# Transformation of UML 2.0 Models Extended with MARTE to Core Scenario Models

by

**Hui Liu**

**A thesis submitted to the Faculty of Graduate Studies**

**in partial fulfillment of the requirement for the degree of**

**Master of Science in Information and System Science**

**Department of Systems and Computer Engineering**
**Carleton University**
**Ottawa, Canada**

**April 2008**

*Your file Votre référence*
*ISBN: 978-0-494-40659-5*
*Our file Notre référence*
*ISBN: 978-0-494-40659-5*

# 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) or its successor UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). 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 metamodel 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 my parents, who have persistently given me love, hope and their unconditional support and infinite encouragement, without which I can do nothing. I love you from the bottom of my heart.

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.

Finally I would like also like to thank my friends for all their help and support: AK, AG, PYS, BL and HYG.

# Index of Content

# List of Figures

# List of Tables

# Chapter1. Introduction

## 1.1 Motivation and Goals

Unified Modeling Language (UML) is the result of the convergence of notations from three main object-oriented methods, OMT (by James Rumbaugh), OOSE (by Ivar Jacobson) and Booch (by Grady Booch) [1]. As a standardized specification language for object modeling, UML is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system, referred to as a UML model. UML is formally defined at the Object Management Group (OMG) by the means of the UML metamodel, which is expressed in turn in a language for defining metamodels known as Meta-Object Facility (MOF). The purpose of UML is to specify, visualize, construct, and document software-intensive systems. UML provides 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.

In software engineering, performance engineering is gaining more attention, as performance of applications is often found below expectations. Performance validation is in many cases one of the last activities before the applications are released. However, the cost for doing performance analysis only at the end of the development cycle can be considerable and could cause severe performance disasters in software projects. It has been recognized that performance analysis needs to be integrated throughout the software life-cycle [2]. To achieve this, earlier performance analysis requires the use of models

for insight into the sources of problems. Rapid inexpensive modeling techniques that integrate performance into software analysis are needed [3].

To bridge the gap between software analysis and performance analysis, OMG introduced the UML Profile for Schedulability, Performance, and Time (SPT) [4], and its successor, UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) for UML2 [5]. The profiles add light-weight extensions to the UML metamodel to express the basic abstractions used in performance analysis. Adding quantitative performance annotations to UML models enables their transformation into performance models, which can be analyzed with existing performance analysis tools. The performance analysis results can be fed back to the software engineers, as well as to other stockholders, at an earlier stage of the development, allowing them to improve the models as necessary.

There are various performance models used in the research community and in industry, such as Layered Queuing Networks (LQN) [6], Extended Queuing Networks (EQN) [7], Stochastic Timed Petri Nets (STPN) [8], Stochastic Process Algebra (SPA) [9] 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 life cycle of the system development. 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 them 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. Both the SPT Profile [4] and MARTE define scenarios as a composition of steps, with precedence relations between them; each step can be a sub-scenario. The UML 2.0 standard [10] 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 realization providing the means for the end user and its developers to state the expectations about the performance of the system.

Scenarios are often modeled using use case diagrams, interaction diagrams, activity diagrams or communicating 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 diagram gives the description of the inter-object behavioral aspect of a system, showing how objects collaborate to realize a scenario. 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 Research Scope of the Thesis

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 a standard performance profile, as shown in Figure 1-1.

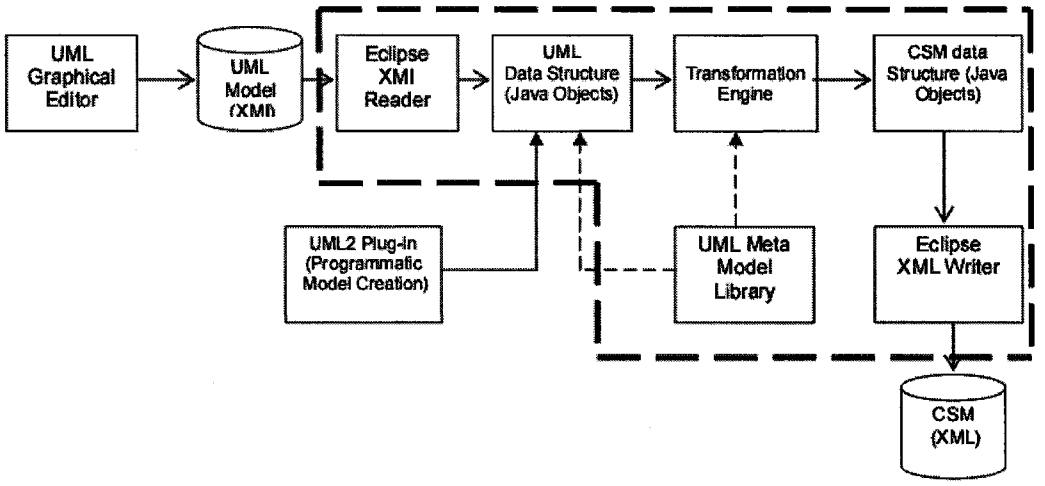



**Figure 1-1 UML 2.0 Model to CSM**

Part of this thesis responsibility is to accept an annotated UML model composed of class, deployment and activity 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 [11], or programmatically with an Eclipse plug-in named UML2 Plug-in [12]. 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 the transformation engine can be used. 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 any UML graphical tools.

## 1.3 Thesis Contributions

The contributions of the thesis are as follows:

- Propose a set of algorithms for the automatic transformation of a UML 2.0 model (represented as class, deployment and activity diagrams) annotated with MARTE performance information into Core Scenario Models. The transformation rules and

algorithm proposed here take into account the characteristics of the behaviour model and its contexts, and create a CSM for every scenario described in activity diagrams.

- Design, implement and test a Java application running on top of Eclipse that achieves the above transformation from UML 2.0 + MARTE 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. Software performance engineering is introduced next along with the Performance Models and Performance Profile.

Chapter 3 consists of the discussion of the source and target models of transformation. The former section focuses 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. The latter one would discuss Core Scenario Models.

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.

Implementation and Verification is presented in Chapter 5, which explains the implantation design and verifies features of the implementation with test cases.

A Case Study is presented in Chapter 6, which deals with a specific example. The internal structure of the UML 2.0 models, generated CSM and verification and testing are discussed. Chapter 7 gives the conclusion of the thesis research and discusses possibilities of future work.

# Chapter 2. Background Literature

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 XML Metadata Interchange (XMI), OMG (Object Management Group)'s MetaObject Facility (MOF), and Software Performance Engineering, including performance models and intermediate models.

## 2.1 Model Driven Architecture

Model-driven architecture is a software development approach launched by the OMG in 2001. MDA supports model-driven engineering of software systems. MDA provides a set of guidelines for structuring specifications expressed as models. The MDA model is related to multiple standards, including the Unified Modeling Language (UML), the Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Enterprise Distributed Object Computing (EDOC), the Software Process Engineering Metamodel (SPEM), and the Common Warehouse Metamodel (CWM).

### 2.1.1 OMG Standards for MDA

#### MOF

The Meta-Object Facility (MOF) is an Object Management Group (OMG) standard for Model Driven Engineering. MOF originated in the Unified Modeling Language (UML); the OMG was in need of Metamodeling architecture to define the UML.

MOF is a closed metamodeling architecture; it defines the M3-model (see Fig. 2-1), which conforms to itself. MOF allows a strict meta-modelling architecture; every model element on every layer is strictly in correspondence with a model element of the layer above. MOF only provides a means to define the structure, or abstract syntax of a language or of data. For defining metamodels, MOF plays the same role that EBNF plays for defining programming language grammars. MOF is a Domain Specific Language (DSL) used to define metamodels, just as EBNF is a DSL for defining grammars. Similarly to EBNF, MOF could be defined in MOF.

In short MOF uses the notion of MOF::Classes (not to be confused with UML::Classes), as known from object orientation, to define concepts (model elements) on a metalayer. MOF may be used to define metamodels for object-oriented modeling languages (such as UML) as well as for non object-oriented ones (such as Petri Net or Web Services).

## UML

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 [13][14][15] and has almost completely absorbed not only these methodologies but also OORAM [16], Syntropy [17] 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 [10] is used throughout the thesis. UML 2.0 standard is described in two documents -- UML2.0 Infrastructure [18] and UML 2.0 Superstructure [10].

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 the UML metamodel composed of a set of UML packages. These packages contain a set of meta-classes that define the language 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 extension mechanisms:

*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 2.2.3.

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 in which UML is defined.

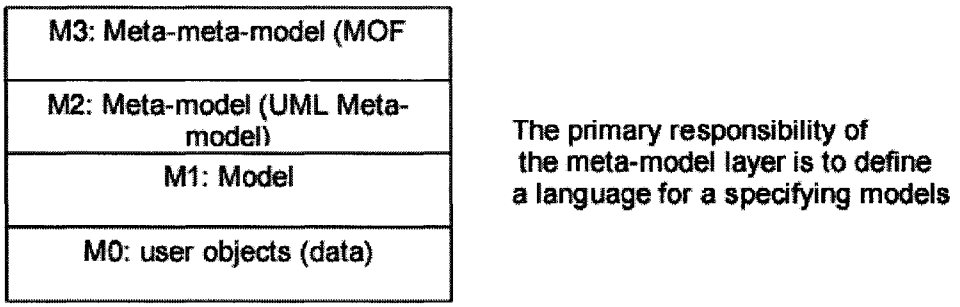



**Figure 2-1.1 UML 2.0 meta-model architecture**

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 activity, class and deployment diagrams are discussed.

## XMI

XML is the eXtensible Markup Language. This 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 design to complement HTML. XML is a way to work with information in a structure form.

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

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 defined 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 usage, makes it a good choice to store large amounts of data.

*Free Style:* XML, a free style language, can be processed by various style sheets to produce outputs in different formats such as PDF, TEX, or some other format.

*Well-Formed:* Because a 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 users the option to choose among several XML parsers depending on the simplification process of the XML data.

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 [21 ]
- UML -- Unified Modeling Language, an OMG Modeling Standard [10 ]
- MOF -- Meta Object Facility, a OMG meta-modeling and meta-data repository standard [42]

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 is 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.

- Rules described in DTDs(Document Type Definitions) and Schemas are used to validate XMI documents .
- It is possible to produce 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.

A notable limitation of the use of XMI by UML tools is the fact that each tool builder adds non-standard information when they serialize the UML models to XMI, they practically limits the exchangeability of UML models in XMI format between different UML tools.

### 2.1.2 Model Transformations

In MDE model transformations play a key role. The MDA guide [19] defines a model transformation as "the process of converting one model to another model of the same system". Kleppe et al. [20] defines a transformation as the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed to a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed to one or more constructs in the target language. In other words, a model transformation program takes as input a model conforming to a given source meta-model and produces as output another model conforming to a target meta-model. The transformation program, composed of a set of rules, should itself be considered as a model. As a consequence, it is based on a corresponding meta-model, that is an abstract definition of the used transformation language.

In MDA, QVT (Queries/Views/Transformations) is a standard for model transformation defined by the Object Management Group. QVT defines three DSL (Domain Specific Languages) named Relations, Core and OperationalMappings. These languages are organized in a layered architecture. Relations and Core are declarative languages at two different levels of abstraction, with a normative mapping between them. The Relations language has a graphical concrete syntax. The QVT/OperationalMapping language is an imperative language that extends both QVT/Relations and QVT/Core. The syntax of the QVT/OperationalMappings language provides constructs commonly found in imperative languages (loops, conditions, etc.) [43]. Figure shows the QVT Architecture:

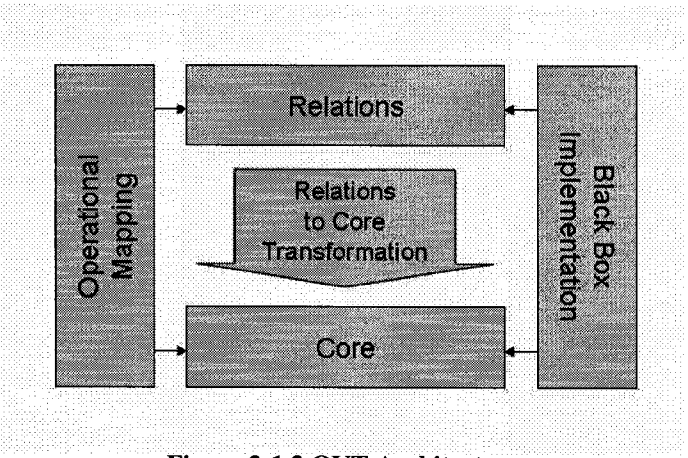



**Figure 2-1.2 QVT Architecture**

Although there are standards for model transformation, this thesis does not adopt QVT or other specific transformation languages to express the transformation. The main reason is that when we started working on the thesis, there were no QVT tools that we could use for our implementation. However, this would be an item for future work.

## 2.2 Software Performance Engineering

### 2.2.1 Concepts

Software Performance Engineering (SPE), firstly presented in [2], is a systematic, quantitative approach to construct software systems that meet performance objectives. It is based on the careful and methodical assessment of performance attributes throughout the lifecycle, from requirements and specification to implementation and maintenance [22].

The SPE basic concept is the separation of the Software Model (SM) from its environment or Machinery Model (MM). The SM captures the essential aspects of software behavior; it can be represented by means of Execution Graphs (EGs). An EG is a graph whose nodes represent software workload components and whose edges

represent transfers of control. Each node is weighted by use of a demand vector that represents the resource usage of the node (i.e., the demand for each resource). The MM models the hardware platform and is based on performance models, which we would discuss later. Taking QN as an example, we need to define: the components (i.e., the service centers), the topology (i.e., the connections among centers) and some relevant parameters (such as job classes, job routing among centers, scheduling discipline at service centers, service demand at service centers). Component and topology specification is performed according to the system description, while parameters specification is obtained from information derived by EGs and from knowledge of resource capabilities. This distinction, on the one hand, allows for defining SMs and MMs separately and solving their combination, on the other improves the portability of the models (e.g., the performance of a specific software system can be evaluated on different platforms, and the performance of a specific platform can be validated under different software systems). After having established the performance objectives, the SMs and MMs are constructed and hence evaluated by a suitable analysis method to compute the performance prediction.

### 2.2.2 Performance Modeling

During performance engineering, four activities occur within different specified phases [23]: Inception, Elaboration, Construction and Transition. Performance modeling would be normally performed during elaboration. Amongst the various modeling techniques used in common practice, the three most common ones are described below:

**Queueing Models.** This well known 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)[44].

**Layered Queueing Network:** LQN was developed as an extension of QN model .With respect to traditional QN, in LQN a server, to which customer requests arrives and queue for service, may become a client to other servers from which it requires nested services while serving its own clients. An LQN model is represented as an acyclic graph whose nodes (named also tasks) are software entities and hardware devices, and whose arcs denote service requests. The nodes with outgoing and no incoming arcs play the role of pure clients. The intermediate nodes with incoming and outgoing arcs play both the role of client and of server, and usually represent software components. A software or hardware server node can be either a single-server or a multi-server. A LQN task may offer more than one kind of service. An entry has its own execution time and demands for other services (given as model parameters). Each server has an implicit message queue, where the incoming requests are waiting their turn to be served. Servers with more than one entry still have a single input queue, where requests for different entries wait together. The default scheduling policy of the queue is FIFO, but other policies are also supported.

**Petri Nets.** Petri net is a graphical and mathematical modeling and analysis language which consists of places, transitions, and arcs. They were introduced by Carl Adam Petri

in 1962 at the Technische University Darmstadt, Germany. Transitions are active components. They model activities which can occur, thus changing the state of the system. Transitions are only allowed to fire if they are enabled, which means that all the preconditions for the activity have been fulfilled. Places are place holders for tokens. The current state of the system being modeled is called marking which is given by the number and type (if the tokens are distinguishable by type) of tokens in each place [24]. Arcs are of two types: input and output. Input arcs start from a places and ends at a transitions, while output arcs start at a transition and end at a place. When the transition fires, it removes tokens from its input places and adds some at all of its output places. The number of tokens removed / added depends on the cardinality of each arc. The interactive firing of transitions in subsequent markings is called token game. Petri nets are a formal language for describing and studying systems that are characterized as concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. As a graphical tool, Petri nets can be used as a visual communication aid similar to flow charts, block diagrams, etc. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. The use of Petri Nets leads to a mathematical description of the system structure that can then be investigated analytically. It is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. Timed Petri nets can be used for studying the performance and dependability issues of systems. Petri nets can be used for analyzing properties like reachability, boundedness, liveness, persistence, fairness etc [25].

### 2.2.3 Profiles for Performance Annotations

A profile in the Unified Modeling Language provides a generic extension mechanism for building UML models in particular domains. Profiles are based on additional stereotypes and tagged values that are applied to elements, attributes, methods, links, and link ends. A profile is a collection of such extensions and restrictions that together describe some particular modeling problem and facilitate modeling constructs in that domain. UML profiles tailor the language to specific areas: business modeling and others.

The solution proposed by OMG to the problem of extending the power of expression of UML for different domains is to define standard UML profiles. One important profile able to add annotations for non-functional characteristics is the "UML Profile for Schedulability, Performance, and Time Specification" (SPT) defined for UML 1.4 [4]. In order to upgrade SPT to UML 2.0 and extend its scope with Real-Time and Embedded System modeling capabilities, a new UML Profile named MARTE ("Modeling and Analysis of Real-Time and Embedded Systems") has been recently adopted by OMG [5].

