Client into an unnamed Step preceded by \( R_{\text{Acquire}}: \text{client} \) and ended by \( R_{\text{Release}}: \text{client} \);

c) transforming the synchronous call into Step: login preceded by \( R_{\text{Acquire}}: \text{server} \) and ended by \( R_{\text{Release}}: \text{server} \);

c) transforming the asynchronous call into a Fork with two threads. One thread of the Fork has a \( R_{\text{Release}}: \text{client} \) followed by a Step: notify preceded by \( R_{\text{Acquire}}: \text{server} \) and ended by \( R_{\text{Release}}: \text{server} \), while the second thread just releases the client by having a \( R_{\text{Release}}: \text{client} \)

d) generating the CSM End step.

The Notify interaction in Figure 4.12 is very similar to Figure 4.11 except at the end, there is actually an asynchronous message notify going from client to server. As expected, the CSM equivalent of this interaction produces a similar CSM as of Figure 4.11 to the point just before the asynchronous message. The asynchronous message yields a Fork and two threads thereafter. Thread 1 obviously represents the sequence of events which occurs on the invoking lifeline of the asynchronous message while the thread 2 represents the sequence of events which occurs on the lifeline(i.e. client) where the asynchronous message is being invoked(i.e. server). After the fork, thread 1 just
releases the resource client and comes to an end. However, thread 2 acquires the server resource, performs the step notify on this resource, releases the resource and ends its execution.

4.4.2.3 Interaction Occurrence

This example with interaction occurrence invokes the following transformation rules:

a) generating the CSM Start element;

b) transforming the first execution occurrence by the Client into an unnamed Step preceded by \textit{R\_Acquire: client} and ended by \textit{R\_Release:client};

c) transforming the interaction occurrence into a complex step,
d) generating the CSM End step.

With ModifyUserList scenario in Figure 4.13, this interaction has only an interaction occurrence which is referring to a RetrieveUserList scenario.

The transformation from a UML 2.0 Interaction to CSM Scenario is shown on the right side of the Figure 4.13. It starts off like the previous CSM models with a Start element, followed by a ResourceAcquire on component client1. There is an initial Step created for the first execution occurrence. Again the attributes of execution occurrence and the attributes of the stereotypes applied to the execution occurrence are assigned to the attributes of the original step. The interaction occurrence, RetrieveUserList, yields the complex Step RetrieveUserList. This complex step refers to a scenario which will have the equivalent of the RetrieveUserList. Server1 resource will be released then and this scenario will come to an end.
4.4.2.4 Simple Interaction with Optional Combined Fragment

<table>
<thead>
<tr>
<th>UML 2.0</th>
<th>CSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>a)</td>
<td>b)</td>
</tr>
</tbody>
</table>

**UML 2.0**

- **RetrieveUserList**
  - **Component:** client:client
  - **Step:** 0.1: login
    - «return»
  - **Component:** server:server
  - **Step:** 0.2: login
    - «return»

**CSM**

- **Start:** RetrieveUserList
- **Step:** R_Acquire: client
- **Step:** login
- **Step:** ProfileDoesNotExistLocally
- **Step:** ProfileDoesNotExistLocally_STEP_ID_21261
- **Step:** RetrieveProfile
- **Step:** R_Acquire: server
- **Step:** login
- **Step:** R_Release: server
- **Step:** R_Release: client
- **End**

**c)**

- **Start:** ProfileDoesNotExistLocally_STEP_ID_21261
- **Step:** R_Acquire: server
- **Step:** RetrieveProfile
- **Step:** R_Release: server
- **End**

Figure 4-14 – Optional Combined Fragment
This simple example with optional combined fragment invokes the following transformation rules:

a) generating the CSM Start element;

b) transforming the first execution occurrence by the Client into an unnamed Step preceded by $R_{\text{Acquire}}: \text{client}$ and ended by $R_{\text{Release}}:\text{client}$;

c) transforming the synchronous call into Step:login preceded by $R_{\text{Acquire}}:\text{server}$ and ended by $R_{\text{Release}}:\text{server}$;

c) transforming the optional combined fragment into a complex step. This complex step generates a complete new scenario. This new scenario has a Start element, Step:retrieveProfile preceded by $R_{\text{Acquire}}:\text{server}$ and ended by $R_{\text{Release}}:\text{server}$; and finally a CSM End

d) generating the CSM End step.

Retrieve User List Interaction in Figure 4.14 shows an optional combined fragment after the initial login messages being passed between client and server. The initial messages, as before, yields similar CSM as before such as the Start, Resource Acquire on Client, Initial Step, Resource Acquire on Server, the login request being invoked on the Server and finally Resource Release on Server. We do not release the resource on Client as there is an optional combined fragment on the execution occurrence on the Client.
For the optional combined fragment, there is a guard condition if the ProfileDoesNotExistLocally. This optional Combined Fragment yields a complex Step. This complex step has a refinement which is linked to a new scenario which is not shown in the CSM viewer in Figure 4.14. This new scenario (Figure 4.14 c) represents all the interaction within the optional combined fragment. As seen in Figure 4.14, the interaction within the combined fragment yields the new scenario in CSM. Server Resource is acquired again, step is created to facilitate retrieveProfile request to the server and then the resource is released. There is obviously an End to this new scenario.
4.4.2.5 Simple Interaction with Parallel Combined Fragment

**UML 2.0**

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>login</td>
</tr>
<tr>
<td>«return»</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>login</td>
</tr>
<tr>
<td>«return»</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SearchPic</td>
</tr>
<tr>
<td>«return»</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SearchPic</td>
</tr>
<tr>
<td>«return»</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SearchFile</td>
</tr>
<tr>
<td>«return»</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SearchFile</td>
</tr>
</tbody>
</table>

**CSM**

- **Start:** Advanced Initialization
- **Step:**
  - R_Acquire: client1
  - R_Acquire: server1
- **Step:** login
- **Component:** server1
- **Fork**
- **Step:** operand1
- **Join**
- **Step:** operan1
- **R_Release:** server1
- **R_Release:** client1
- **End**

**Start:** operand1_STEP_ID_92039
- **R_Acquire:** server1
- **Step:** SearchPic
- **R_Release:** server1
- **End**

Figure 4-15a – Parallel Combined Fragment

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
This simple example with parallel combined fragment invokes the following transformation rules:

a) generating the CSM Start element;

Figure 4-15b – Parallel Combined Fragment
b) transforming the first execution occurrence by the Client into an unnamed \textit{Step}
preceded by \textit{R\_Acquire: client} and ended by \textit{R\_Release: client};

c) transforming the synchronous call into \textit{Step: login} preceded by
\textit{R\_Acquire: server} and ended by \textit{R\_Release: server};

c) transforming the parallel combined fragment into a complex step preceded by
Fork and ended by Join. This complex step generates two complete new scenarios.
These both scenarios being very similar have a Start element, \textit{Step: searchPic}
for first scenario, \textit{Step: searchFile} for second scenario, each preceded by \textit{R\_Acquire: server} and ended by \textit{R\_Release: server}; and finally a CSM \textit{End}

d) generating the CSM \textit{End} step.

Figure 4.15a and 4.15b illustrates a use of Parallel Combined Fragment. Again,
this interaction starts off with a login message being passed to the server which yields
expected CSM as seen in Figure 4.15a and 4.15b. The parallel combined fragment yields
a fork in CSM and creates as many threads as there are operands in the combined
fragment, in this case 2. Each operand is yielded as a complex step in each thread and a
join is created to join both of the threads after their respective execution. The client
resource is released and the main scenario has ended.