Two standard profiles, SPT and MARTE contain sub-profiles for performance annotations. Performance annotations are required to bridge the gap between the domain of software development and of performance analysis. As such they should be usable by software designers, and, at the same time, they also must support the performance analysis concepts.

**SPT 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 the UML profile for Schedulability, Performance and Time. This profile defines standard paradigms for the

modeling of time, schedulability, and performance related aspects of real time systems. SPT 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.

**MARTE Profile**: In spite of the fact that the OMG has adopted the UML Profile for SPT to model real-time concerns, a lack of flexibility has limited a broad adoption by the RTES community. The need for major modifications [26], evolution of standards and the need to be in compliance with other relevant OMG standards (e.g., UML 2) led to a Request For Proposals (RFP) for a new UML Profile named MARTE (Modeling and Analysis of Real-Time and Embedded systems) [27].

The UML MARTE profile extends the scope of SPT as follows:

- Modeling both software and hardware aspects
- Modeling of platform, platform-independent, and their allocation viewpoints in a MDA style approach [28]
- Compliance with the UML Profile for Modeling Quality of Service and Fault Tolerance (QoS & FT) [29]
- Specification not only of real-time constraints, but also of other QoS characteristics for embedded systems, such as power consumption and memory size

- Modeling of embedded, reactive, control/command, and intensive data flow computational systems
- Component-based architectures modeling and analysis
- Capability to represent Asynchronous/Causal, Synchronous/Clocked and Real/Continuous time models.

Specifically, two MARTE packages, GRM and PAM, are discussed here since they are more related to the thesis.

**Generic Resource Modeling (GRM).** The generic resource model (GRM) includes the features which are required for dealing with:

- Modeling of executing platforms at different level of details. The level of granularity needed for platform modeling depends on the concern motivating the description of the platform, as for example the type of the platform, the type of the application, or the type of analysis to be carried out on the model.
- Modeling of both "hardware" (e.g., memory units or physical communication channels) and "software" (e.g., real-time operating systems) platform.

The central concept of the GRM is the notion of a Resource. A Resource represents a physically or logically persistent entity that offers one or more ResourceServices. Resources and its services are the available means to perform the expected duties and/or satisfy the requirements for the system under consideration.

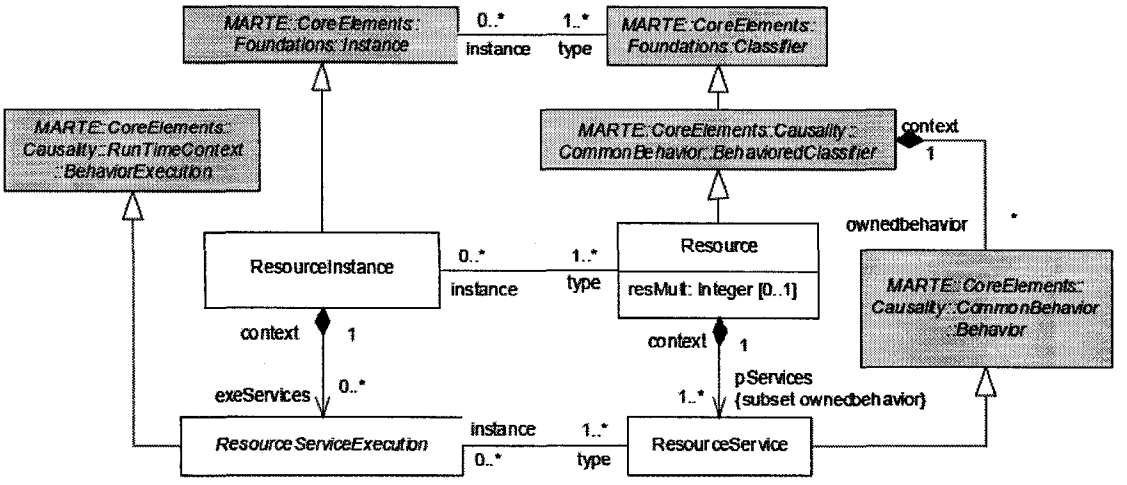



**Figure 2-2 Instance/Classifier nature of core resource elements in GRM**

The different facets of the GRM are grouped in individual packages, following the structure shown in Figure2-3

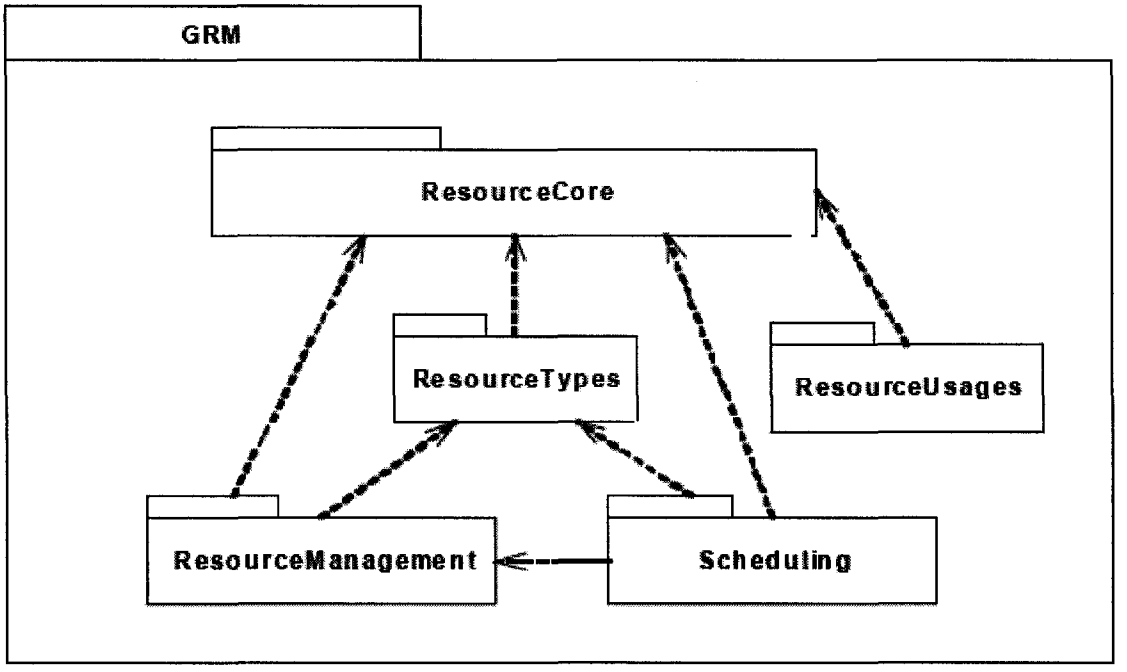



**Figure 2-3 Architecture of the GeneralResourceModel (GRM) package**

Figure 2-4 presents the basic resource types defined along with their specific attributes.

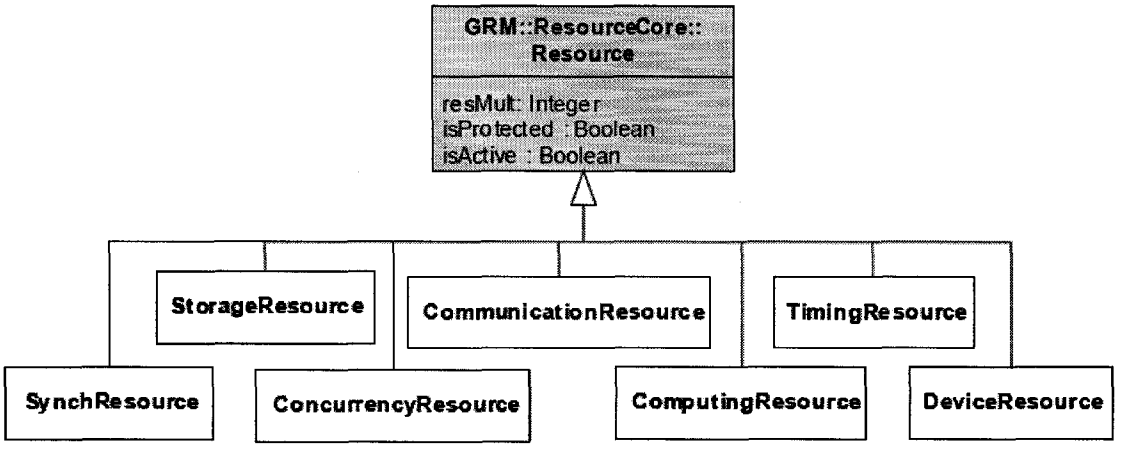



**Figure 2-4 Types of resources in the ResourceTypes package**

**Performance Analysis Modeling (PAM).** In MARTE, "performance modeling" describes the analysis of temporal properties of best-effort systems and soft-real-time embedded systems, including systems supplying information, web-based services, enterprise services, multimedia, telecommunications, and networked services. Performance measures (analysis outputs) are statistical, such as mean throughput and delay, or the probability of missing a target response time. Input parameters to the analysis may also be probabilistic, such as a random arrival process, random execution time for a media frame, or a probability of a cache hit. The common performance analysis techniques include simulations, extended queueing models, and discrete-state models such as Stochastic Petri Nets. For the purposes of analysis, behaviour is often considered as non-terminating ( for example, steady state behaviour for capacity analysis).

Performance analysis includes single-case analysis for a given set of input parameters, or multicase analysis such as:

- sensitivity analysis which explores a parameter space, to find ideal operational parameters or to identify risky workload situations. Sensitivity analysis may also include alternative scenarios, platforms, physical deployments, and configurations.
- capacity analysis which explore the capabilities of the design or configuration.

Performance analysis is determined by how the system behaviour uses system resources. Important resources include hardware ExecutionEngines, concurrent process threads (ScheduledResources), and LogicalResources defined by the software. A logical resource can be any entity to which the software requires access, and for which the program may have to wait at some point. In this sense, a semaphore, a lock, a buffer, a block of memory are resources. A pool of access control tokens can be modeled as a logical resource.

A process resource, or pool of process threads, is also a kind of logical resource which is modeled separately, by the concept of SchedulableResource imported from the General Resource domain model (GRM). Because processes may be identified in behaviour specifications by other entities (lifelines and swimlanes in particular), a special concept of RunTimeObjectInstance is introduced to represent an alias for a process resource.

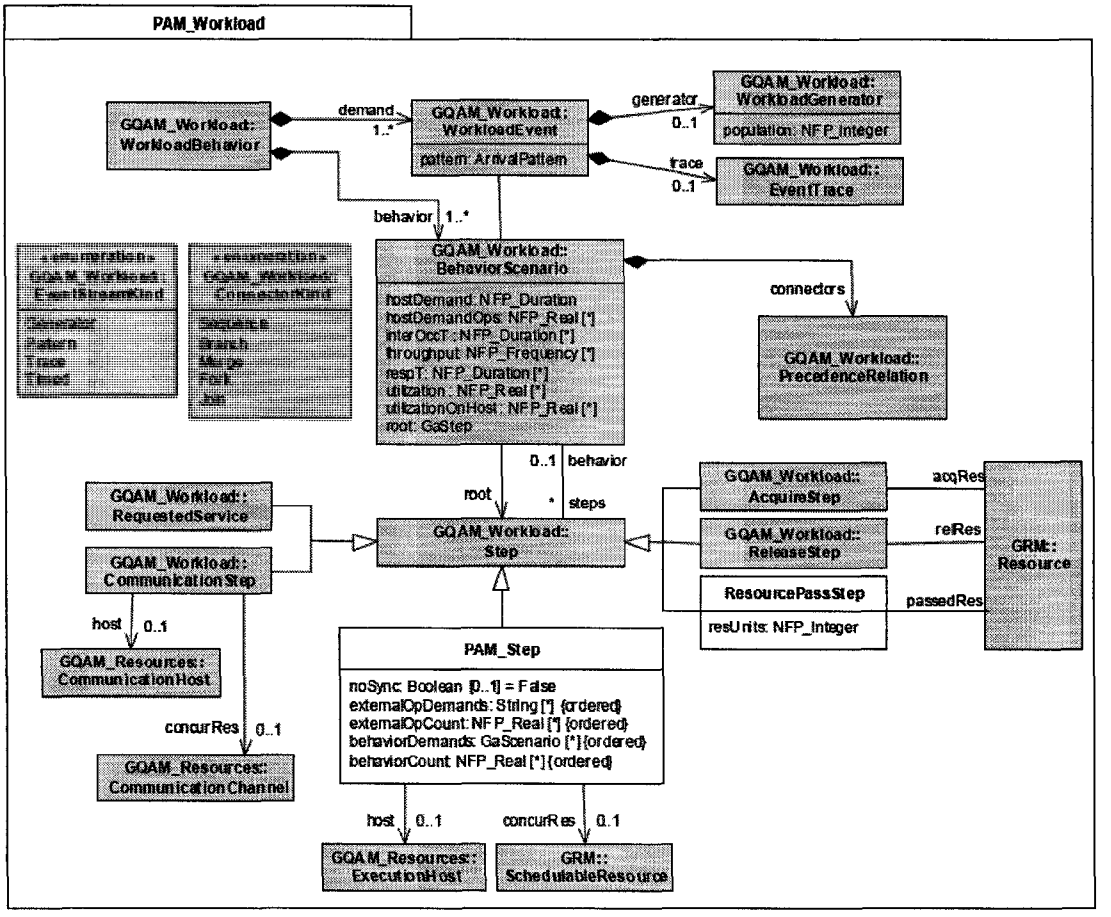



**Figure 2-5 Part of the performance domain model relevant to behavior**

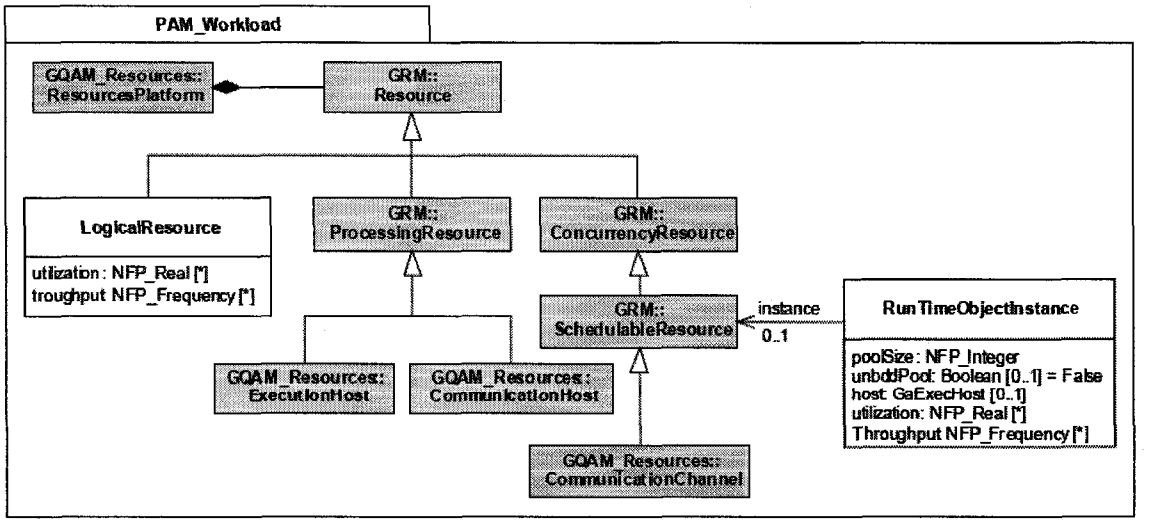



**Figure 2-6 Part of the performance domain model relevant to resources**

## 2.3 From Software Models To Performance Models

### 2.3.1 Literature Survey

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 good survey of the techniques for deriving performance models from UML models is given in [30]. The technique from [31] follows the SPE methodology very closely, generating the same kind of models as in [2]. The work from [33] introduces an UML-based notation and framework for describing performance models. In [34] UML models are transformed into Petri Nets, but the contention for hardware resources is not considered yet. [35] 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 [36] uses an existing graph-rewriting tool PROGRES [37]. Another solution presented in [38] implements in Java an ad-hoc graph transformation at the UML 1.4 metamodel level; the third solution given in [39] uses Extensible Stylesheet Language Transformation. A two-phase transformation from UML to LQN using XML algebra concepts and implemented in XSLT was presented in [40].

### 2.3.2 PUMA

Another ongoing research project at Carleton University, named PUMA [41] 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, leaving away the design details irrelevant to such analysis. The advantage of CSM is evident when one considers adding a new transformation from UML to another performance formalism. It is much simpler to translate from CSM to a new performance model, than directly from UML. The present thesis is a part of the larger project PUMA. Figure2-7 shows the complete tool architecture, including not just evaluation, but also exploration of the performance properties of the design, and feedback to the design domain.