Now the complex steps, which are a resultant of the operands (in this case,
Step:Operand1, and Step:Operand2) has a refinement which links to new scenarios.
Thread1’s scenario, which is a transformation of the interaction from operand1, can be
seen in Figure 4.15c, and Thread 2’s scenario, which is a transformation of the interaction
from operand2, can be seen in Figure 4.15d. In operand 1, SearchPic request is invoked on the server, which results the CSM. Server Resource is acquired, step to represent the SearchPic is created, Server Resource is released and the scenario is to an end. Operand 2 yields similar CSM as seen in Figure 4.15 d, except instead of SearchPic, step is created to represent SearchFile.

This simple example with alternate combined fragment invokes the following transformation rules:

a) generating the CSM Start element;

b) transforming the first execution occurrence by the Client into an unnamed Step preceded by \textit{R\_Acquire:client} and ended by \textit{R\_Release:client};

c) transforming the synchronous call into Step: login preceded by \textit{R\_Acquire:server} and ended by \textit{R\_Release:server};

c) transforming the alternate combined fragment into a complex step preceded by Branch and ended by a Merge. This complex step generates two complete new scenarios. These both scenarios being very similar have a Start element, Step: accessDatabase for first scenario, Step: guestLogin for second scenario, each preceded by \textit{R\_Acquire:server} and ended by \textit{R\_Release:server}; and finally a CSM End

d) generating the CSM End step.
4.4.2.6 Simple Interaction with Alternative Combined Fragment

![Diagram of UML 2.0 and CSM for simple interaction with alternative combined fragment]

Figure 4.16a – Alternate Combined Fragment
Figure 4.16b – Alternate Combined Fragment
Figure 4.16a and 4.16b illustrates what happens if you get a choice of actions. If a user tries to login and is successful, what will happen and if they are unsuccessful, what will happen then. A login request is sent out to the server with userid and password. If the login gets accepted, then the database is accessed. However, for whatever the reason, if the login returns an error, the client tries to do a guest login.

Initial CSM equivalent is the same as it was for previous CSM transformations. Start is created, Client Resource is acquired, and an initial step is created. Server resource is acquired and a login step is created. Since the request on the server is completed, the server resource is released. Upon encountering the alternate combined fragment, a branch is created. There are as many threads coming out of the branch as there are operands for the alternate combined fragments. For each thread, there is a complex step created which has a refinement which has a link to a scenario. The threads are then merged which is shown by a “merge” meta element, client resource is released and the main scenario comes to an End.
Figure 4.16 b – Alternate Combined Fragment
<table>
<thead>
<tr>
<th>UML 2.0</th>
<th>CSM</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Figure 4.16b – Alternate Combined Fragment
The two scenarios yielded from the operands of the combined fragment are shown in Figure 4.16 c and d for the two operands. The two figures are similar since they both are requesting an operation on the server. Scenario Figure 4.16 c represents the CSM equivalent of a successful login. This will acquire the server resource, create the step representing the request to AccessDatabase, and finally releasing the resource. On the other hand, Figure 4.16 d represents the CSM equivalent of a failed login where the user tries to do a guest login then. This series of actions will result in acquiring the server resource, creating the step representing the request to do a guest login and then finally releasing the resource.
4.4.2.7 Simple Interaction with Loop and Alternate Combined Fragment

Figure 4-17a – Loop Combined Fragment
This simple example with alternate combined fragment invokes the following transformation rules:

a) generating the CSM Start element;
b) transforming the first execution occurrence by the Client into an unnamed Step preceded by \textit{R\_Acquire: client} and ended by \textit{R\_Release:client};

c) transforming the loop combined fragment into a complex step. The number of repetition on the complex step will be 3 as detailed by the guard condition on the combined fragment. This will generate a completely new scenario which will have a CSM \textit{Start}, \textit{Step\_login} preceded by \textit{R\_Acquire:server} and ended by \textit{R\_Release:server}

c) transforming the optional combined fragment into a complex step. This complex step generates a complete new scenario. This scenario being very similar has a Start element, \textit{Step\_guestLogin}, preceded by \textit{R\_Acquire:server} and ended by \textit{R\_Release:server}; and finally a CSM End

d) generating the CSM \textit{End} step.

Figure 4.7 illustrates a usage of loop combined fragment in the interaction Entry. The idea behind this interaction is for the user to attempt their login 3 times maximum. If the user is not successful and has failed attempts, then the user will do a guest login.

The initial CSM shown in Figure 4.17a is similar to the previous ones in the sense that it starts off again by a CSM start element, resource acquire on the client and an initial step. For the transformation of a loop, a complex step is created. After the loop combined fragment, the optional combined fragment gets translated again into a complex step which has a refinement which is linked to another scenario. The client resource is released then and the transformation Ends.
The complex step, which represents the loop combined fragment, has a refinement which links to a new scenario. This complex step's number of repetition reflects those of the number of repetitions as indicated on the loop combined fragment. The new scenario (as seen in Figure 4.17 b) acquires the server resource, creates a step to represent the login request and releases the server resource.

The complex step, which represents the optional combined fragment, also has a refinement which links to a new scenario. This complex step's precondition will be that which will reflect the guard condition of the optional combined fragment. The new scenario acquires the server resource, creates a step to represent the guest login request and releases the server resource.

4.5 Complete Algorithm for the Transformation of UML 2.0 Model to CSM

The transformation algorithm described in this section (whose pseudocode is given in Appendix A) decides when to invoke the mapping rules discussed in section 4.4.1. The input to this transformation engine is the annotated model in XMI format.

As Figure 4-18 to Figure 4-21 suggests, the transformation engine takes in the model as the input. It searches and stores all the interactions within the given model. Within a interaction, the engine looks for the following three things: event occurrences, combined fragments, and interaction occurrences. Hence, the transformation algorithm is broken down into 3 major parts, as shown in Figure 4-18:
i) normal execution with no combined fragments nor any interaction occurrences

ii) normal execution with combined fragments

iii) normal execution with interaction occurrences

**Analyze Interaction**

![Flowchart](image)

Figure 4-18 – Analyze Interaction Flowchart

### 4.5.1 Normal execution with no combined fragments nor any interaction occurrences

The transformation engine applies the rules of section 4.4.1.1, 4.4.1.2 and 4.4.1.3, with the flowchart shown in Figure 4-19 for this scenario and produces the expected output.

The algorithm for normal execution with no combined fragments nor any interaction occurrences uses the event occurrences as the guide mark to realize the significance of important key points within the interaction list. The interaction list has the ordered pair of all interaction fragments. The algorithm goes through the list of the ordered set of event occurrences.
Analyze EventOccurrence

In Interaction given, check if beginning of Execution Occurrence

check ev is beginning of an ex?

no

msg = async?

yes

resource representing lifeline on which action has invoked, is acquired?

yes

Acquire Resource

createStep

no

Msg invoked on an active thread?

no

Join the new thread with existing thread

Fork from the invoking thread

Release caller's resource

yes

release of the caller's resource

Figure 4-19 – Analyze EventOccurrence FlowChart
If the event occurrence is a beginning of an execution occurrence, then it will check if the execution occurrence has been caused by a message or not. If it is, then they will check if the message is synchronous or asynchronous.

If the message is asynchronous, then there is a check to see if there is an execution occurrence from which this message was sent. If there is an execution occurrence on from which the message was originated from, the resource for that execution occurrence is located. That resource is released. Then a fork is created within the CSM model. For the execution occurrence which has received the message, the corresponding resource is acquired.