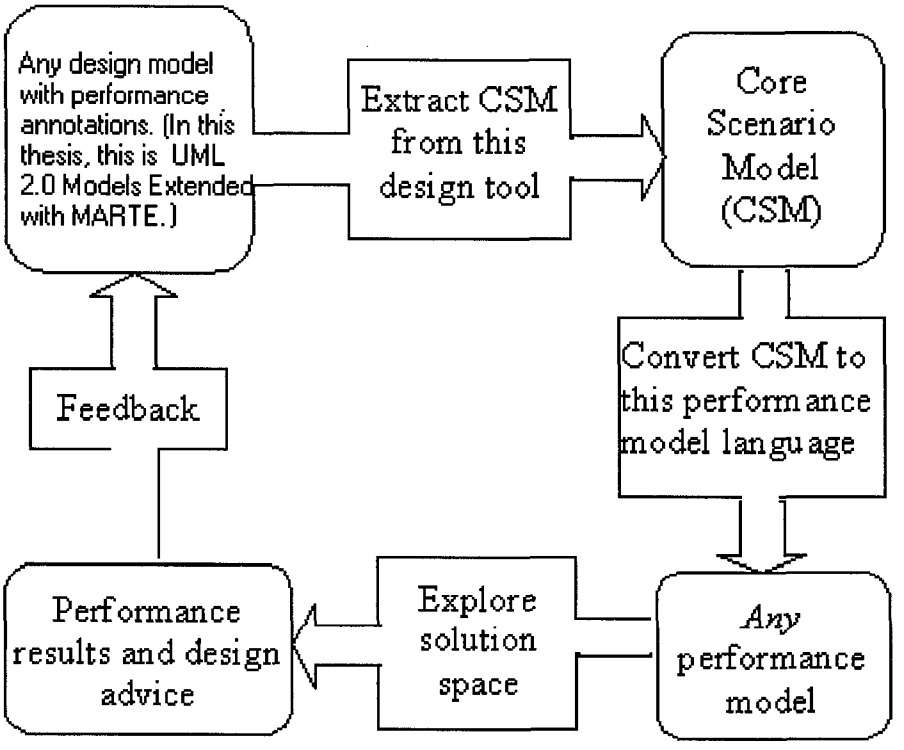



**Figure 2-7 PUMA Architecture**

# Chapter 3. Source and Target Models

## 3.1 Source Model: UML 2.0

The source model for the transformation proposed in the thesis is UML 2.0 extended with the MARTE profile. In this section are presented only the parts of the UML 2.0 metamodel that are relevant to the transformation described in chapter 4.

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 [18] for the Infrastructure and [10] for the Superstructure. (The current versions of UML standards follow UML Superstructure v. 2.1.2, UML Infrastructure v. 2.1.2. However, since most tools that we are using are based on UML2.0, we use this version as the source model.)

UML 2.0 is divided into three major parts -- structural, behavioural and supplement. The structural part discusses the meta-model regarding the static, structural constructs such as classes, components, packages, etc. used in various structural diagrams. The behavioural part goes onto describing constructs for describing behaviour, 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 domains.

The structural part of UML 2.0 is described by the following diagrams as illustrated in Figure3-1: Class Diagram, Composite Structure Diagram, Component Diagram,

Deployment Diagram, Object Diagram, and Package Diagram. The behavioural aspects are 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.

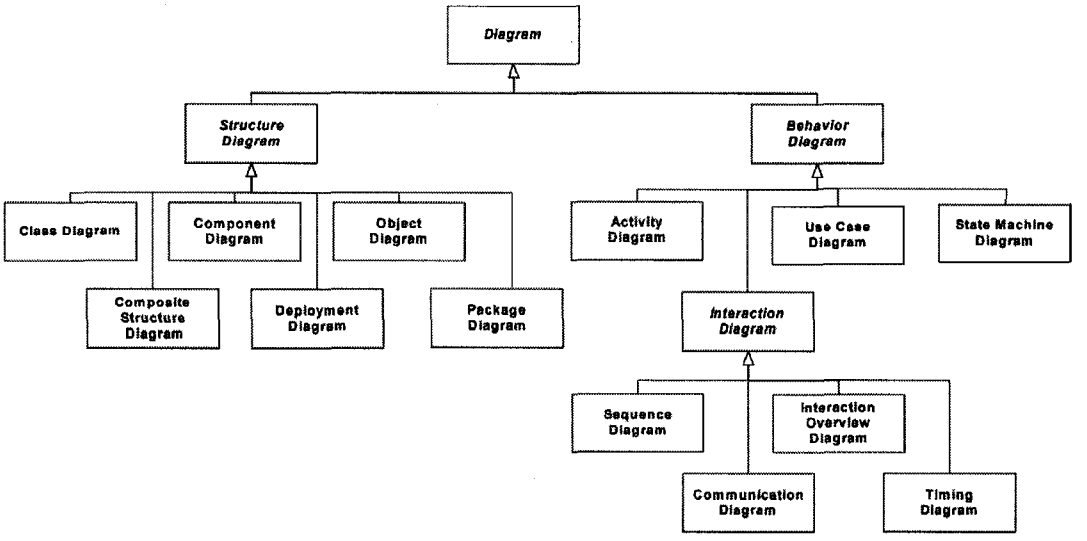



**Figure 3-1 UML 2.0 Diagrams**

### 3.1.1 Structural View

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

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.

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.

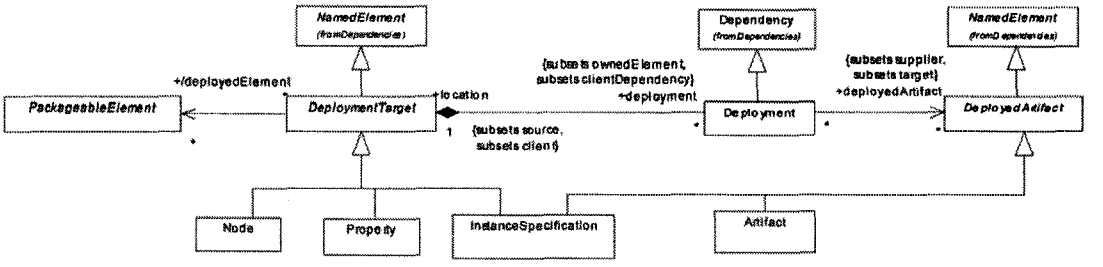



**Figure 3-2 Meta-model of UML 2.0 Deployment Diagram**

A component diagram depicts how a software system is split up into physical components and shows the dependencies among them. Component diagrams can be used to model and document any system architecture. A component represents a modular part of a system that encapsulates its contents and whose manifestation can be replaced with another component. A connector kind attribute is added to the Connector metaclass. Its value is an enumeration type with valid values "assembly" or "delegation".

### 3.1.2 Behavioral View

Dynamic and behavioural constructs (e.g., activities, interactions, state machines) used in various behavioural diagrams, such as activity diagrams, sequence diagrams, and state machine diagrams. The UML packages that support behavioural modeling, along with the structure packages they depend upon (CompositeStructures and Classes) are shown in Figure 3-3 below.

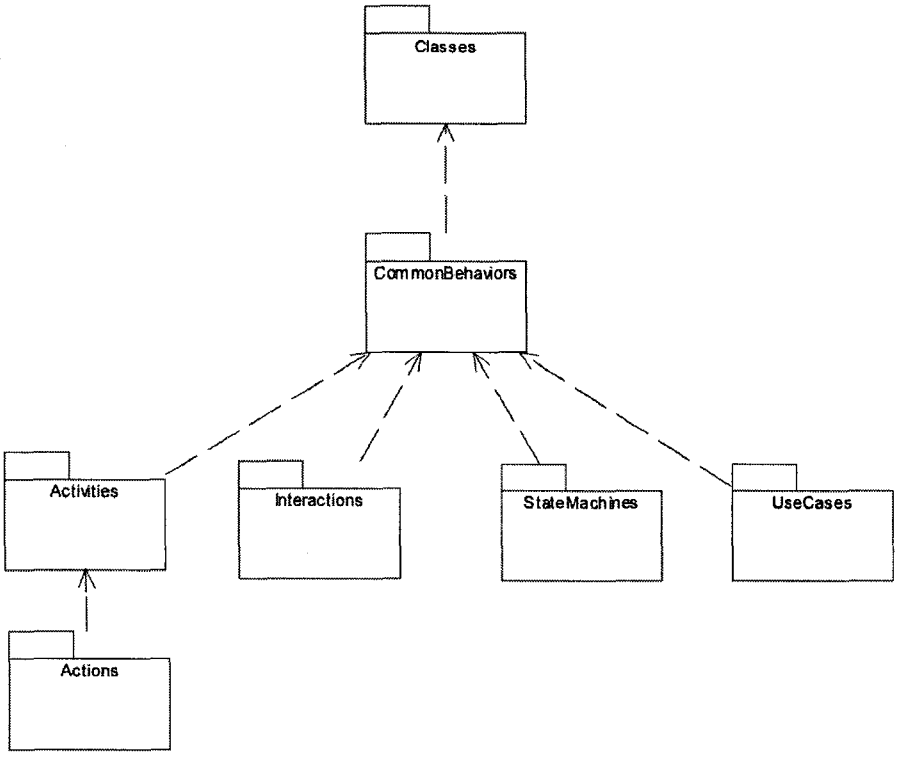



**Figure 3-3 The UML packages that support behavioural modeling**

In this thesis, from the Behavioral Part of the UML 2.0 Specifications, only the Activity Diagram is used for the transformation to Core Scenario Model.

Activity diagram emphasizes the sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors. These are commonly called control flow and object flow models. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because events occur external to the flow.

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

**Activity:** An activity specifies the coordination of executions of subordinate behaviors, using a control and data flow model. The subordinate behaviors coordinated by these models may be initiated because other behaviors in the model finish executing, because objects and data become available, or because events occur external to the flow. The flow of execution is modeled as activity nodes connected by activity edges. A node can trigger the execution of a subordinate behavior, such as an arithmetic computation, a call to an operation, or manipulation of object contents. Activity nodes also include flow-of-control constructs, such as synchronization, decision, and concurrency control. Activities may form invocation hierarchies invoking other activities, ultimately resolving to individual actions. In an object-oriented model, activities may be invoked indirectly as methods bound to operations that are directly invoked.

**Activity Node:** An activity node is an abstract class for the steps of an activity. It covers executable nodes, control nodes, and object nodes. The following meta-classes are sub-classes of Activity Node related to the transformation.

**Control Node**: A control node is an abstract activity node that coordinates flows in an activity. It covers initial node, final node and its children, fork node, join node, decision node, and merge node.

**Initial Node:** An initial node is a control node at which flow starts when the activity is invoked. A control token is placed at the initial node when the activity starts, but not in initial nodes in structured nodes contained by the activity. Flows can also start at other nodes, so initial nodes are not required for an activity to start execution.

**Final Node:** A final node is an abstract control node at which a flow in an activity stops. All tokens offered on incoming edges are accepted. See children of final node for other semantics.

**Activity Final Node:** An activity may have more than one activity final node. The first one reached stops all flows in the activity.

**Flow Final Node:** A flow final node is a final node that terminates a flow. A flow final destroys all tokens that arrive at it. It has no effect on other flows in the activity.

**Fork Node, Join Node, Decision Node, and Merge Node**: A fork node splits a flow into multiple concurrent flows. A join node synchronizes multiple flows. A decision node chooses between outgoing flows. A merge node brings together multiple alternate flows. It is not used to synchronize concurrent flows but to accept one among several alternate flows.

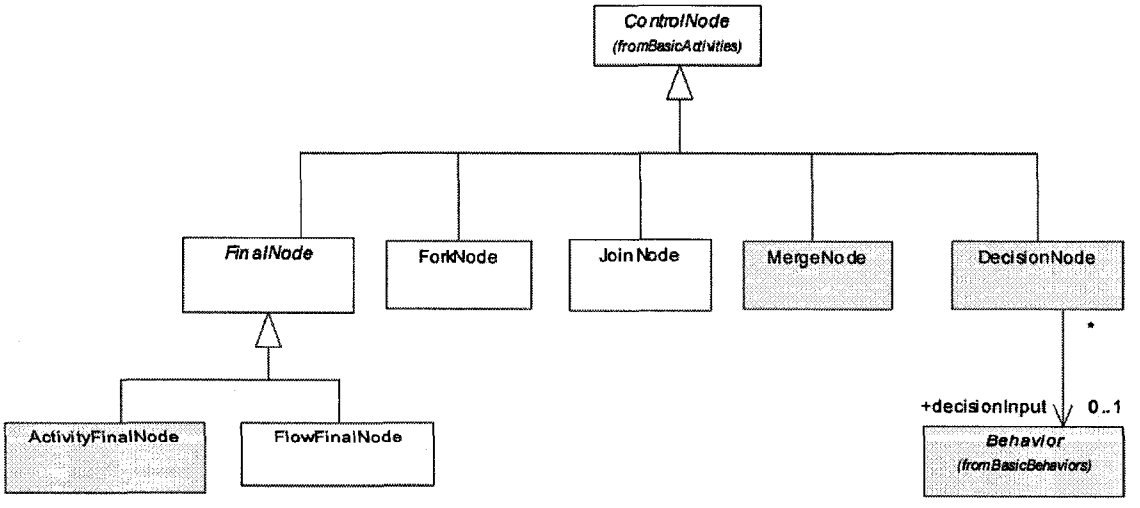



**Figure 3-4 ControlNode and its subclasses**

**Send Signal Action:** SendSignalAction is an action that creates a signal instance from its inputs, and transmits it to the target object, where it may cause the firing of a state machine transition or the execution of an activity. The argument values are available to the execution of associated behaviors. The requestor continues execution immediately. Any reply message is ignored and is not transmitted to the requestor.

**Accept Event Action:** AcceptEventAction is an action that waits for the occurrence of an event meeting specified condition. If an AcceptEventAction has no incoming edges, then the action starts when the containing activity or structured node does, whichever most immediately contains the action. In addition, an AcceptEventAction with no incoming edges remains enabled after it accepts an event. It does not terminate after accepting an event and outputting a value, but continues to wait for other events. This semantic is an exception to the normal execution rules in Activities. An AcceptEventAction with no incoming edges and contained by a structured node is terminated when its container is terminated.

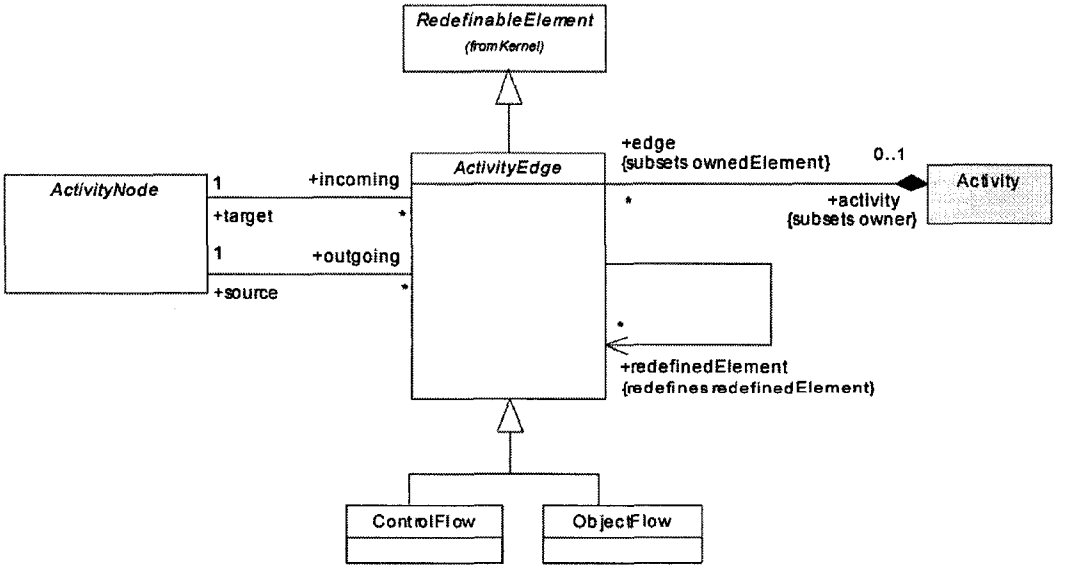



**Figure 3-5 Class Diagram Centered by ActivityEdge**

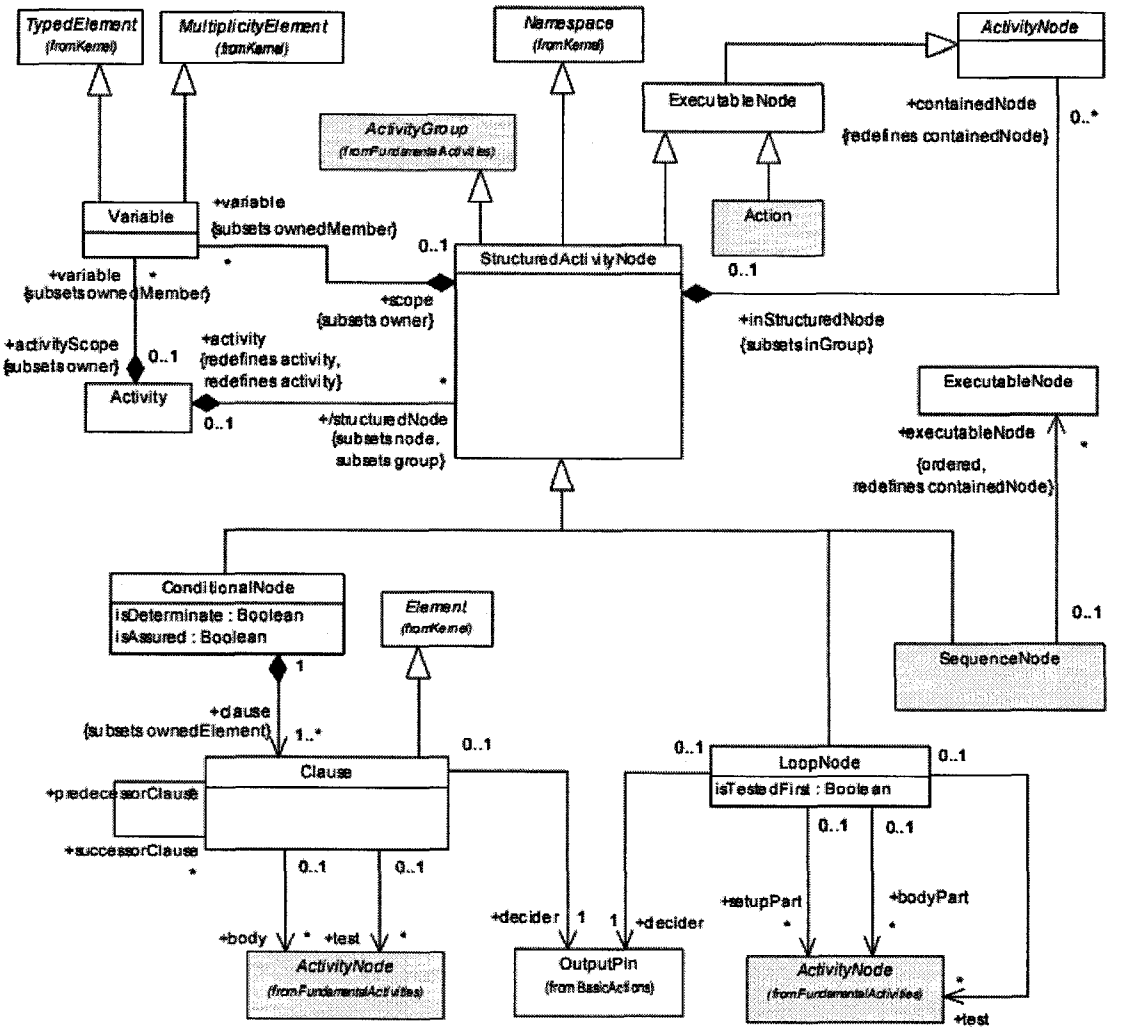



**Figure 3-6 Class Diagram Centered by StructuredActivityNode**

**Activity Edge:** An activity edge is an abstract class for directed connections between two activity nodes. Activity edges are directed connections, that is, they have a source and a target, along which tokens may flow.