If the message is asynchronous, and the message is received on an already existing execution occurrence, then that means that there was already a thread running and this message is coming from another thread joining it. Hence, a CSM element join will be created there.

If there is no execution occurrence on from which the message was originated from, then for the execution occurrence which has received the message, a corresponding resource is acquired.

If there is no message or a synchronous message attached to the event occurrence and that event occurrence is a beginning of an execution occurrence, then this will simply result in a resource acquire for the corresponding resource for the execution occurrence for which this event occurrence is a beginning of. After that a step is created and added to a scenario.
If the event occurrence is an ending of an execution occurrence which will have caused a resource acquire, then a resource release associated with the same resource.

4.5.2 Normal execution with Combined Fragments

The transformation engine applies the rules of section 4.4.1.1 to 4.4.1.3, and 4.4.1.5 to 4.4.1.8 for this scenario with the flowchart shown in Figure 4-20 and produces the expected output.

Normal execution with combined fragments will follow normal execution except whenever there is a combined fragment, the engine will check the operator on the combined fragment to determine what kind of a combine fragment this is.

If there were an optional combined fragment, then a Step will be created. Refinement will be created and added to the step. This refinement will be associated with a new scenario. The new scenario will have the realization of the activities inside the operand of this optional combined fragment. The guard condition on the operand will result as Precondition element and will be added to the step.

If there were an alternative combined fragment, then a Fork will be created. For each operand, a step will be created and be forked out. A refinement will be created for each step and will be associated with a new scenario. The new scenario will have the realization of the activities inside each operand of this combined fragment. Finally a Join will be created to join all the separate Steps which were created. The guard condition on the operand will result as Precondition element and will be added to the step.
Figure 4-20 – Analyze Combined Fragment FlowChart
If there is a parallel combined fragment, then a Branch will be created. For each operand, a step will be created and be branched out. A refinement will be created for each step and will be associated with a new scenario. The new scenario will have the realization of the activities inside each operand of this combined fragment. Finally a Merge will be created to join all the separate Steps which were created. The guard condition on the operand will result as Precondition element and will be added to the step.

If there is a loop combined fragment, then a Step will be created with number of repetitions as per described in the operand. A refinement will be created for the step and will be associated with a new scenario. The new scenario will have the realization of the activities inside the operand of this combined fragment. The guard condition on the operand will result as Precondition element and will be added to the step.

4.5.3 Normal Execution with Interaction Occurrence

The transformation engine applies the rules of section 4.4.1.1 to 4.4.1.3, and 4.4.2.3 with the flowchart shown in Figure 4-21 for this scenario and produces the expected output.

Normal execution with interaction occurrence will follow the transformation of normal execution until it encounters the interaction occurrence. If there is an interaction occurrence, the engine will create a Step as soon as it encounters an Interaction Occurrence. A refinement will be created for the step and will be associated with
Analyze Interaction Occurrence

For Given Operand, Create Complex Step & Refinement

Analyze EventOccurrence

Figure 4-21 – Analyze Interaction Occurrence

a new scenario. The new scenario will have the transformation of the interaction to which this interaction occurrence refers to.
Chapter 5. Implementation and Testing

5.1 Implementation

The implementation for the transformation from UML 2.0 Interaction Model to Core Scenario Model was designed in a two-step process. First step for the transformation is to produce an internal data structure, which is quite similar to the Core Scenario Model’s data structure.

The UML 2.0 internal data structure does not have some of the information as apparent as the transformation will prefer to. For this reason, the internal data structure is created and a preprocessing is done on the meta elements. This is vital to be done especially in the cases where there are multiple threads due to asynchronous messages and other various situations.

5.1.1 PreProcessing to Produce Internal Data Structure

5.1.1.1 PathConnector Class

PathConnector is an enumeration, which can have the following values:

- Simple: a standard sequence connection between two elements.
- Fork: an and-split where two or more threads are resulting and those threads get executed in parallel.
- Join: an and-join where two or more threads are joining before the next execution can occur.
Branch: an or-split where two or more threads are resulting and only one of those threads get executed.

Merge: an or-join where only thread is required to come in for the next execution to occur.

5.1.1.2 PathFragment Class

PathFragment Class is introduced as part of the internal data structure. Mainly, its purpose is to keep track of the interaction fragments in relation with each other. It has 5 important attributes which need to be discussed. It has:

List of Previous Fragments – this will be a list of immediately previous fragments that are right before this interaction fragment. There will be usually a list of previous interaction fragments if there is a merge or a join occurring caused by multiple threads merging/joining.

List of Next Fragments – this will be a list of immediately next fragments that are right after this interaction fragment. There will be usually a list of next interaction fragments if there is a branch or fork caused by either an asynchronous message or some combined fragments.

Type of Previous PathConnector – this will have the type of the previous path connector

Type of Next Connector – this will have the type of the next PathConnector
InteractionFragment – this is a pointer to the interaction fragment that this
PathFragment is created for

5.1.1.3 Discussion of PreProcessing

A UML 2.0 model is created, either programmatically or through Rational
Software Architect. The UML 2.0 model is saved with all its UML 2.0 meta objects.
These meta objects are examined and a preprocessing is performed on them.

Event Occurrences and Execution Occurrences are searched within the UML 2.0
meta-object model. For each event occurrence, and an execution occurrence found, a
PathFragment will get created and the attribute “InteractionFragment” will point to the
under observation meta-element(either that event occurrence or execution occurrence).
The list of PreviousFragments in Path Fragment will have the previous fragments right
before this under observation meta-element. Similarly, the list of NextFragments in Path
Fragment will have the next fragments right after this under observation meta-element.
For normal scenarios, the type of nextFragment and previousFragment will be simple.

If there is an event occurrence that comes along which is the receive of an
asynchronous message that comes along, for that message, there is a PathFragment
created. The type of nextFragment for a fragment right before this one will be a fork.
The list of nextFragments for a fragment right before this one will include two
pathFragments – one will be PathFragment that will represent the event occurrences /
execution occurrences of the new thread which just spawned off due to the asynchronous
message, second will be PathFragment that will represent the be event occurrences / execution occurrences of the original thread.

If there is a parallel combined fragment, there is a PathFragment created which would have the interactionFragment point to that parallel combined fragment. The type of nextFragment for a fragment right before this one will be a fork. The list of nextFragments for a fragment right before this one will include at least two pathFragments – one pathFragment for event / execution occurrence for each operand. At the end of the combined fragment, all the separate threads will come together by a join. The list of previous fragments for the next pathFragment after the end of a combined fragment will have the last elements of the branches representing the separate threads.

If there is an alternate combined fragment, there is a PathFragment created which will have the interactionFragment point to that alternate combined fragment. The type of nextFragment for a fragment right before this one will be a branch. The list of nextFragments for a fragment right before this one will include at least two pathFragments – one pathFragment for event / execution occurrence for each operand. At the end of the combined fragment, all the separate threads will come together by a branch. The list of previous fragments for the next pathFragment after the end of a combined fragment will have the last elements of the branches representing the separate threads.

If there is an interaction occurrence, a Path Fragment will be created with the similar attributes as if a PathFragment will be created for an event / execution occurrence.
The only difference will be that the interactionFragment attribute would point to the interactionOccurrence.

5.1.2 Transformation from Internal Data Structure to CSM

The internal data structure is primarily composed of PathFragments with all the necessary information set in its attributes. This was rendered as a result of the preprocessing done earlier.