**Activity Partition:** An activity partition is a kind of activity group for identifying actions that have some characteristic in common. Partitioning activity diagrams into "swim lanes" is a typical and practical usage of activity partition. These swim lanes represent objects that are responsible for execute the activity nodes and edges in their own partition.

**Structured Activity Node:** A structured activity node is an executable activity node that may have an expansion into subordinate nodes as an ActivityGroup. The subordinate nodes must belong to only one structured activity node, although they may be nested.

### 3.1.3 Rational Software Architect and UML2 Plug-in

IBM Rational Software Architect (RSA) is a design and development tool which allows software architects and developers within a development team to specify and maintain all aspects of an application's software architecture [11]. 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. It 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 supports 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 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 code generation patterns used in 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 feature is very useful since this can produce the meta-objects of any given diagrams for further transformations.

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 UML2 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 user requirements. Although this is all a Java-based implementation, the files are all stored in XML format for easier interoperability.

### 3.1.4 Assumptions about the Source Model

Due to research limitation and performance analysis considerations, the proposed transformations do not work on the whole set of UML models. Instead, only the input

models satisfying the following assumptions are guaranteed to give the expected transformation results.

**a) The UML model should contain the following information**: high-level software architecture showing the main components, their relationships and deployment on hardware resources, as well as scenario models in the form of annotated activity diagrams.

**b) Every partition corresponds to a software resource.** An activity partition is a kind of activity group containing actions that have some characteristic in common. However, this research adopts the swim-lane approach, which means that the activities in one partition are executed on one the same software resource (i.e., process or thread).

**c) One-dimensional partition**. In general, UML partitions could be multi-dimensional. However, in this research, only one-dimensional partitions are assumed. More specifically, every activity node must belong to one and only one partition.

**d) Specifying the "owner" partition for every activity node**. In RSA, for some reasons, actions are automatically assigned to the partition in which the actions are drawn in, while the control nodes, including initial node, fork node, etc., are not. However, since we assume every activity node must be in one and only one partition, we emphasize this assumption

## 3.2 Target Model: CSM

### 3.2.1 Core Scenario Model (CSM)

Core Scenario Models (CSM) [32] 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.

**Behavioural Part of CSM**

CSM metaclass contains Scenarios metaclass. 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 the 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 rival coming from the environment or they can be a "closed workload" where there is 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 metaclass Start.

A step has a Refinement meta-class. A Refinement metaclass 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.

**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).

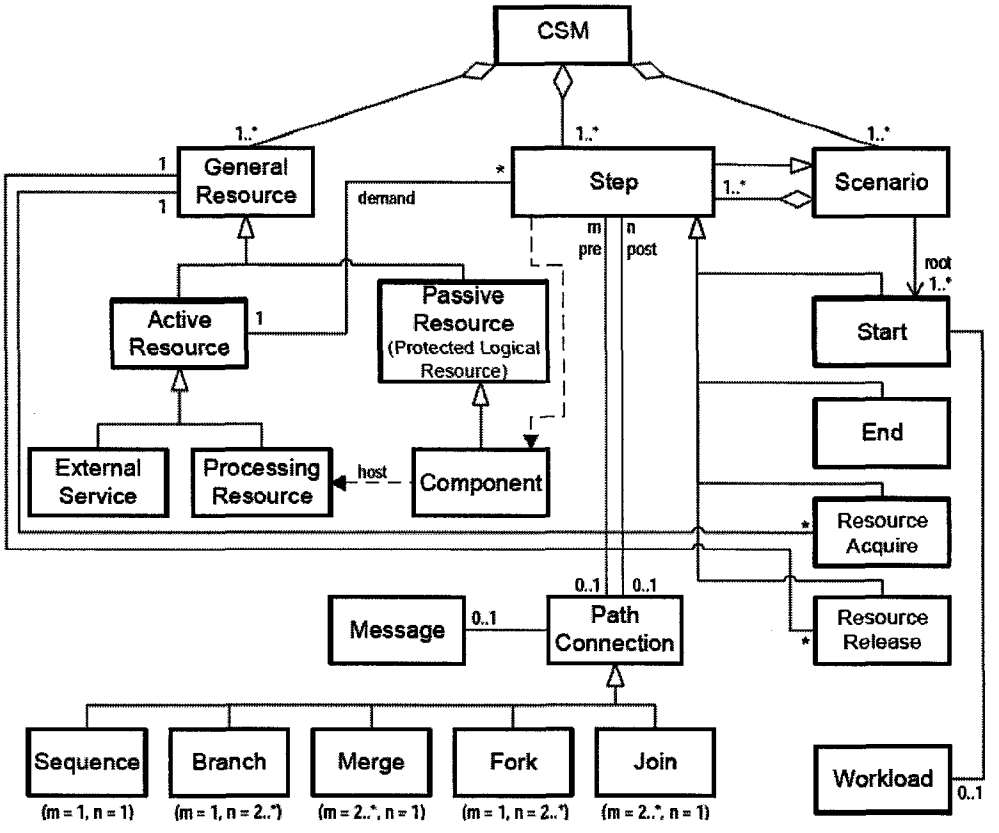



**Figure 3-7 CSM Metamodel [32]**

Active Resources can further be specialized into External Operation and Processing Resource. 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.

# Chapter 4. Transformation

## 4.1 Overview of the Transformation Approach

The transformation is composed of a set of transformation rules (described in the next section) and a transformation algorithm, which invokes these rules to perform 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 sections 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 result.

The proposed transformation takes as input UML models extended with a Performance Profile (either from SPT or from MARTE). Stereotypes extend some of the UML metamodel elements, adding stereotype attributes (or tagged values) to the metamodel atttributes defined in the UML standard [10]. These new attributes are used to add performance-related information to the model.

Not all of the MARTE and SPT Performance Profile stereotypes and attributes are going to be discussed here, as that could be quite extensive. Table 4-1 shows the mapping from UML metaclasses extended with stereotypes to CSM model elements. The first column gives the UML 2.0 metamodel element that is being examined, which should be extended either with a SPT stereotype (shown in the second column) or a MARTE stereotype (shown in the third column). The corresponding CSM element is given in the fourth column.

**Table 4-1** Mapping of Stereotypes to various CSM Elements

| UML 2.0 Elements | SPT profile | MARTE profile | CSM Elements |
|---|---|---|---|
| ***Deployment Diagram*** | | | |
| Node | PAresource | Schedulable Resource | Passive Resource |
| Node | PAhost | GaExecHost | Processing Resource |
| Component | PAresource | Schedulable-Resource | PassiveResource, Component |
| Component | none | None | Component |
| ***Activity Diagram*** | | | |
| Activity | <<PAcontext>> | <<GaAnalysis Context>> | Scenario |
| StructuredActivityNode | <<PAstep>> | <<PaStep>> | Sub-Scenario |
| ActivityNode | <<PAstep>> | <<PaStep>> | Step |
| InitialActivityNode | none | | Start connector , a Resource Acquire step |
| ActivityFinalNode | none | | Resource Release step + End Connector |
| FlowFinalNode | none | | Resource Release step |
| ActivityEdge (From Activity Node) | none | | Sequence, might be along with Resource Acquire step, Resource Release step |
| ActivityEdge (From Control Node) | none | | Sequence, possibly along with Resource Acquire step, Resource Release step |
| ActivityEdge(From Signal Action) | none | | Sequence and fork or join, possibly along with Resource Acquire step |
| ForkNode | none | | Fork |
| JoinNode | none | | Join |
| DecisionNode | none | | Branch |
| MergeNode | none | | Merge |
| ActivityPartition | none | <<PaRun TInstance>> | Processing Resource |

## 4.2 Mapping Rules

### 4.2.1 Finding Resources

In an activity diagram, a partition is a kind of activity group for identifying actions that have some characteristic in common. As described above, we make the following assumptions about partitions:

1. the activity diagram has partitions representing resources allocated to activity nodes,
2. all nodes are within these partitions,
3. there is only one-dimensional partitions, which means that there is no one node within more than one partition.

In fact, this assumption comes rather naturally, as we are used to grouping in a swimlane the activities executed by a given schedulable resource (process or thread).

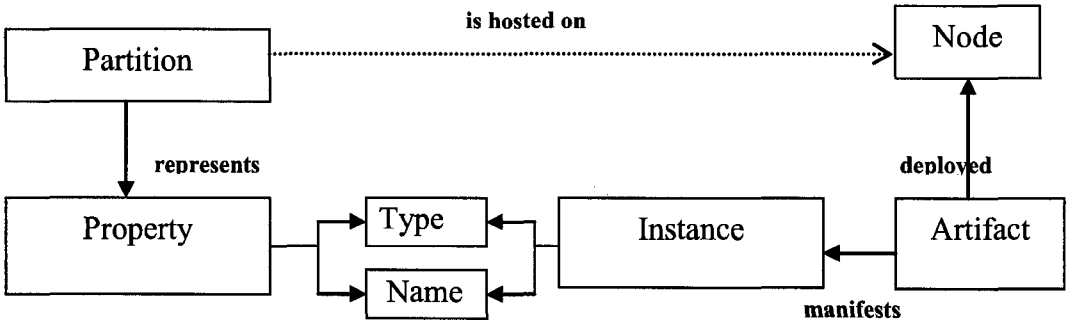



**Figure 4-1 Deployment diagram relationship (for SPT)**

According to the above assumptions, partitions in activity diagrams should represent properties that correspond to components in the CSM metaclass diagram. A property,

that represents a partition, has an element "type" in UML 2.0. (See Figure 4-1). In a deployment diagram, an artifact is deployed on a node. A node represents a processing resource. An artifact manifests an instance specification, which has a type. If this type is the identical to the property type, and also if the property name is the same as the instance specification, then the component corresponding to the partition property is hosted on the processing resource represented by the node.

For UML2 extended with MARTE, the mapping rule is more straightforward, because swimlanes are stereotyped with << PaRunTInstance>>, which has an attribute host of GaExecHost class, indicating the host of the process and thus the host of all Steps associated with this run-time instance. Thus, we can easily find the resource that is stereotyped with the GaExecHost instance. The process is illustrated by the following figure.

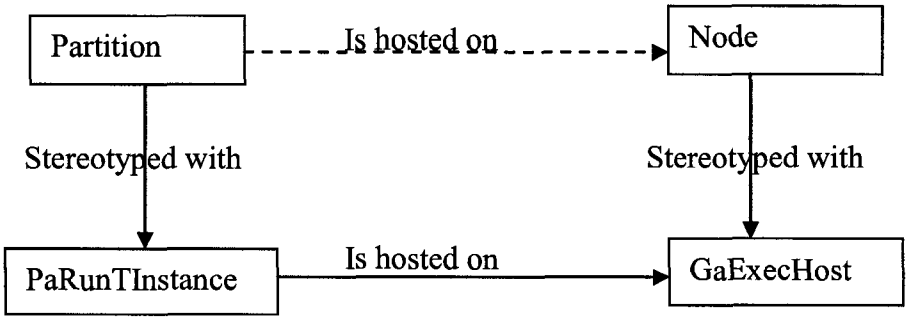



**Figure 4-2 Deployment diagram relationship (for MARTE)**

In Figure 4-3, the browser and the server are both transformed into CSM components. However, only the server gets a processing resource, Server, 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 ServerProgram which is deployed on the Server node.

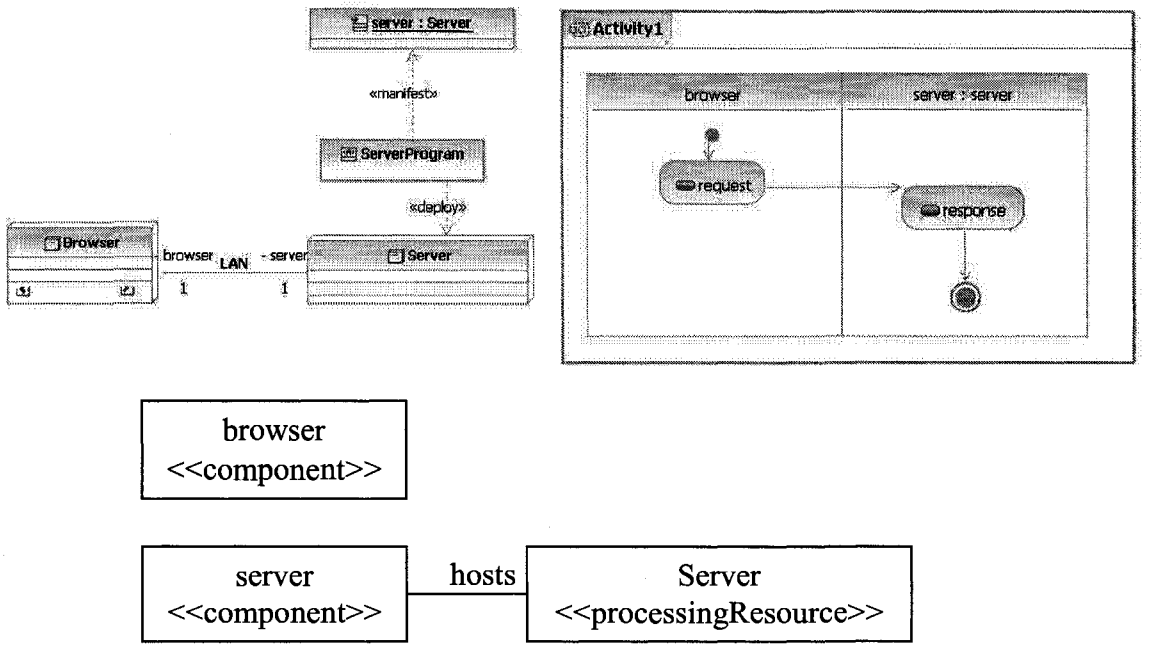



**Figure 4-3 Structural Transformation**

## 4.2.2 Transformation Rules

In this section, we will introduce the transformation rules. First, we will discuss how stereotyped UML2.0 elements are mapped to CSM elements; based on that, another important issue, the order of transformation, will be discussed later.

### Activity and StructuredActivityNode

As stated above, scenarios are translated from two types of UML2.0 elements. The first one is Activity. The translator assumes that for each scenario there is one activity that describes the behaviour in a UML2.0 model, and searches for the activity in the given

model. The other element to be translated into scenario is StructuredActivityNode, which represents a structured portion of the activity that is not shared with any other structured node and is translated into a sub-scenario.

When the translator finds the activity in UML2.0, it starts to translate elements one by one according to a certain order, which will be discussed later. The translation starts with an InitialActivityNode, and does not end until all elements get translated.

Translating StructuredActivityNode is similar. However, StructuredActivityNode in UML2.0 is different from Activity in that the former one does not necessarily have InitialActivityNode and FlowFinalNode, and because of this, we treat StructuredActivityNode as Activity after the former is converted to the latter. The translated scenario from StructuredActivityNode will be arranged as a sub-scenario.

**Table 4-2** Mapping rule for Activity and StructuredActivityNode

| **Meta-model in UML2.0** | **Meta-models in CSM** |
|---|---|
| Activity | {Scenario} |
| StructuredActivityNode | {Scenario} *(as sub-scenario)* |

## ActivityNode

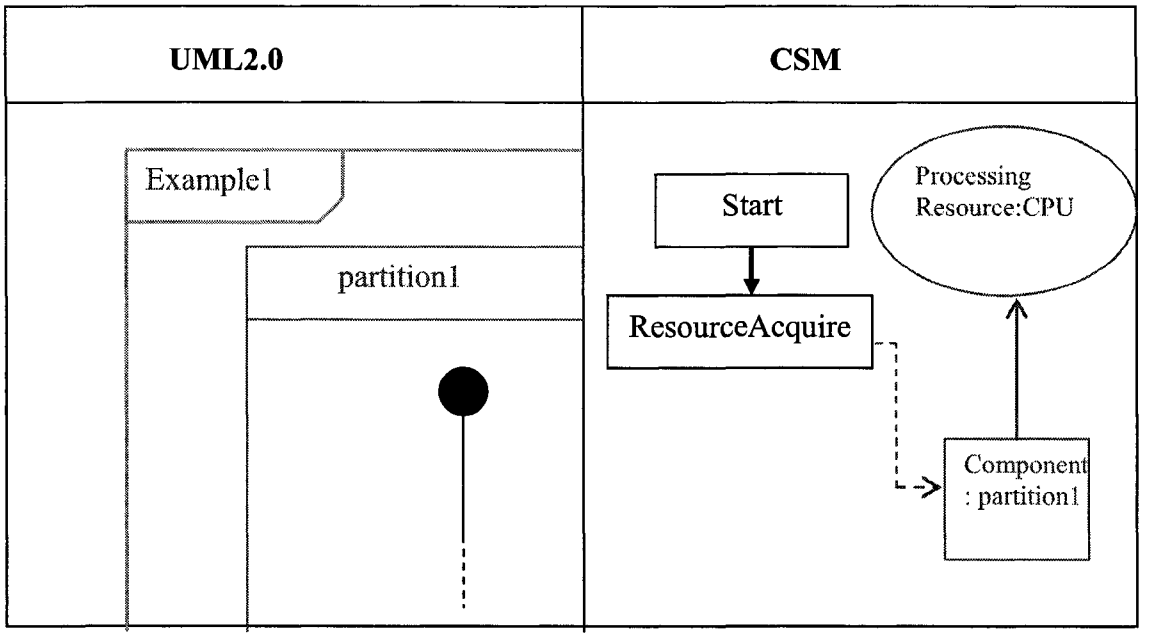
Basically speaking, ActivityNode is translated into Step. However, we still need to pay more attention to some special ActivityNodes.

### InitialNode

InitialNode is important for the whole translation in that:

1. the translation starts with InitialNode, and
2. InitialNode is related to resource acquiring, so that it will be translated into more than one CSM element.

In fact, InitialNode would be translated into a Start and a ResourceAcquire step, which is associated with the Resource translated from the partition in which the InitialNode is. At the same time, if the initial node is stereotyped with <<GaWorkloadEvent>>, a workload will be generated and associated with the translated Start step.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| InitialNode | {Start, ResourceAcquire} |

**Figure 4-4 Transformation of InitialNode**

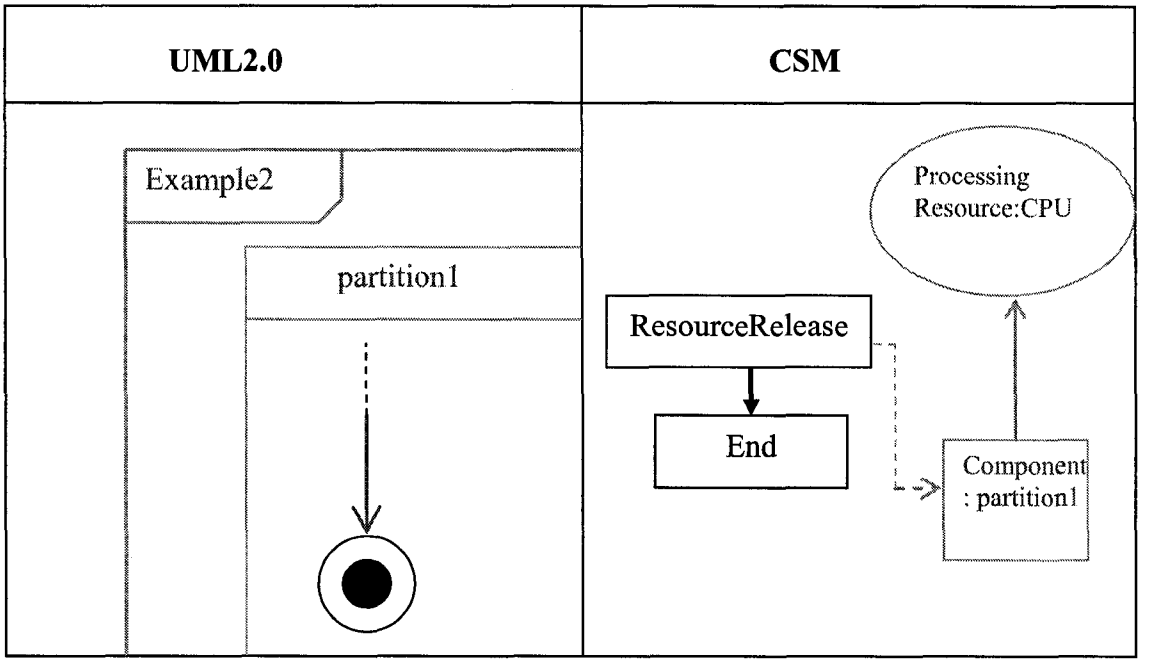
In Figure 4-4, the elements represented in black are mapped from UML to CSM, and the elements in gray represent the context in which the black ones appear.

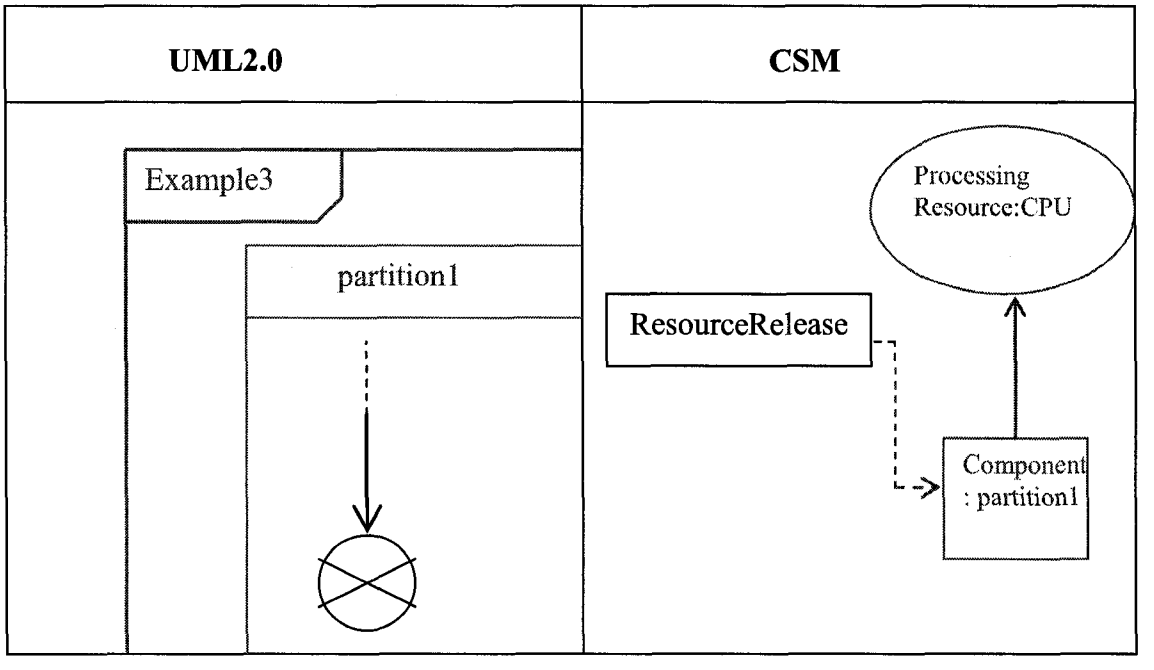
**FinalNode**

There are two types of FinalNode in the UML2.0 metamodel. FlowFinalNode terminates a flow, while AcitvityNode stops all flows in an activity.

FlowFinalNode will be translated into a resource release step, of which the predecessor is the CSM element that is translated from the FlowFinalNode's incoming edge. This resource release will be associated to the resource corresponding to the partition that the FlowFinalNode is in.

Translating ActivityFinalNode is similar. An ActivityFinalNode would be translated into a resource release and an end since it stops the whole scenario. Similarly, the resource release would be associated to the resource corresponding to the partition that the ActivityFinalNode is in.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityFinalNode | {ResourceRelease, End} |
| FlowFinalNode | { ResourceRelease } |

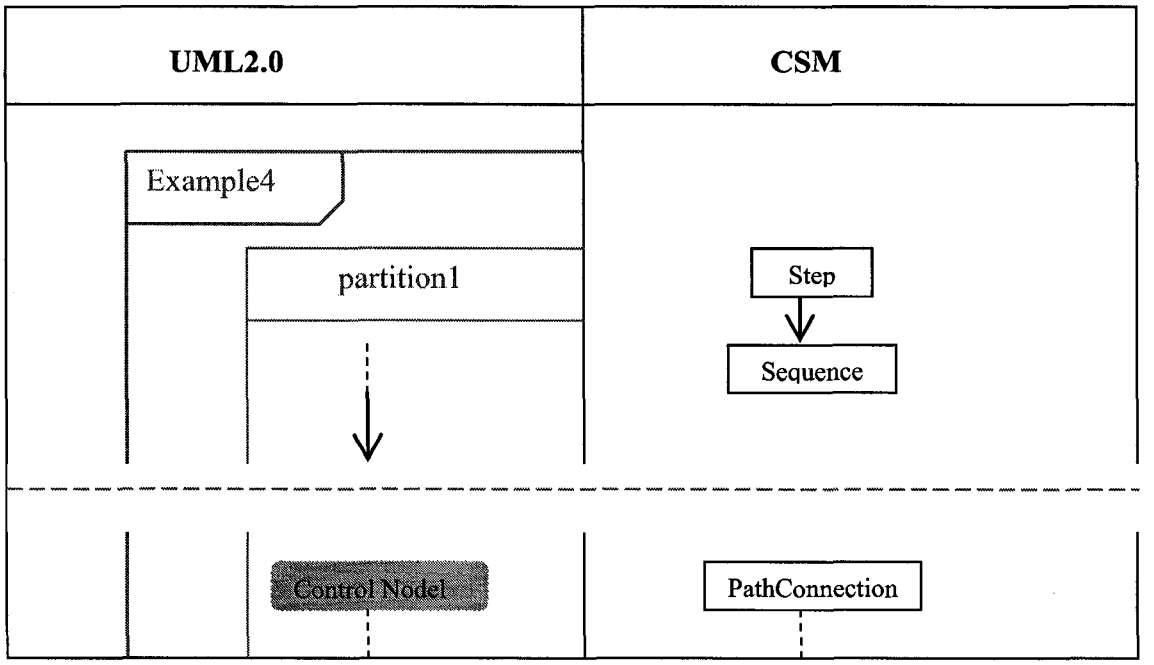
**Figure 4-5 Transformation of ActivityFinalNode and FlowFinalNode**

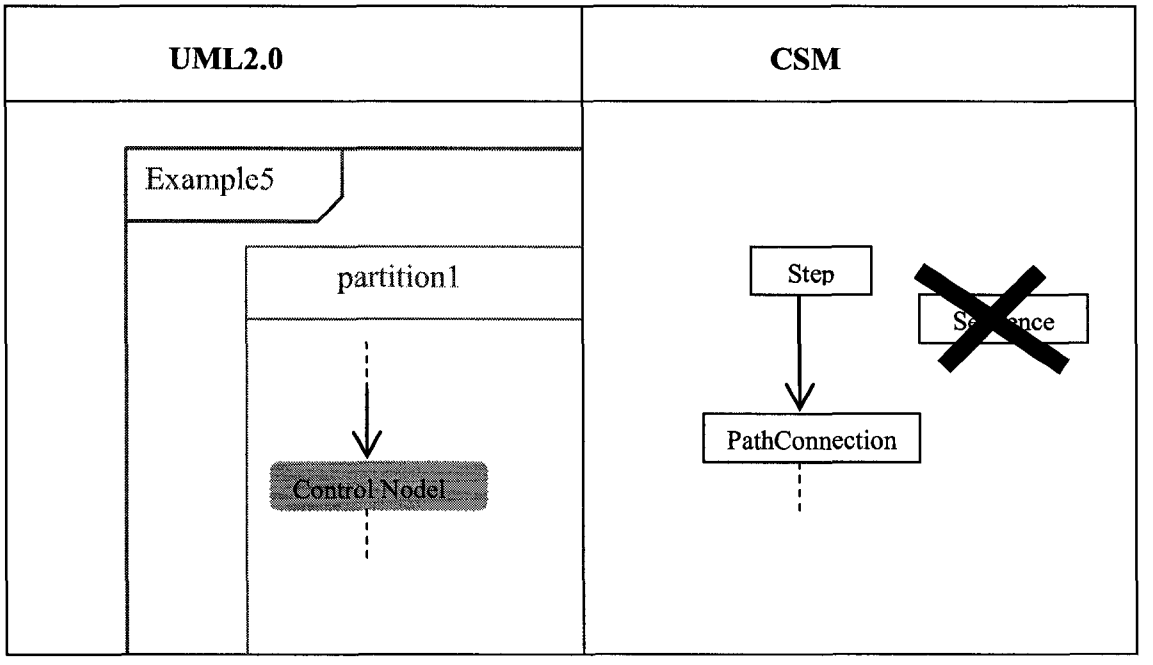
**Control Node**

Generally speaking, control nodes will be translated into corresponding pathConnections, including forks, joins, merges and branches. However, different from UML2.0 metamodel, which links control nodes with edges, pathConnections in CSM cannot be either predecessor or successor of another pathConnection. For this reason, we cannot simply link the two elements that are translated from the control node and its incoming edges, respectively. The solution can be described in the following steps:

1. Find the sequence that is translated from the incoming edge,
2. find the step to which the found sequence is linking to,
3. directly link the pathConnection that is translated from the control node to the step found in step2,
4. delete the sequence found in step1.

The following figure illustrates the steps. The top figure shows that the edge and the node are initially translated into {step, sequence} and {pathConnection} respectively. (The dotted line means that the translations are done separately at first.) The following figure shows the combination and the deletion.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ControlNode | {PathConnection} |

**Figure 4-6 Transformation of ControlNode**

Specifically, fork nodes, decision nodes, merge nodes and join nodes are translated into forks, branches, merges and joins in CSM metamodel, respectively. The latter two have some differences from the former ones since they might have more than one incoming edge, which requires doing the above steps for each edge.

**Converting Structured Nodes**

As discussed above, structured nodes will be treated as an activity, and will be translated into sub-scenarios. The conversion can be described by the following steps:

1. Clone all nodes and edges in the structured node,
2. find the node which does not have an incoming edge,
3. if the node is not an initial node, an artificial one plus an edge would be added before the node found in step 2;
4. find the node which does not have an outgoing edge,
5. if the node is not a final node, an artificial one plus an edge would be added after the node found in step 4.

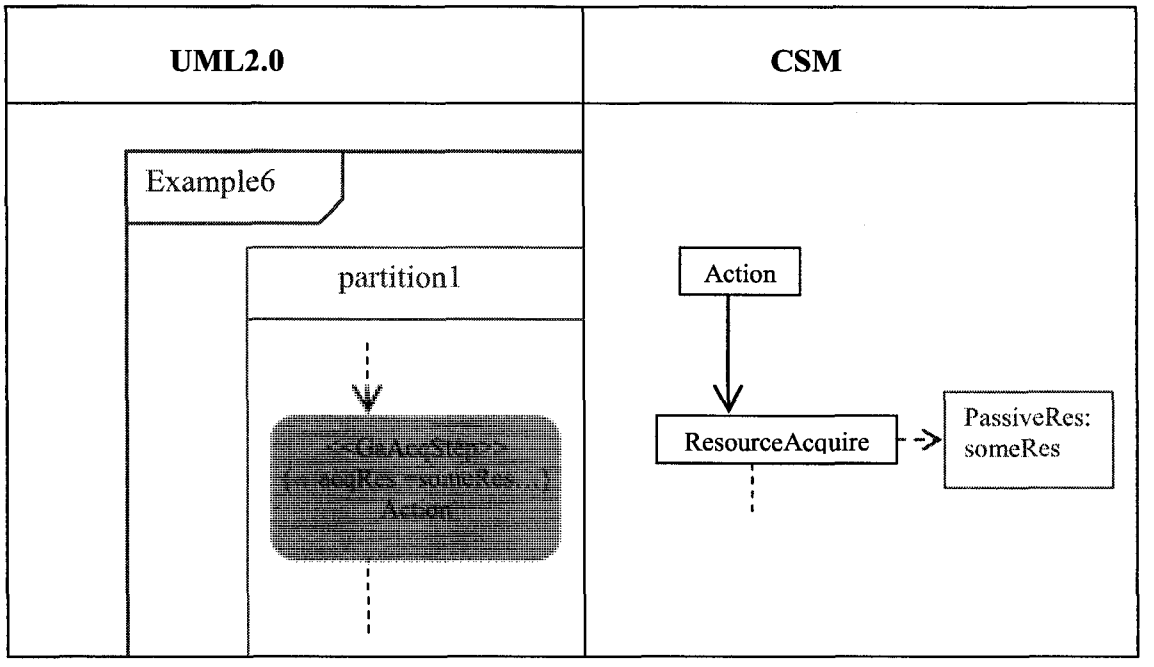
Thus, a complete activity is constructed.