Now the transformation rules apply which were discussed earlier in Chapter 4.0. For each pathFragment which links to an execution occurrence, the transformation engine creates a Step. Every time there is an event occurrence and that event occurrence is the receive of a message, that will be an indication for of an execution occurrence. This will mean that a resource might be required to be acquired. An internal list of acquired resources is checked to see if the list contains the property that lifeline represents on which that execution occurrence is. If it is already in the acquired resource list, then no resource acquire is created. However, if the resource is not found on the acquire resource list, then Resource Acquired in CSM is created as well as that property is added to the internal acquired resource list.

Similar thing happens when dealing with a PathFragment which links to an event occurrence which is the end of an execution occurrence. The same internal list of acquired resources is checked to see if the list contains the property that lifeline represents on which that execution occurrence is. If it is in the acquired resource list, then no resource acquire is created but rather resource release is created for that resource.
This resource is then removed from the internal list of acquired resources. However, if the resource is not found on the acquired resources list, then nothing is done.

If a PathFragment is found with its next PathConnector to be of branch / fork, then similar CSM elements (CSM branch or CSM fork) are created as per the situation will be. If a PathFragment is found with its previous PathConnector to be of merge / join branch, then similar CSM elements (CSM merge or CSM join) are created as per the situation will be.

If a PathFragment is found with interactionFragment pointing to an interaction occurrence, a CSM Step is created with CSM refinement. This CSM refinement will point to a scenario, in which the interaction this interactionOccurrence points to will have been transformed into CSM.

5.2 Testing and Verification

The main purpose of this thesis was to develop a transformation algorithm from UML 2.0 Interaction Diagrams to Core Scenario Models. The transformation algorithm was developed and implemented and tested out. Various models were used for the testing purposes, simples ones as well complex ones. Test Cases (TC), as well as the case study as discussed in this chapter, were developed to test the transformation rules and verify the XML outputs. Not only the expected output model elements are present but also the XML output is also verified against XML Spy 2005 and IBM Eclipse for the validity of the XML output being generated.
As already discussed UML 2.0 is a very detail specification with plenty of information. However, Core Scenario Model, a simpler model, will have enough information to satisfy the needs. There is quite a bit of information which is not useful for the purposes of the PUMA project [PUMA05]. For this reason, this thesis has focused on the transformation from UML 2.0 to Core Scenario Models. Almost all the elements were used for the transformation to CSM. There was not a one to one mapping between UML 2.0 Interaction to CSM as the point of using CSM is to remove the details which UML 2.0 Interaction brings in. The meta-elements which were not used at all in relations to the transformation are:

Continuation – This meta element is used to describe continuations of difference branches of an Alternative Combined Fragment. There is no need for the transformation of this element as there is no concept of continuation in Core Scenario Model.

Gate – This meta element is used to describe a connection point for relating a Message outside an Interaction Fragment with a message inside the interactionFragment. There are concepts of refinement which link one scenario to another scenario themselves but not link Steps within one scenario to another one.

StateInvariant – This meta-element describes the constraints on the state(or the values of eventual attributes) of a Lifeline. In the transformation, there is no need to keep track of any kind of constraints on the state of the lifelines as there are no matching CSM constraints for any CSM elements of this sort.
5.2.1 Testing with Various Test Scenarios

The initial simple test cases, Test Cases #1 - #7 from Table 1-8, which tested each transformation, along with the corresponding input / output figures, have been documented in Chapter 4. The more complex test cases, Test Cases #8 – 10, test out a mix combination of some of these simple transformations using a case study as per described in Section 5.1 to 5.4. Test Case #8 reflects the Acquire / Store Video Scenario (Section 5.4.1), where the description is given for the transformations to CSM of a simple synchronous message, of an asynchronous message, and a loop combined fragment. Test Case #9 reflects the Access Control Scenario (Section 5.4.2), where the transformation to CSM is illustrated of a simple synchronous message, of an asynchronous message, couple of optional combined fragments and an alternate combined fragment. Test Case #10 reflects the Manage Rights (Section 5.4.3), where the transformation to CSM is illustrated of a simple synchronous message, of an asynchronous message, and an optional combined fragment.
<table>
<thead>
<tr>
<th></th>
<th>TC #10</th>
<th>TC #9</th>
<th>TC #8</th>
<th>TC #7</th>
<th>TC #6</th>
<th>TC #5</th>
<th>TC #4</th>
<th>TC #3</th>
<th>TC #2</th>
<th>TC #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Simple Interaction with Synchronous Message
- Simple Interaction with Asynchronous Message
- Interaction Occurrence
- Simple Interaction with Optional Combined Fragment
- Simple Interaction with Parallel Combined Fragment
- Simple Interaction with Alternative Combined Fragment
- Simple Interaction with Loop Combined Fragment
5.2.2 Verification

The CSM models produced by the transformation algorithm are in XML format. These XML documents are being produced using the transformation engine built in IBM’s Eclipse. Eclipse has its own built-in XML tools which validate XML documents being fed back into Eclipse. Eclipse will discover any errors if there were any based on the given XMI Schema. Eclipse confirmed the well-formedness of the produced documents as well as the validity.

Another tool, Altova XML Spy 2005, was also used to check the well formedness and the validity of the given XML documents. XML Spy 2005 validated the files and gave a clean bill of health in well-formedness area as well.
Chapter 6. Case Study

The previous chapters have discussed the concepts and the algorithm of the transformation from UML 2.0 to CSM. This chapter presents the application of the proposed UML to 2.0 models to the CSM transformation algorithm to a more substantial example: a Building Security System that was introduced in [Petriu+02] and used in other papers [PUMA05 ].

6.1 Description of the Building Security System

The Building Security System (BSS) application describes a security-management function for a medium size building. The primary purpose of this system is to prevent/report any unauthorized entry into a building or a secure area of the building. It examines the patrols of the security guards and reports any anomalies in their patrol schedules. This system also is responsible for monitoring fire and smoke detectors along with the raising of the appropriate alarms.

This system is responsible for the following three tasks:

i) to acquire and store the video feed of video cameras at some intervals

ii) to control the entry into a building when a user uses a card through a card reader

iii) to give access to managers to manage rights of a normal user
This thesis takes all three scenarios as its examples and applies the transformations and produces the appropriate CSM output. These scenarios will be described first by a use case, followed by an annotated deployment diagram and lastly an annotated interaction diagram to describe the flow of events in this scenario. Then the transformation will be applied and the resultant CSM will be discussed.

6.2 Use Cases

![Use Case Diagram](image)

Figure 6-1 – BSS Use Cases

6.2.1 Access Control Scenario

The Access Control Use Case is responsible for the card reader events. It communicates to the database to check the access rights of the card holder in a database. If the access is granted, then it triggers an actuator in the door frame to permit access.
This system should be able to respond to a random stream of card accesses at an average rate of 2 per minute, with 95% of responses being completed within one second.

6.2.2 Acquire / Store Video

The Acquire / Store Video Use Case monitors a given number $n$ of video surveillance cameras one at a time, retrieves images at every given interval, and stores it in a database. From each camera, an image is retrieved at least once per second and it is interpreted into a requirement that 95% of polling cycles for all $n$ cameras should be completed in one second. This system should be capable of taking care of 50 cameras.