**Nodes stereotyped with GaAcqStep/GaRelStep (MARTE) or GRM Annotations (SPT)**

These kinds of nodes are special in that the stereotypes explicitly show that some resources, which might not be represented by partitions, would be acquired or released,

while the nodes themselves do not imply such operations on resources. Thus, to reflect the scenario correctly, a passive resource should be looked up, and the translated step should include an additional ResourceAcquire/ResourceRelease.

In figure 4-7, only the stereotypes from MARTE are illustrated, the mapping rule would be the same for GRM Annotations in SPT Profile. However, ResourceAcquire/ ResourceRelease is not necessarily translated since the algorithm would check if the resource has been acquired/released already, and if so, no more ResourceAcquire/ ResourceRelease is needed.



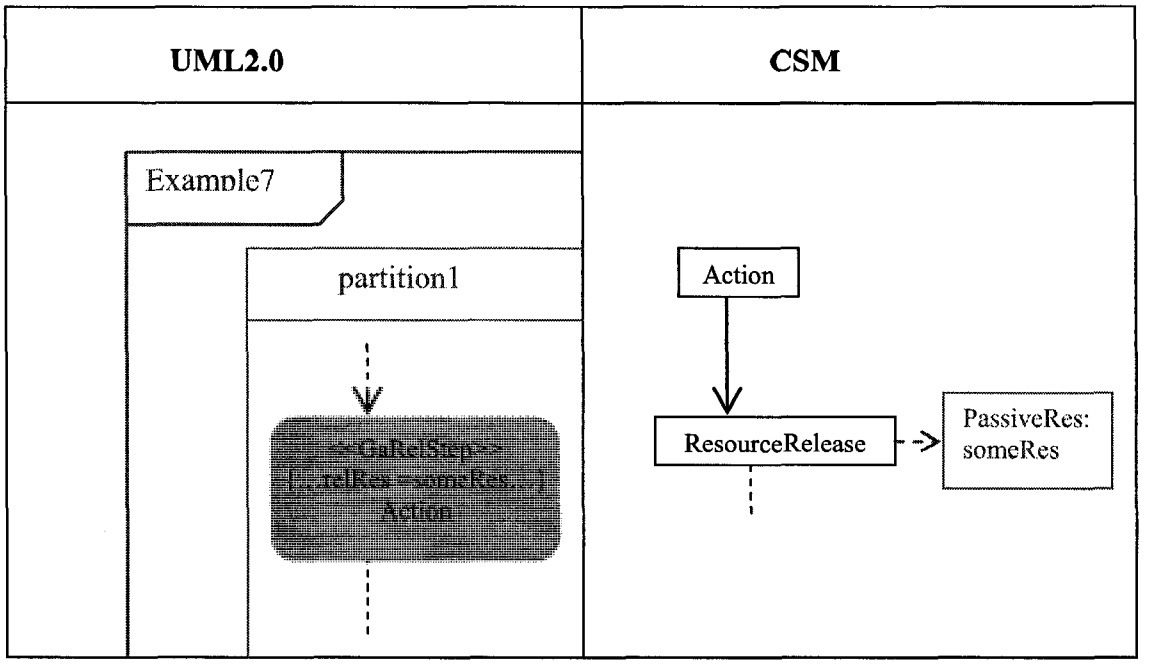



**Figure 4-7 Transformation of action stereotyped by << GaAcqStep >> and <<GaRelStep>>**

The activity nodes other than the ones discussed above will be simply translated into CSM steps.

**ActivityEdge**

Translating edges is more complicated that translating nodes. A single edge could unlikely be translated in one single sequence since we have to consider resource acquire/release. Because of this, we need to pay attention to the context of the edge, including:

1. the type of the edge's source node,

2. whether or not it crosses partitions,

3. whether or not it has siblings. (We define sibling here as follows: an activity edge's siblings are activity edges that have the same source node with the activity edge.)

4. its siblings' context;

Edges whose source is a control node, signal action or accept event action will be discussed later. Other edges, which we call "ordinary" activity edge, will be discussed first.

**Ordinary activity edge**

***Context 1***

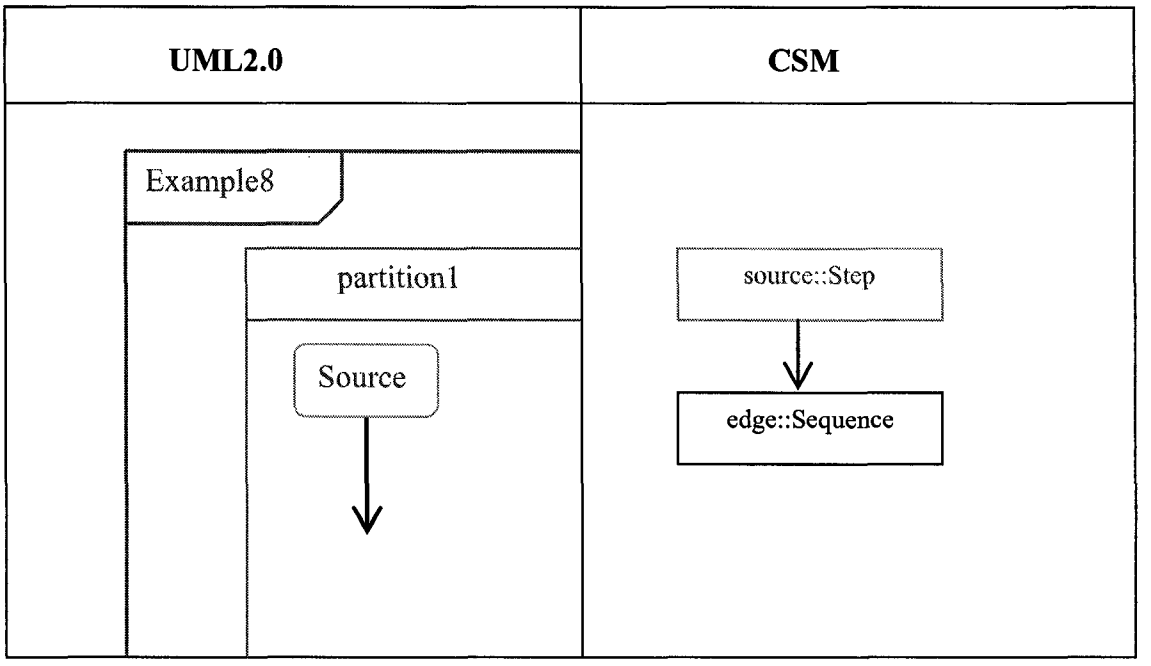
The simplest edge is an edge which:

1. does not cross partitions, or

2. crosses partitions but has a sibling not crossing partitions, and

3. the resource corresponding to the partition that the edge is heading to has been acquired, which means the resource does need to be acquired anymore.

We just need to translate the edge into a sequence, and link the sequence to the step that is translated from the edge's source node.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityEdge (in Context1) | {Sequence} |

**Figure 4-8 Transformation of ActivityEdge(Context1)**

***Context 2***

Another kind of edges have the following characteristics:

1. they cross partitions,

2. they have siblings that do not cross the partitions (for this reason, no resource release step is needed),

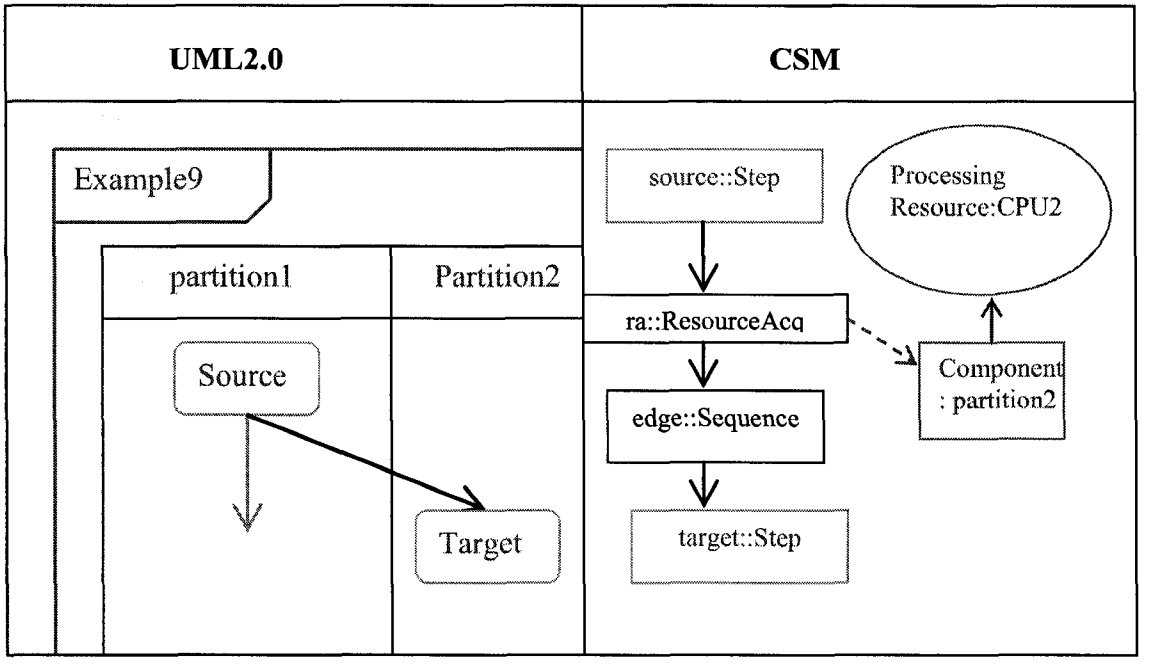
3. the resource corresponding to the partition that the edge is heading to is a "fresh" one that needs to be acquired.

We treat this kind of edges in the following steps:

1. translate the edge into a sequence,

2. create a resource acquire step,

3. since an edge's targets are likely translated into CSM steps, and in CSM, steps cannot be either predecessor or successor of other steps, we need to make sure that the last elements (whose successor is to be added later) among the ones translated from an edge are pathConnection. For this reason, we still need to artificially add a sequence after the resource acquire step that generated in step2.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityEdge (in Context2) | {ResourceAcquire, Sequence} |

**Figure 4-9 Transformation of ActivityEdge(Context2)**

***Context 3***

Some edges have the following characteristics:

1. they cross partitions,

2. they do not have siblings that stays at the source partition (for this reason, resource release step is needed),

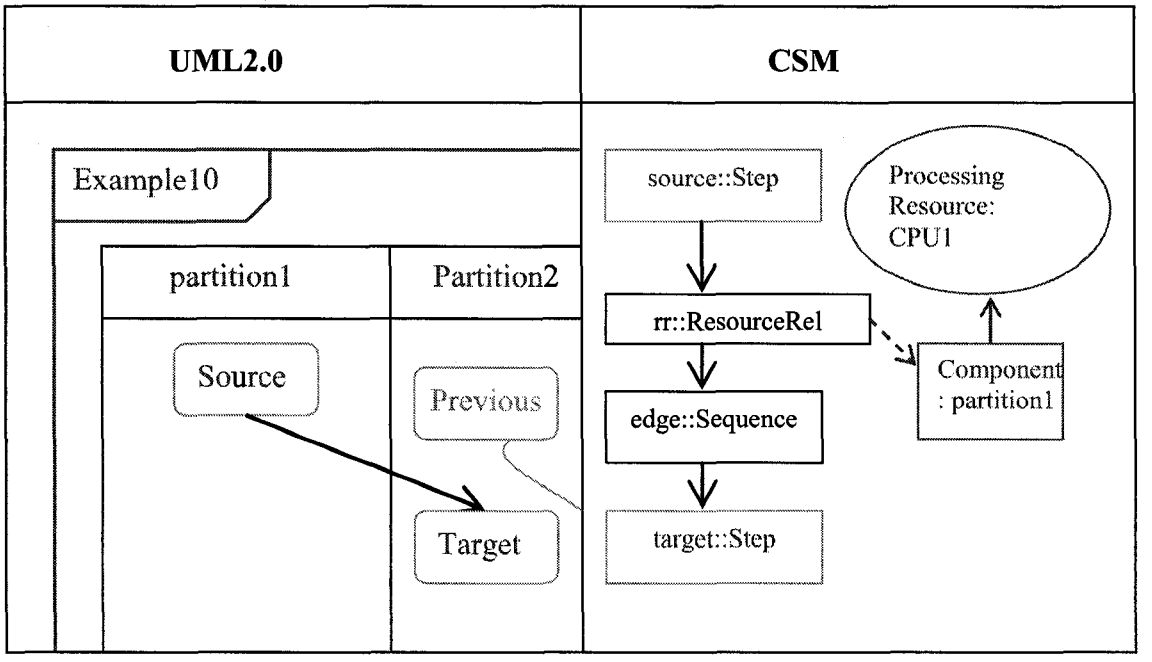
3 the resource corresponding to the partition that the edge is heading to is a "used" one that does not need to be acquired.

We treat this kind of edges in the following steps:

1. translate the edge into a sequence,

2. create a resource release step,

3. for the reason stated above, we still need to artificially add a sequence after the resource acquire step that generated in step2.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityEdge (in Context3) | {ResourceRelease, Sequence} |

**Figure 4-10 Transformation of ActivityEdge(Context3)**

*Context 4*

This kind of edges have more complex characteristics:

1. they cross partitions,

2. they do not have siblings that stays at the source partition (for this reason, resource release step is needed),

3. the resource corresponding to the partition that the edge is heading to is a "fresh" one that needs to be acquired.

We treat this kind of edges in the following steps:

1. translate the edge into a sequence,

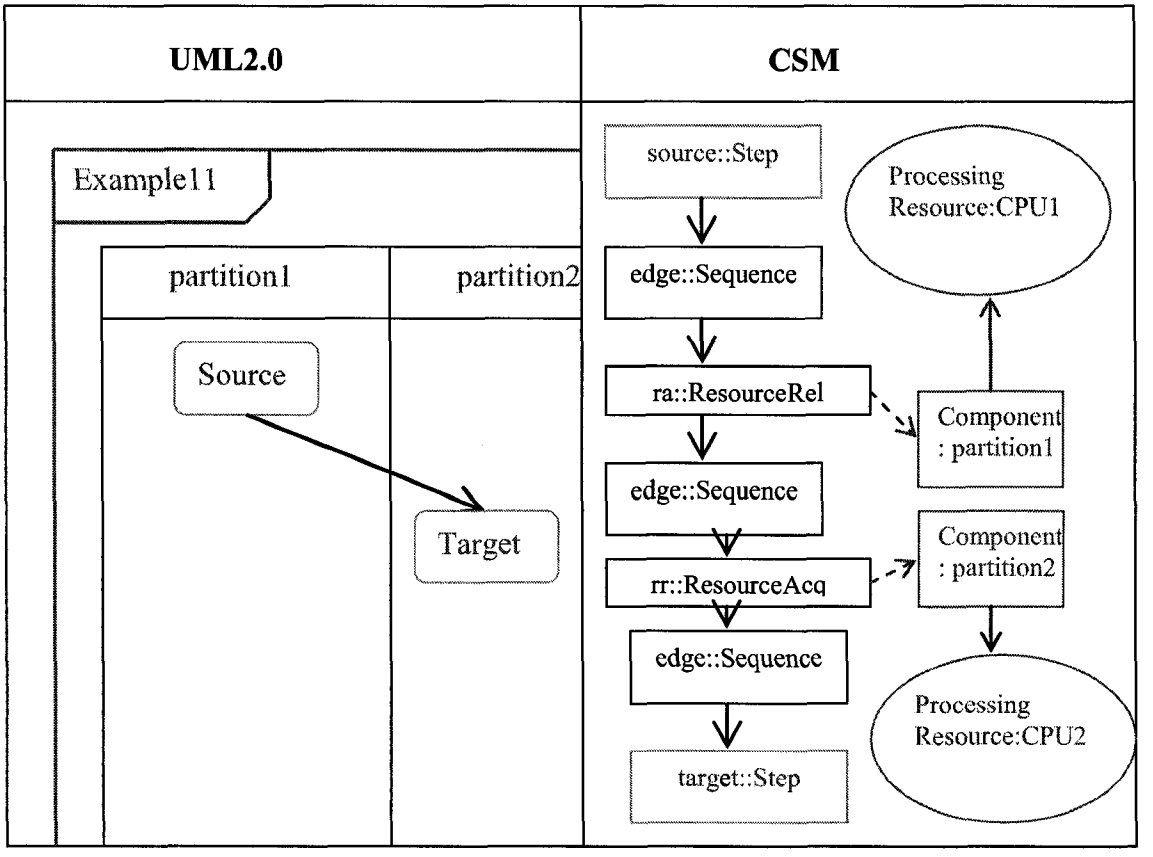
2. create a resource acquire step,

3. create a resource release step,

4. for the similar reason , we need to artificially create a sequence to link the resource acquire step and the resource release step,

3. we still need to artificially add a sequence after the resource release step that generated in step3.