6.2.3 Manage Rights

The Manage Rights Use Case will be responsible for handling the requests made by a manager to change rights of a user. The system will first check if the user exists in the database. If they do, then the rights will be modified and the database will be updated. At the end, the manager will be notified of the status of their request. The system should be able to handle a random stream of modification of rights at an average rate of 2 per minute with 95% of responses being completed within once second.
6.3 Deployment Diagram

Figure 6-2 – BSS Deployment Diagram

As illustrated in Figure 5.1, for the Building Security System, there are 6 nodes to support this system. There is a security card reader node, door lock actuator node, video camera node, and a disk node as PAresources. These are linked through the Local Area Network (as another resource), to the application processor and a database, with a separate processor.
6.4 Scenarios

6.4.1 Acquire / Store

As illustrated in Figure 5.3, the acquiring and storing video from the camera is described in a UML 2.0 sequence diagram. There are 5 lifelines, each representing a class and annotated as a PAresource. This whole sequence is repeated N times and hence a Loop combined fragment is used to model this repetition. All the messages being passed back and forth are synchronous messages except the storeImage which is an asynchronous message.

Video Controller requests the AcquireProc to process one image at a time. AcquireProc gets a buffer and requests BufferManager to allocate the buffer. The image is retrieved over the network and passed in. Then, asynchronously the image is sent to the database to be stored where the image is written to the database. Finally the buffer is released. While the asynchronous call is made for the image to be written, the acquireProc replies back to the video controller to say that the request to the processImage is complete and is ready for more images to be processed.

The CSM transformation for this scenario is shown in Figure 5.4 a and 5.4 b. Since the scenario is only one combined fragment, a loop combined fragment, this is shown in the transformation in Figure 5.4a where there is only one complex step to represent that loop combined fragment.

The complex step is examined more in detail in Figure 5.4 b where everything inside the loop combined fragment is actually transformed into CSM. Video Controller,
since starts off the executions, the first thing is to acquire the video controller and create an initial step to show the initial start of the executions. AcquireProc is the next resource which has methods being invoked so AcquireProc is going to be acquired. Steps are created to show the processOneImage and getBuffer. Next, bufferManager is acquired for the allocation of the buffer as shown by the ResourceAcquire and a Step. Since nothing else is required of the BufferManager, the BufferManager Resource is released. GetImage and passImage is invoked on AcquireProc which are shown by the corresponding steps on the CSM diagram. StoreImage is invoked on the resource StoreProc. However, this is an asynchronous call and hence there is another thread created to satisfy this call. As shown in the CSM, a fork is created and two threads come out of it. One is for the StoreProc and its any execution leading StoreProc, and the second one is the continuation of the execution on the same lifeline(i.e. further executions on AcquireProc).

AcquireProc send a message back videoController as a reply back to the initial procOneImage request. This is translated in CSM as the release of AcquireProc and since there are no further executions on VideoController also, there is a release on VideoController as well.

StoreImage is an asynchronous call to the storeProc so that thread will first acquire the StoreProc and create a Step to represent the storeImage. Then there is a step created for store function being invoked which inturns writes to the database. Before the writing to the database, database resource is acquired and then a Step is created to reflect the writing of the Image to the database. Since there is no more executions on the
database, the database resource is released. FreeBuf is executed to free up the buffer and a Step is created to reflect this request. This request also goes to the bufferManager resource and executes releaseBuf. For this request the bufferManager resource is acquired, a step is created to represent releaseBuf, and the bufferManager is released. This brings the scenario to an end.

Figure 6-3 -- Store / Acquire Video Sequence Diagram
Figure 6-4 a – Acquire / Store Video CSM Diagram

The generated CSM model is given in the Appendix B.
Figure 6-4 b -- Acquire / Store Video CSM Diagram
6.4.2 Access Control

Another scenario is that of Access Control as illustrated in Figure 5.5. This is quite an interesting scenario as it has multiple combine fragments which get transformed into CSM as well as synchronous and asynchronous messages.

The user starts the sequence off by inserting a card in a cardReader. This invokes the asynchronous function, cardRead, in the CardReader. This calls the asynchronous message, admit, to be called at the accessController. The accessController gets the rights for the user. For this, the accessController reads the user’s rights from the database. If the information is not in the cache of the database, then the information is retrieved from the disk. Once the accessController has the rights, it checks them and then either the doorLock is opened and the user is requested to enter the building or an alarm is raised. Optionally, this whole sequence gets logged and to the database and eventually to the disk.

The CSM transformation is quite straightforward for the most part. Initially, User resource is acquired and readCard is invoked on the cardReader resource. CardReader resource is acquired and a step is created to reflect the execution of the readCard. Then admit function is invoked on the accessController, accessController resource is acquired and a step is created to reflect that of the execution of admit function. Since the readCard and the admit function both are asynchronous message, there is a fork before them and another thread starts off, leaving the original thread to continue. Since there is nothing more being done on either of the original thread after the fork, the resources (i.e.
cardReader and AccessController) are released which were acquired before the invocation of readCard and admit.

Since there is execution being done on the accessController, accessController resource is acquired and getRights function is invoked. The rights are read from the database. This is reflected by the ResourceAcquire of the database resource and a Step to reflect the execution of that retrieval of the rights. If the rights are not found in the database cache, the rights will be retrieved from the disk. This produces an optional combined fragment which gets transformed into a complex step type. This complex step type details are shown in Figure 5.6 c. The disk resource is acquired and the rights are retrieved from the disk as illustrated by a Step and the disk resource is released. These rights are passed back to the accessControl resource and are checked. If the rights are sufficient enough, the doors shall open, else an alarm shall be raised. This little choice is transformed into CSM by a branch followed by a couple of complex steps. Either the rights are OK, and the door is opened (Figure 5.6 c) or the rights are NOT OK, and an alarm is raised (Figure 5.6 d).
Figure 6-5 – Access Control Annotated Sequence Diagram
Figure 6.6 a - CSM Transformation of Access Control Scenario
Figure 6.6 b – CSM Transformation of Access Control Scenario

Figure 6.6 c – CSM Transformation of Access Control Scenario

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
6.4.3 Manage Rights

The manager is the one that can initiate this scenario (Figure 5.7) of settings rights for a user. Manager will request the accessController to set the rights for a particular user. The accessController will check the database for the user’s existence. If the user exists, then the accessController goes ahead and updates the database with the given
rights for the given user. At the end, a status is displayed back to the manager, giving
them appropriate information on if the rights have been updated successfully or not.

This gets transformed into CSM as shown in Figure 5.8a. As before, the first
resource (in this case, Manager) is acquired and an initial step is created to show the first
execution. Since the first message is an asynchronous one, the call for setRights, there is
a fork to show the second thread which just spawned off. Since there is no more
execution to be done on the first thread anymore, that thread releases the Manager
resource and comes to an end. The spawned thread continues ahead and acquires the
AccessController and has a Step to show the execution of setRights. Next, Database
resource is acquired and a step is created to reflect the execution of checkUserExists.
This leads to disk resource being acquired and another step created to reflect the
execution of checkUserExists. If the user exists, then the rights will be set for that user.
To show this execution, there is an optional combined fragment with the guard condition
of “EXIST.” This combined fragment gets transformed into a complex step. The inner
detail of this complex step is shown in Figure 5.8 b. Database Resource is acquired, a
Step is created to reflect execution on setRights call on the database. Then disk resource
is acquired, and a Step is created to reflect execution on setRights call on disk. Finally an
asynchronous message is invoked at the manager resource to give the status of the
request. Since this is asynchronous, there is another thread created for this execution.
Manager resource is acquired and a step is created to reflect Status message. Since there
is nothing after that on either thread, both of the threads come to an end.
Figure 6-7 – Manage Rights Annotated Sequence Diagram
Figure 6-8 a – Manage Rights CSM Diagram
Figure 6-8 b – Manage Rights CSM Diagram
Chapter 7. Conclusions

7.1 Conclusion

The thesis proposed and implemented a Java based algorithm for the transformation from UML 2.0 annotated interaction and deployment diagrams to the Core Scenario Model (CSM) as part of the PUMA project [PUMA05]. The input to this transformation algorithm is the Java data structure of a UML 2.0 annotated model, which can be created in one of two ways: a) by an Eclipse transformation of an XMI file obtained from a UML 2.0 editor, such as Rational Software Architect, or b) programmatically with UML2, an Eclipse plug-in. The output is the corresponding CSM model description file in XML format, which can then be read by other tools for further transformations. An important part of this thesis is the adapting of the UML Profile for Schedulability, Performance, and Time Specification for UML 2.0.

The proposed transformation was implemented as a mapping from the UML 2.0 meta-model to the CSM meta-model. The implementation applies the algorithm discussed in section 4.5, which invokes the transformation rules presented in section 4.4. The mapping from UML 2.0 to CSM is heavily based on the SPT stereotypes. Therefore, the UML model elements which are stereotyped with SPT are transformed to CSM, while non-stereotyped model elements would get skipped over. We assume that the construction of the input models follows the patterns presented in section 4.4.1 and 4.4.2. If the input model does not follow those restrictions, there is no guarantee of the correctness of the transformation. The implementation demonstrates the feasibility of the approach of the transformation from UML 2.0 to CSM. For the robustness of the
transformation, more has to be done in terms of recognizing incorrect input as well as better error reporting.

The main results of the thesis were reported, together with other aspects of the PUMA project, in the following conference paper:


This work as a step in a larger research project, aims at deriving Core Scenario models from UML 2.0 models. Although the UML 2.0 standard, the XMI standard, the Core Scenario Model Schema and the UML Profile for Schedulability, Performance, and Time Specification standard are still evolving, the conceptual approach proposed in this thesis will be applicable to any future versions.

7.2 Future Work

Although the work done for this thesis is quite thorough in realizing the transformation from UML 2.0 Interaction Models to Core Scenario Models, the following are some of the many possibilities for future research:

- As mentioned earlier, this thesis is part of a larger project, PUMA, aimed at deriving performance models from UML models and integrating the results back to the UML models. This thesis does a forward UML to CSM transformation, but does not attempt the backward CSM to UML transformation. This would be quite useful for revisiting already analyzed models.
• Since this is part of the PUMA project, it has to be tested more thoroughly with the other parts of PUMA.

• The current implementation of the transformation is based on IBM’s tools such as Eclipse and Rational Software Architect to produce the input to feed into the transformation engine. Ideally, it would be a very good idea to have multiple tools to produce compatible inputs for the transformation engine.

• The SPT Profile is by no means updated and standardized for UML 2.0. The profile, after its standardization, would need to be reimplemented and reapplied and the transformation engine would need to be visited again to take care of these changes.

• UML 2.0 Standard Specification is not completely finalized as there are revisions that keep on coming along. The transformation engine along with the input model would need to be updated to produce the correct outputs.

• The Core Scenario Model is not completely finalized as well as there are revisions that are released. The transformation engine along with the output model would need to be updated to produce the correct outputs.
References


[EclipseUML2] Slides from the Eclipse Foundation review of the UML 2 1.1 Release, See http://dev.eclipse.org/viewcvs/indextools.cgi/%7Echeckout%7E/uqml-home/docs/presentations/UML2_1_1_ReleaseReview.html


[JUML2] Javadoc for the 1.1 release of the UML2 project., See http://download.eclipse.org/tools/uml2/1.1.0/javadoc/


[Reenskaug96] Reenskaugh, T., Working Objects, Manning, Greenwich, 1996


140

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Shen, H., “XMI-Based Transformation From UML Models to LQN Performance Models”, Ottawa, Thesis (MEng), Carleton University, 2002


UML2.0 Plug-in http://www.eclipse.org/uml2/


World Wide Consortium, See http://www.w3c.org

Williams, L.G., & Smith, C.U., “Performance Evaluation of Software Architectures” in Proc of WOSP’98, Santa Fe, New Mexico, USA, 1998

Appendix A. Transformation Algorithm

UML2CSM Transformation Algorithm:

gets a references to a model => model
Search within the model and nested elements for all Interactions =>
listOfInteractions

CSM = createCSMElement(CSM, <>)

foreach Interaction in ListOfInteractions {
    scenario = createCSMElement(Scenario)
    addCSMElement(scenario, CSM)

    start = createCSMElement(Start)
    addCSMElement(start, scenario)

    foreach interactionFragment in Interaction {
        switch interactionFragment {
            case eventOccurrence {
                createCSMForEventOccurrence(ev, scenario)
            }
            case combinedFragments {
                createCSMForCombinedFragments(cf)
            }
            case interactionOccurrence {
                createCSMForInteractionOccurrence(io)
            }
        }
    }
}

createCSMForEventOccurrence(ev1, scenario):
// EV1 stands for Event Occurrence
// EO1 stands for any Execution Occurrence

if ( beginningOfEO(EV1)) {
    EO1 = EV1.getStartExec
    if ( EV1.getMessage is (NOT null & instance of a message)) {
        msg = EV1.getMessage
        if ( msg = async ) {

        }
    }
}
// this is if there is already an active execution occurrence when this message arrives so this is an async merging in with an existing thread
if (findExecutionOccurrenceForThisEV(EV1)) {
    join = createCSMElement(Join)
    addCSMElement(join, scenario)
}

fork = createCSMElement(Fork)
addCSMElement(fork, scenario)

if (acquiredResources.find(EO1)) {
    EV2 = msg.getSendEvent
    EO2 = findExecutionOccurrenceForThisEV(EV2)
    resource = getResourceForThisEO(EO1)
    releaseResource = createCSMElement(ReleaseResource)
    addCSMElement(releaseResource, scenario)
}

if (doesNotExistResourceForThisEO(EO1)) {
    acquireResource = createCSMElement(AcquireResource)
    addCSMElement(acquireResource, scenario)
    // internal list: acquiredResourcesList
    addAcquiredResources(acquireResource, EO1)
}

step = createCSMElement(Step)
addCSMElement(step, scenario)

if (endingOfEO(EV1)) {
    EO1 = EV1.getFinishExec
    if (acquiredResourcesList.found(EO1)) {
        releaseResources = createdCSMElement(ReleaseResource)
        addCSMElement(releaseResources, scenario)
        removeAcquiredResources(EO1)
    }
}

createCSMElement(nameOfCSMElement) {
    case nameOfCSMElement:
switch "CSM":
    CSM CSM = new CSM(
    CSM.setID(CSMID)
    return CSM

switch "Step":
    Step step = new Step()
    step.setID(stepID)
    return step

addCSMEElement(CSMEElement, parentCSMEElement) {
    parentCSMEElement.add(CSMEElement)
}

addAcquiredResources(acquiredResource, EO1) {
    resource = findResource(EO1)
    resourceArray [0] = resource
    resourceArray [1] = EO1
    acquiredResourceVector.add(arrayElement)
}