**Edges from control nodes**




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityEdge (in Context4) | {Sequence, ResourceRelease, Sequence, ResourceAcquire, Sequence} |

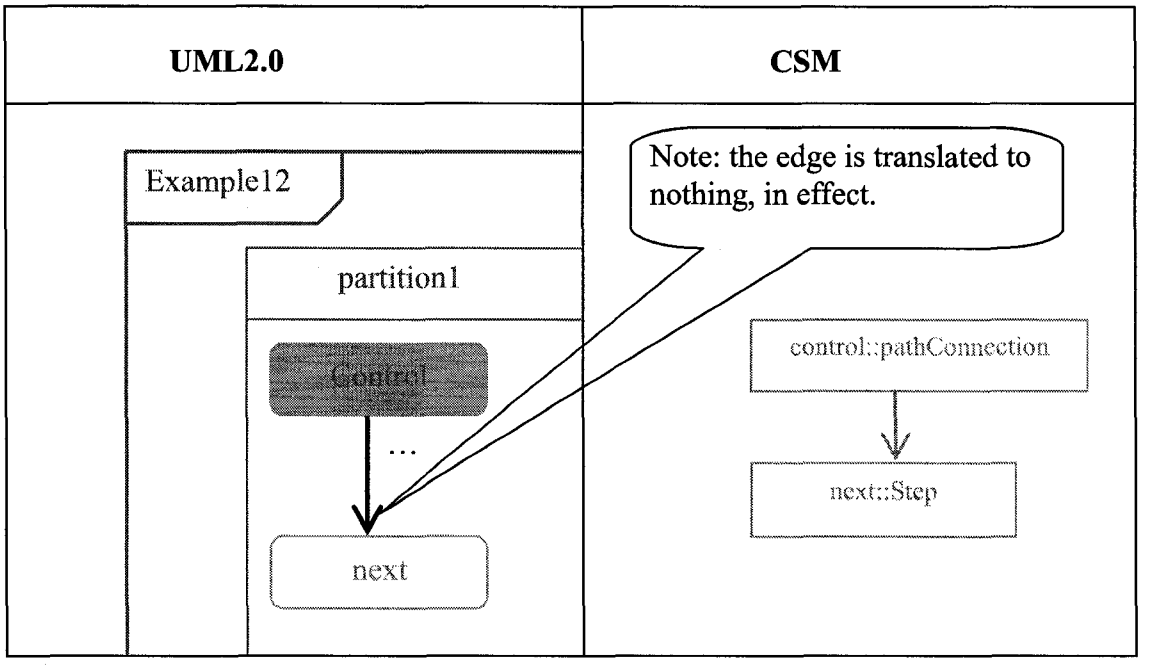
**Figure 4-11 Transformation of ActivityEdge(Context4)**

Translating activity edges, of which the source node is a control node, is similar. However, the CSM elements that UML2.0 control nodes are corresponding to are pathConnections, which could not be directly linked to the pathConnection that translated from edges. For this reason, the translation is different.

Similarly, we discuss the translation from the simplest context. To avoid redundancy, we make use of the four context that have been described above.

*Context 1*

The context for edges from control nodes is even simpler but tricky. Since the source control node has been translated to a pathConnection, what is left to do is just linking this pathConnection to the future node, which actually is the target of the edge. Therefore, we do not need to anything, except to remember that the edge is corresponding to the pathConnection from the control node, the step that the edge's target is translated into should be linked to this pathConnection.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityEdge (from ControlNode, in | {} |

| | |
|---|---|
| Context1) | |

**Figure 4-12 Transformation of ActivityEdge(from ControlNode ,Context1)**

***Context 2***

In this context, we need to acquire the resource that the next partition is corresponding to, but do not need to release any resource. Thus, we do the following steps:

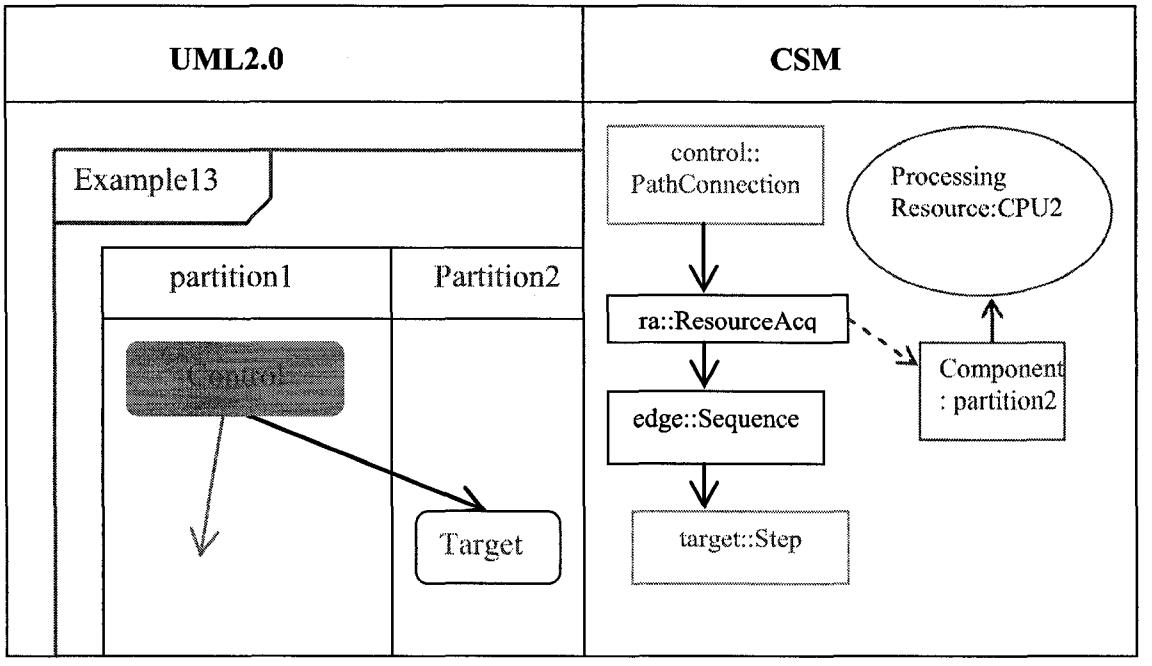
1. create a resource acquire step, and link it to the pathConnection that the control node is corresponding to,
2. artificially create a sequence, which is for link the resource acquire step to the next step that the edge's target would be translated into.

***Context 3***

Context 3 is opposite to Context2, and we need to release the resource that the current partition is corresponding to, but do not need to acquire any resource. Thus, we do the following steps:

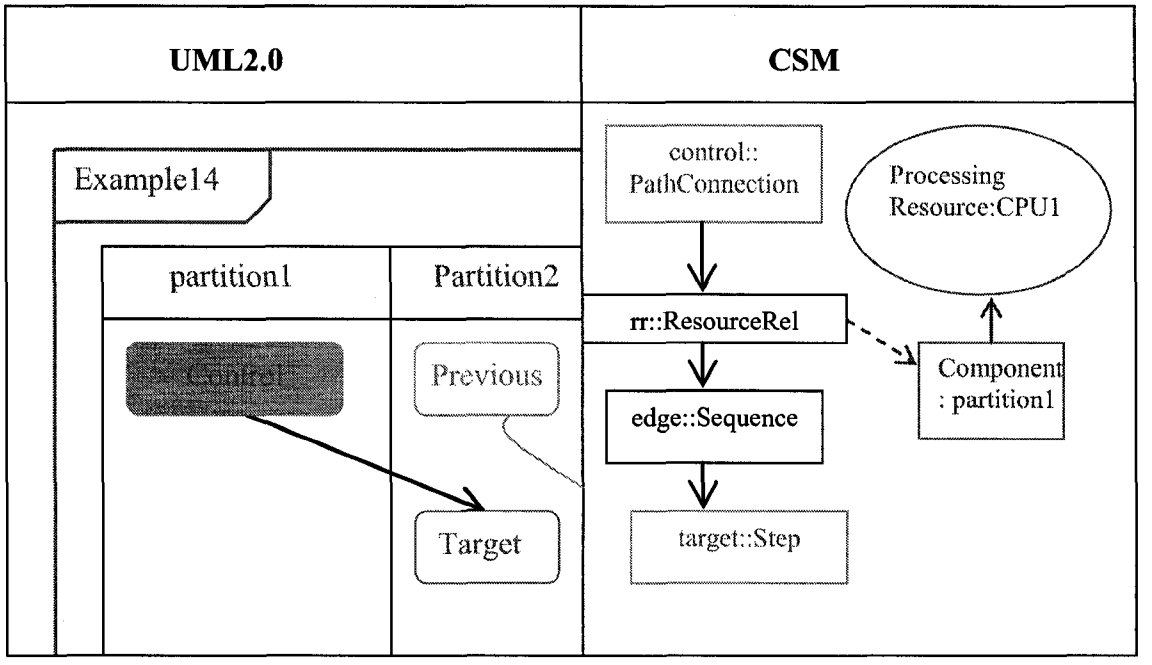
1. create a resource release step, and link it to the pathConnection that the control node is corresponding to,

2. translate the edge into a sequence, which is for link the resource release step to the next step that the edge's target would be translated into.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityEdge (from ControlNode, in Context2) | {ResourceAcquire, Sequence} |

**Figure 4-13 Transformation of ActivityEdge(from ControlNode ,Context2)**




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityEdge (from ControlNode, in Context3) | {ResourceRelease, Sequence} |

**Figure 4-14 Transformation of ActivityEdge(from ControlNode ,Context3)**

***Context 4***

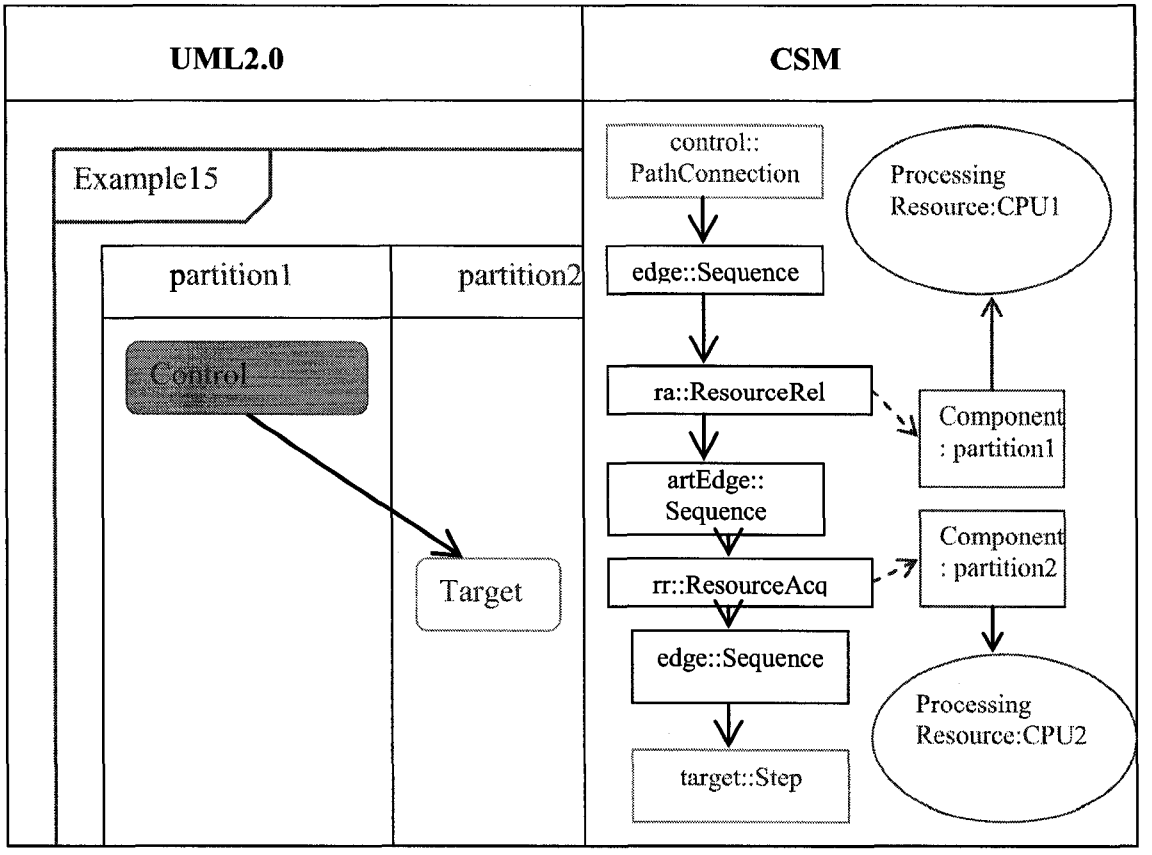
In Context4, we need both to acquire and to release resources corresponding to the next partition and the current partion. So we do the following steps:

1. create a resource release step, and link it to the pathConnection that the control node is corresponding to,

2. translate the edge into a sequence, which is for link the resource release step to the coming resource acquire step.

3. create a resource acquire step, and link it to the sequence that is created at step 3,

4. artificially create a sequence, which is for link the resource acquire step to the next step that the edge's target would be translated into.




| Meta-model in UML2.0 | Meta-models in CSM |
|---|---|
| ActivityEdge (from ControlNode, in Context4) | {Sequence, ResourceRelease,Sequence, ResourceAcquire, Sequence} |

**Figure 4-15 Transformation of ActivityEdge(from ControlNode ,Context4)**

**Edges from Send Signal Actions and Accept Event Actions**

There are 2 kinds of activity nodes that are very useful and common in UML2.0 activity diagram. Differently from other UML elements, it is difficult to find CSM elements that these 2 kinds of nodes could be mapped to.

A send signal action is an action that creates a signal instance from its inputs, and transmits it to the target object, where it may cause the firing of a state machine transition or the execution of an activity, while an accept event action is an action that waits for the occurrence of an event meeting specified conditions.

We study the strategy to translate Send Signal Actions and Accept Event Actions by taking a look at a simple example with these 2 actions.

Let us suppose that we have a snippet of an activity diagram shown in the figure. We assume that the signal that S sends out would be received by and trigger R to fire. We could describe the behavior in this way:

1. after execution of PS, S would send a signal, and then AS would be executed;
2. after execution of PR, R would be waiting the signal;
3. when R receives the signal, it would be triggered, thus AR would be then executed.

If we use CSM elements to describe this model while ignoring resource acquire and release, the CSM would be like the following figure. From the figure, we could conclude that what we need to do is

1. adding a fork after the Send Signal Action,
2. adding a join before the Accept Event Action,

3. translating other elements, and

4. linking the two PathConnections(join and fork). In order to link them, introduction of a NEW step between them is needed.

We still need to consider resource acquire and release since the Send Signal Action and the Accept Event Action could in different partitions. However, to avoid redundancy, we omit the similar translation strategies of the contexts stated above.