removeAcquiredResources(EO1) {
    iterator acquiredResourceIterator = acquiredResourceVector.iterator
    while (acquiredResourceIterator.hasNext() ) {
        Array resourceArray = acquiredResourceIterator.next()
        if ( resourceArray[1] == EO1 ) {
            acquiredResourceVector.remove(resourceArray)
            break;
        }
    }
}

findExecutionOccurrenceForThisEV(EV2) {
    parentCSMEElement = EV2.getParentCSMEElement
    indexOfEV2 = parentCSMEElement.getIndexOf(EV2)
    for (int i =0 ; i < indexOfEV2; i ++ ) {
        if (parentCSMEElement.getIndex(i) instanceof EventOccurrence) {
            someEv = parentCSMEElement.getIndex(i)
            if (someEv.getLifeLine == EV2.getLifeLine) {

if (someEv.getStartExec() != null) {
    someEo = someEv.getStartExec()
    someEvFinish = someEo.getFinish
    if (parentCSMElement.getIndexOf(someEvFinish) > indexofEV2) {
        return someEo;
    }
}

findResource(EO1) {
    lifeline1 = E01.getCovers()
    lifeLineIndex = ResourceLifeLineVector.indexOf(lifeline1)
    iterator resourceLifeLineIterator = resourceLifeLineVector.iterator
    while (resourceLifeLineIterator.hasNext()) {
        Array resourceLifeLineArray = resourceLifeLineIterator.next()
        if (resourceArray[0] == lifeLine1) {
            return resourceArray[1]
            break;
        }
    }
}

PopulateResourceLifeLineArray(interaction) {
    Component component = createCSMElement(Component)
    addCSMElement(component, CSM)
    property p1 = lifeline.getProperty
    if (p1.getType does exist) {
        Class class = p1.getType();
        component.setName(class.getName)
    } else {

    }
}

CombinedFragments

createCSMForCombinedFragments(cf) {
    switch interactionOperator {

case optPath:
    follow optPath

case altPath:
    follow altPath

case parPath:
    follow parPath

case loopPath:
    follow loopPath

}

optPath:
    step = createCSMElement(Step)
    refinement = createCSMElement(Refinement)
    addCSMElement(refinement, step)
    createAndPopulateScenario(Operand)
    if (guardConditionExists) {
        precondition = createCSMElement(PreCondition)
        addCSMElement(precondition, step)
    }

altPath:
    createCSMElement(Branch)
    foreach oper operandList {
        createCSMElement(Step)
        createCSMElement(Refinement)
        addCSMElement(refinement, step)
        createAndPopulateScenario(Operand)
        if (guardConditionExists) {
            precondition = createCSMElement(PreCondition)
            addCSMElement(precondition, step)
        }
    }
    createCSMElementMerge

parPath:
    createCSMElement(Fork)
    foreach oper OperandList {
        createCSMElement(Step)
        createCSMElement(Refinement)
        addCSMElement(refinement, step)
        createAndPopulateScenario(Operand)
        if (guardConditionExists) {
            precondition = createCSMElement(PreCondition)
            addCSMElement(precondition, step)
        }
    }

addCSMElement(precondition, step)

createCSMElement(Join)

**loopPath:**
createCSMElementStep with n = N
createCSMElement(Refinement)
addCSMElement(refinement, step)
createAndPopulateScenario(Operand)

---

**Interaction Occurrence**

createCSMForInteractionOccurrence(io) {
createCSMElementStep
createCSMElement(Refinement)
addCSMElement(refinement, step)
createAndPopulateScenario(Operand)
}

---

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Appendix B. CSM model for BSS case study

B.1. CSM Model for Acquire / Store Video Scenario

<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
  <Scenario id="ID_76458" name="acquire/store video take 3" probability="1.0">
    <Step id="ID_76332" name="" predecessor="ID_28406" successor="ID_53074"/>
    <Step id="ID_38849" name="0,2147483647" predecessor="ID_53074" successor="ID_97872">
      <Refinement parent="ID_38849" sub="ID_60848"/>
    </Step>
    <ResourceAcquire id="ID_59269" predecessor="ID_20246" successor="ID_28406" acquire="ID_72347" rUnits="1.0"/>
    <ResourceRelease id="ID_32243" predecessor="ID_97872" successor="ID_81024" release="ID_72347"/>
    <Start id="ID_20246" target="ID_59269"/>
    <End id="ID_81024" source="ID_32243"/>
    <Sequence id="ID_28406" source="ID_59269" target="ID_76332"/>
    <Sequence id="ID_53074" source="ID_76332" target="ID_38849"/>
    <Sequence id="ID_97872" source="ID_38849" target="ID_32243"/>
  </Scenario>
</CSM:CSMType>
<Step id="ID_24651" name="freeBuf" predecessor="ID_58364"
successor="ID_67456"/>
<Step id="ID_12221" name="releaseBuf" predecessor="ID_2317"
successor="ID_27548"/>
<ResourceAcquire id="ID_45176" predecessor="ID_99588"
successor="ID_20251" acquire="ID_89195" rUnits="1.0"/>
<ResourceAcquire id="ID_92664" predecessor="ID_32138"
successor="ID_31082" acquire="ID_15749" rUnits="1.0"/>
<ResourceAcquire id="ID_97635" predecessor="ID_42586"
successor="ID_63336" acquire="ID_85222" rUnits="1.0"/>
<ResourceAcquire id="ID_13553" predecessor="ID_95222"
successor="ID_86622" acquire="ID_19017" rUnits="1.0"/>
<ResourceAcquire id="ID_85381" predecessor="ID_67456"
successor="ID_2317" acquire="ID_15749" rUnits="1.0"/>
<ResourceRelease id="ID_60233" predecessor="ID_21421"
successor="ID_82860" release="ID_15749"/>
<ResourceRelease id="ID_45176" predecessor="ID_48965"
successor="ID_42586" release="ID_89195"/>
<ResourceRelease id="ID_18012" predecessor="ID_47888"
successor="ID_58364" release="ID_19017"/>
<ResourceRelease id="ID_51788" predecessor="ID_27548"
successor="ID_52738" release="ID_15749"/>
<ResourceRelease id="ID_25387" predecessor="ID_52738"
successor="ID_92502" release="ID_85222"/>
<ResourceRelease id="ID_91602" predecessor="ID_48965"
successor="ID_68731" release="ID_89195"/>
<Start id="ID_99588" target="ID_45176"/>
<End id="ID_92502" source="ID_25387"/>
<End id="ID_68731" source="ID_91602"/>
<Sequence id="ID_20251" source="ID_45176" target="ID_85888"/>
<Sequence id="ID_51977" source="ID_85888" target="ID_5700"/>
<Sequence id="ID_32138" source="ID_5700" target="ID_92664"/>
<Sequence id="ID_31082" source="ID_92664" target="ID_32616"/>
<Sequence id="ID_21421" source="ID_32616" target="ID_60233"/>
<Sequence id="ID_82860" source="ID_60233" target="ID_51927"/>
<Sequence id="ID_29949" source="ID_51927" target="ID_45289"/>
<Sequence id="ID_42586" source="ID_50495" target="ID_97635"/>
<Sequence id="ID_63336" source="ID_97635" target="ID_22693"/>
<Sequence id="ID_89464" source="ID_22693" target="ID_38081"/>
<Sequence id="ID_95222" source="ID_38081" target="ID_13553"/>
<Sequence id="ID_86622" source="ID_13553" target="ID_79014"/>
<Sequence id="ID_47888" source="ID_79014" target="ID_18012"/>
<Sequence id="ID_58364" source="ID_18012" target="ID_24651"/>
<Sequence id="ID_67456" source="ID_24651" target="ID_85381"/>
<Sequence id="ID_2317" source="ID_85381" target="ID_12221"/>
<Sequence id="ID_52738" source="ID_51788" target="ID_25387"/>
<Fork id="ID_48965" source="ID_45289" target="ID_50495 ID_91602"/>
</Scenario>
<PassiveResource xsi:type="CSM:ComponentType" id="ID_72347"
name="video Controller2"/>
<PassiveResource xsi:type="CSM:ComponentType" id="ID_89195"
name="acquireProc2"/>
<PassiveResource xsi:type="CSM:ComponentType" id="ID_15749"
name="bufferManager2"/>
<PassiveResource xsi:type="CSM:ComponentType" id="ID_85222"
name="storeProc2"/>
<PassiveResource xsi:type="CSM:ComponentType" id="ID_19017"
name="database2"/>
</CSM:CSMType>
B.2. CSM for Access Control Scenario:

```xml
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform/resource/edu.carleton.sce.puma/CSM.xsd">
  <Scenario id="ID_77511" name="Access Control" probability="1.0">
    <Step id="ID_16043" name="" predecessor="ID_13717" successor="ID_2073"/>
    <Step id="ID_91094" name="readCard" predecessor="ID_70916" successor="ID_16280"/>
    <Step id="ID_61542" name="admit(cardInfo)" predecessor="ID_85979" successor="ID_4456"/>
    <Step id="ID_29764" name="getRights" predecessor="ID_4456" successor="ID_40643"/>
    <Step id="ID_77065" name="readRights" predecessor="ID_19806" successor="ID_10888"/>
    <Step id="ID_1056" name="not in cache" predecessor="ID_10888" successor="ID_38646">
      <Refinement parent="ID_1056" sub="ID_3247"/>
    </Step>
    <Step id="ID_32287" name="checkRights" predecessor="ID_19862" successor="ID_10132"/>
    <Step id="ID_33636" name="OK" predecessor="ID_10132" successor="ID_88004">
      <Refinement parent="ID_33636" sub="ID_77234"/>
    </Step>
    <Step id="ID_13680" name="not OK" predecessor="ID_10132" successor="ID_88004">
      <Refinement parent="ID_13680" sub="ID_81395"/>
    </Step>
    <Step id="ID_12565" name="need to log" predecessor="ID_88004" successor="ID_23240">
      <Refinement parent="ID_12565" sub="ID_55167"/>
    </Step>
    <ResourceAcquire id="ID_20965" predecessor="ID_46958" successor="ID_13717" acquire="ID_63290" rUnits="1.0"/>
    <ResourceAcquire id="ID_43422" predecessor="ID_33587" successor="ID_70916" acquire="ID_18329" rUnits="1.0"/>
    <ResourceAcquire id="ID_93330" predecessor="ID_91952" successor="ID_85979" acquire="ID_57646" rUnits="1.0"/>
    <ResourceAcquire id="ID_28785" predecessor="ID_40643" successor="ID_19806" acquire="ID_41502" rUnits="1.0"/>
    <ResourceRelease id="ID_40905" predecessor="ID_2073" successor="ID_33587" release="ID_63290"/>
  </Scenario>
</CSM:CSMType>
```
<Scenario>
  <Scenario id="ID_96167" name="not in cache_STEP_ID_35153">
    <Step id="ID_94702" name="readData" predecessor="ID_31223">
      <ResourceAcquire id="ID_11323" predecessor="ID_86729">
        <ResourceRelease id="ID_18840" predecessor="ID_44736"/>
      </ResourceAcquire>
      <Start id="ID_86729" target="ID_11323"/>
      <End id="ID_94702" source="ID_18840"/>
      <Sequence id="ID_31223" source="ID_94702" target="ID_11323"/>
      <Sequence id="ID_86729" source="ID_31223" target="ID_94702"/>
    </Step>
    <Step id="ID_36285" name="OK_STEP_ID_15466">
      <Step id="ID_81794" name="openDoor" predecessor="ID_92424">
        <ResourceAcquire id="ID_12387" acquire="ID_23857" rUnits="1.0"/>
        <ResourceRelease id="ID_18840" predecessor="ID_44736"/>
        <Start id="ID_81794" target="ID_12387"/>
        <End id="ID_92424" source="ID_18840"/>
      </Step>
      <ResourceRelease id="ID_11323" predecessor="ID_94702"/>
      <Sequence id="ID_12387" source="ID_94702" target="ID_11323"/>
      <Sequence id="ID_36285" source="ID_12387" target="ID_92424"/>
    </Step>
  </Scenario>
</Scenario>
<Scenario>
  <Step id="ID_64578" name="enterBuilding" predecessor="ID_31896" successor="ID_76309"/>
  <ResourceAcquire id="ID_12307" acquire="ID_82741" rUnits="1.0" predecessor="ID_74151" successor="ID_92424" acquire="ID_43044" rUnits="1.0"/>
  <ResourceRelease id="ID_40707" release="ID_82741" predecessor="ID_57284" successor="ID_82741" />
  <ResourceRelease id="ID_43044" release="ID_40707" predecessor="ID_22801" successor="ID_64578" />
  <ResourceRelease id="ID_19639" release="ID_43044" predecessor="ID_82741" successor="ID_89512" />
  <ResourceRelease id="ID_31896" release="ID_19639" predecessor="ID_26371" successor="ID_64578" />
  <Start id="ID_74151" target="ID_12307"/>
  <End id="ID_89512" source="ID_19639"/>
  <End id="ID_23745" source="ID_64578"/>
  <Sequence id="ID_76309" source="ID_64578" target="ID_19639"/>
  <Sequence id="ID_31896" source="ID_26371" target="ID_64578"/>
  <Fork id="ID_22801" source="ID_81794" target="ID_40707 ID_65300"/>
</Scenario>

<Scenario id="ID_27474" name="NOT OK STEP ID_46083">
  <Step id="ID_65780" name="alarm" predecessor="ID_42534" successor="ID_82855"/>
  <ResourceAcquire id="ID_72104" acquire="ID_27246" rUnits="1.0" predecessor="ID_64057" successor="ID_42534" acquire="ID_45671" rUnits="1.0"/>
  <ResourceRelease id="ID_37703" release="ID_27246" predecessor="ID_82855" successor="ID_16350" release="ID_27246" target="ID_65780"/>
  <Start id="ID_64057" target="ID_72104"/>
  <End id="ID_16350" source="ID_37703"/>
  <Sequence id="ID_42534" source="ID_72104" target="ID_65780"/>
  <Sequence id="ID_82855" source="ID_65780" target="ID_37703"/>
</Scenario>

<Scenario id="ID_45015" name="need to log_STEP ID_74970">
  <Step id="ID_9666" name="logEvent" predecessor="ID_33129" successor="ID_25786"/>
  <Step id="ID_91726" name="writeEvent" predecessor="ID_44477" successor="ID_48866"/>
  <Step id="ID_90134" name="writeRec" predecessor="ID_1686" successor="ID_90134"/>
  <ResourceAcquire id="ID_92339" acquire="ID_85206" rUnits="1.0" predecessor="ID_25786" successor="ID_8213" acquire="ID_8213" rUnits="1.0"/>
  <ResourceAcquire id="ID_1686" acquire="ID_45671" rUnits="1.0" predecessor="ID_92339" successor="ID_48866" acquire="ID_93788" rUnits="1.0"/>
</Scenario>