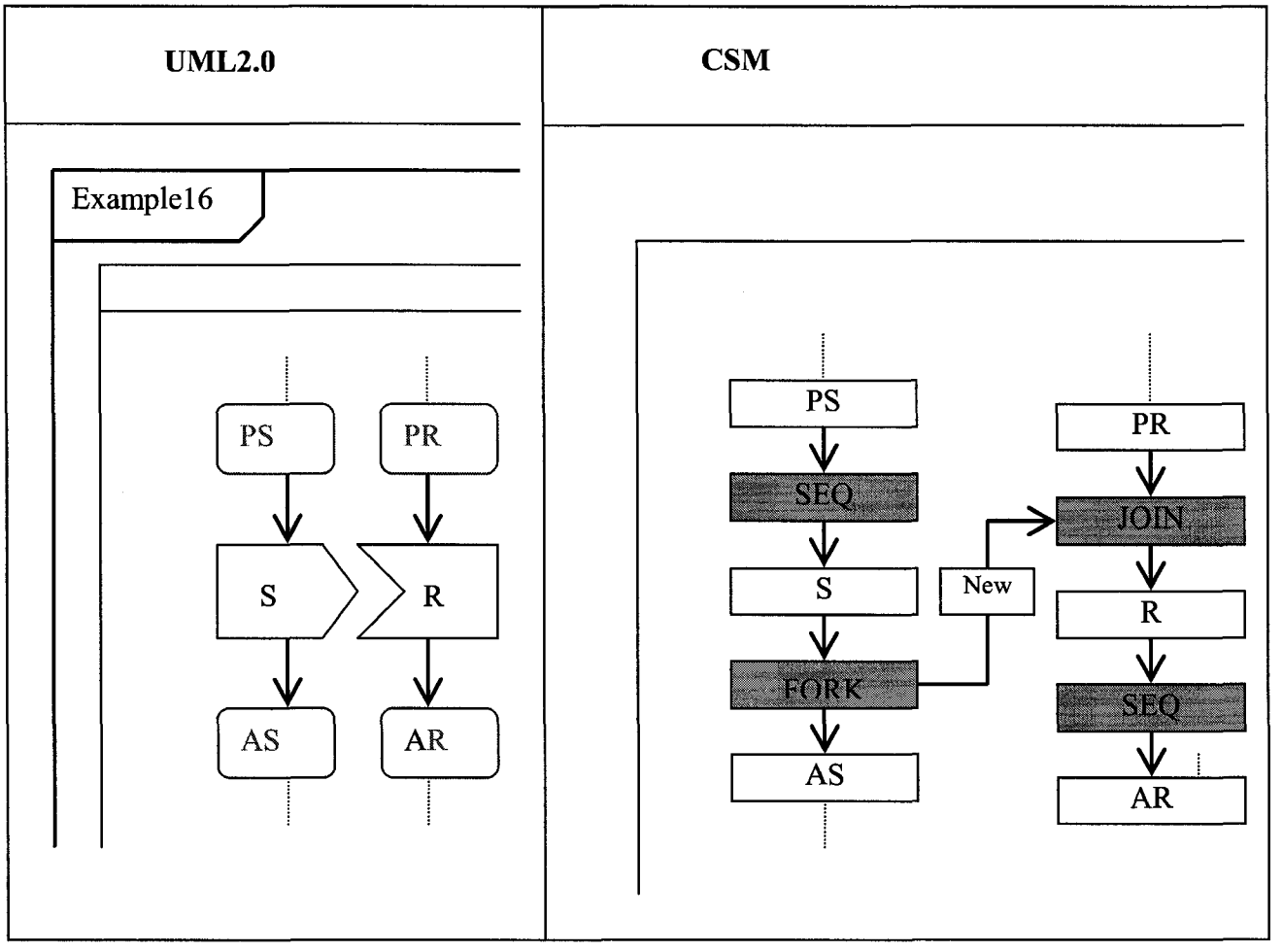



**Figure 4-16 Transformation of Edges from Send Signal Actions and Accept Event Actions**

## 4.3 Traversing Algorithm

Both CSM model and UML2.0 Activity Diagram are about sequence of elements, which means that the order of their elements is important. This section presents a traversing algorithm used to determine the sequencing of elements in activity diagrams. The algorithm works only for acyclic activity diagrams, which is a limitation that could be eliminated, as discussed at the end of this section.

According to the translation rules stated above, we can see that:

1. The translation starts from the initial activity node;
2. Some branches start from nodes that do not have incoming edges. We called those nodes source nodes. (*A branch is defined as a list of nodes and edges that are linked to each other, and a branch starts at a node that has no incoming edge, and ends at nodes without outgoing edges.)*
3. The translation traverses all UML2.0 activity nodes and edges in an activity, and translates them one by one, following a certain translation sequence;
4. One UML2.0 model will be translated into a set of CSM elements, the number of which could be one or more;

We also could describe the translation process of a single UML2.0 element in this way:

1. The translation will start when the previous UML2.0 element has been translated;
2. Rules stated above will be applied to generate a set of CSM element mapping to the UML2.0 element,

3. The translation needs to find and link the right CSM element, among the ones translated from the source/incoming UML2.0 element, to the first CSM element mapping to the current UML2.0 node or edge.

4. The translation needs to remember the translation map, which indicates the translated sets of CSM element for all UML2.0 element.

The process entails that one UML2.0 element MUST be translated AFTER its source/incoming one. That means that the translation sequence mentioned above should guarantee that:

1. A node is listed behind its incoming edge in the sequence, and

2. An edge is listed behind its source node in the sequence.

Note that for activity nodes and edges, there exist the following rules:

1. An edge has and only has one target node, and

2. The number of the outgoing edges of a node could be zero to any number.

The following figure illustrates the relationship between edges and nodes, where circles represent nodes ands lines represent edges. Note that for a scenario there is always only one initial node starting the sequence, however, there might be source nodes that cause branches. The B node in the figure is a source node.

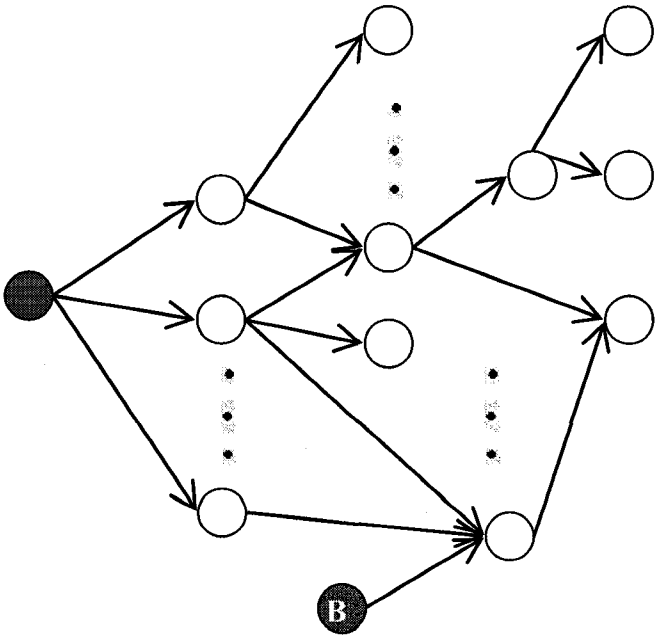



**Figure 4-17 Relationship between edges and nodes**

To guarantee the translation sequence, every element will be numbered in the following way:

1. All source nodes including initial node would be found, and the translation would be started with the first source node, initial node.
2. The source/initial node would be numbers firstly,
3. All outgoing edges of the node(s) of the previous step are then numbered ;
4. All target nodes of the edges of the edges of the previous step are then numbered;
5. Repeat step2 and 3;
6. A node or an edge that has been numbered will be re-written.
7. when a branch is completely translated, step 1 to 6 would be repeated for an iteration of another branch.

We use pseudo code to represent the numbering algorithm:

```
number=0;
theNode=initialActivityNode;
setNumber(theNode, number);
number++;
theOutgoingEdges=getOutgoingEdges(theNode);
numberingEdges(theOutgoingEdges, number);

function numberingEdges(edges, n)
        while(edges)
                setNumber(next(edges), n);
        n++;
        whileEnd
        theTargetNodes=getTargetNodes(edges);
        numberingNodes(theTargetNodes, n);

function numberingNodes(nodes, n)
        while(nodes)
                setNumber(next(nodes), n);
        n++;
        whileEnd
        theOutgoingEdges=getOutgoingEdges(edges);
        numberingEdges(theOutgoingEdges, n);
```

The traversing will take place according to the numbering.

The following example shows how the algorithm works. Suppose we have edges and nodes shown in the following figure. Upper-cased letters name nodes and lower-cased name edges.

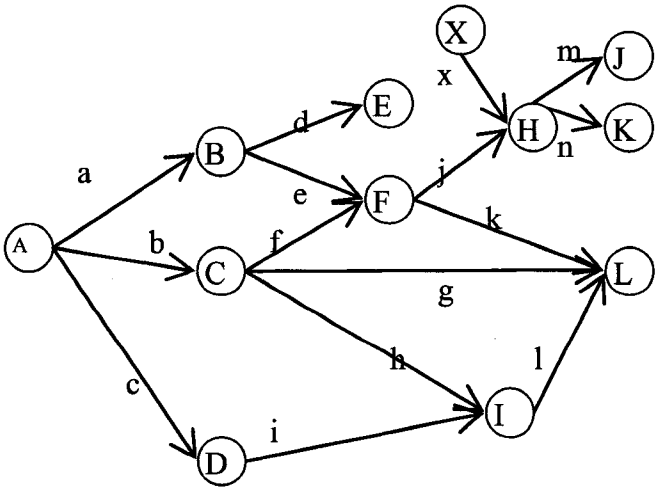



**Figure 4-18 Example of Algorithm (Before)**

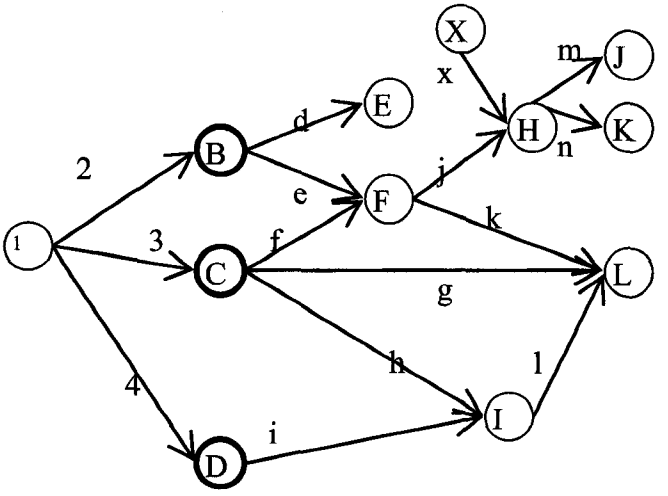



**Figure 4-19 Example of Algorithm (Step1)**

First, we number the initial node, A, with 1. Then we get all the outgoing edges, which

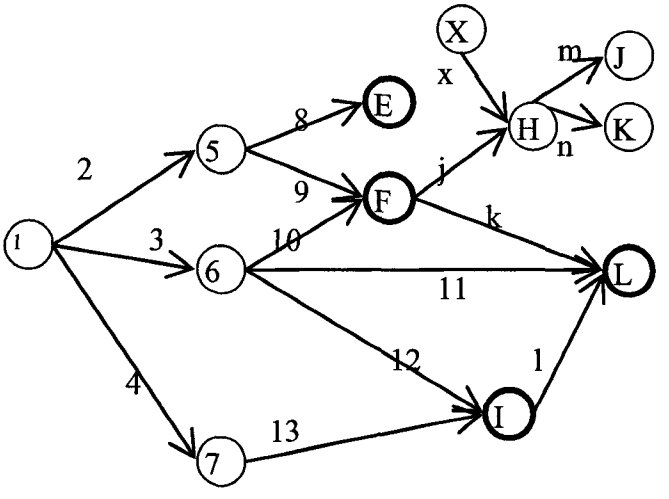



**Figure 4-20 Example of Algorithm (Step2)**

are a, b and c. thus, the edges are numbered as 2, 3 and 4, respectively.

Based on 2, 3 and 4, we need to obtain the list of target nodes. A list of target nodes of a list of edges is obtained according to the edges' sequence. In this example, we obtain target nodes of 2, 3 and 4, and then put them is a single list with their original sequence. So, the list of target nodes would be {B,C,D}, which are then numbered as 5,6 and 7. And we get the list of outgoing edges: {d,e,f,g,h,i}, which are numbered as 8,9,10,11,12,13.

Things grow more complicated next. Since 9 and 10, 12 and 13 have same target nodes, F and I. So the list of targets we obtain is {E, F, L, I}, and they would be numbered as 14,15,16 and 17.

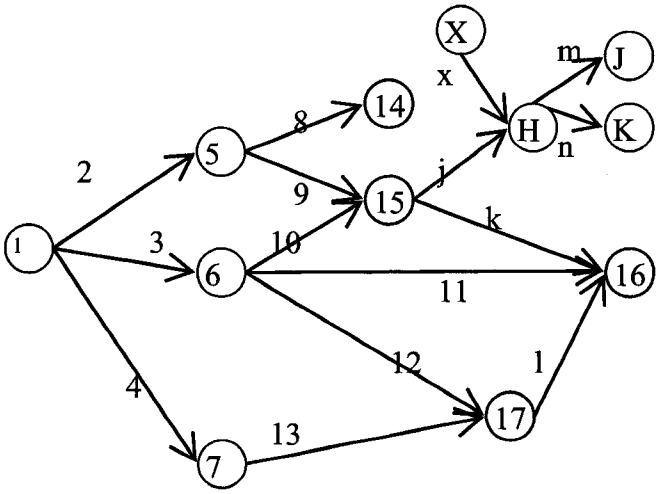



**Figure 4-21 Example of Algorithm (Step3)**

The next outgoing edges would be {j,k,l}, and the target nodes would be {H,16}. Note that 2 nodes among the target nodes have been numbered. {j,k,l}get numbered as

{18,19,20}, while {H,16} would be numbered as {21,22}, where Node16 is re-numbered as Node22.

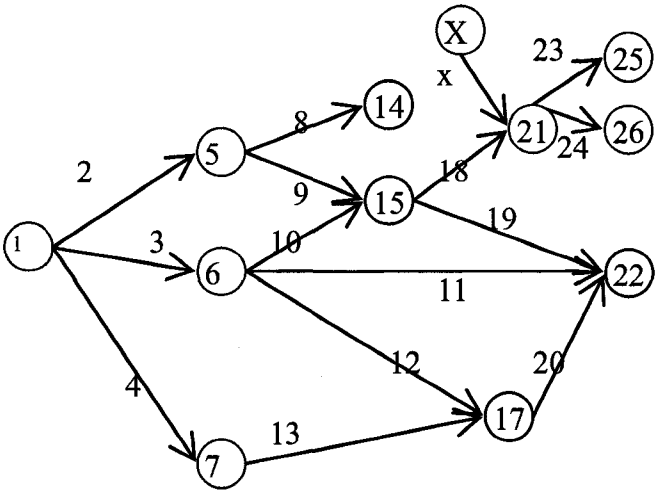



**Figure 4-22 Example of Algorithm (Step4)**

This branch is finished numbering when {m,n} and {J,K} get numbered as {23,24} and {25,26}. At this point, the algorithm would detect another branch, which is started at NodeX since there is no incoming edge linked to this node.

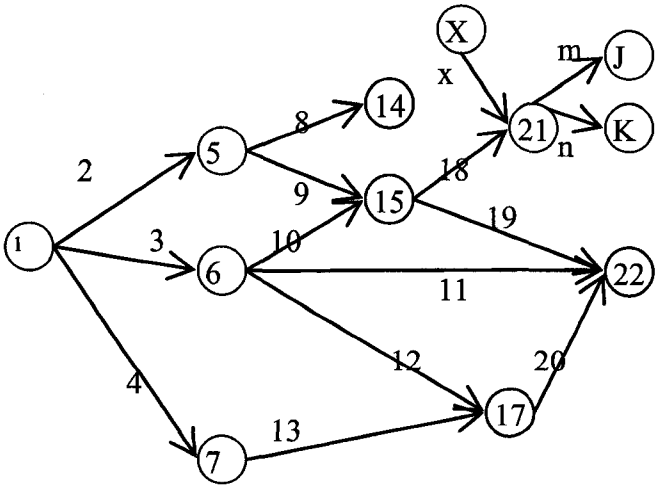



**Figure 4-23 Example of Algorithm (Step5)**

This branch would be numbered from 27. All elements that have been numbered would be re-numbered according to the algorithm.

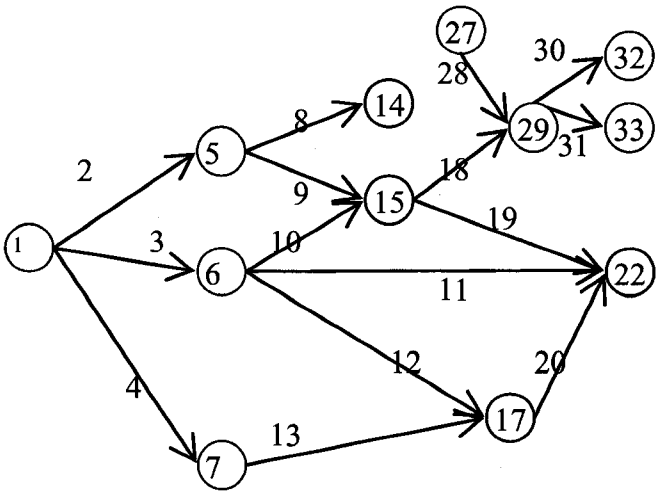



**Figure 4-24 Example of Algorithm (Step6)**

Thus, as we see, all nodes' incoming edges have smaller numbers than the nodes, and all edges' target nodes have smaller numbers than the edges.

As mentioned at the beginning of this section, this algorithm only works for acyclic activity diagrams. For graphs containing cycles, the algorithm would run in an infinite loop. Future work needs to be done to deal with cyclic graphs. A very brief conceptual approach is described here.

First, all subsets of graph elements that form a cycle (called below cyclic subset) need to be located, and outgoing and incoming edges for these subsets need to be found. The subsets would be artificially considered as a sub-activity, therefore treated as one single node to be numbered at the higher level (see Fig. 4-25).

Secondly, a solution to handling basic cyclic graphs needs to be figured out. The approach is that when we meet a cycle, we need to identify the "beginning" and the "ending" of the cycle. The "beginning" element will be treated as if it did not have any incoming/source elements. After all elements in the cycle have been traversed, an additional step of the algorithm would work to link the CSM elements translated from the "ending" with the ones from the "beginning".

Following such an approach, a graph would be numbered in multiple layers. A possible example is shown in Figure4-25, where a cyclic subset is located in the dotted circle, and thus the dotted circle is numbered 17 in the higher layer, as a single node. In the subset, Node17.1 is recognized as the "beginning", and Edge17.6 as "ending", which means Node17.1 is translated as if it did not have incoming edges, and its corresponding CSM elements would be linked after Edge17.6 is translated.

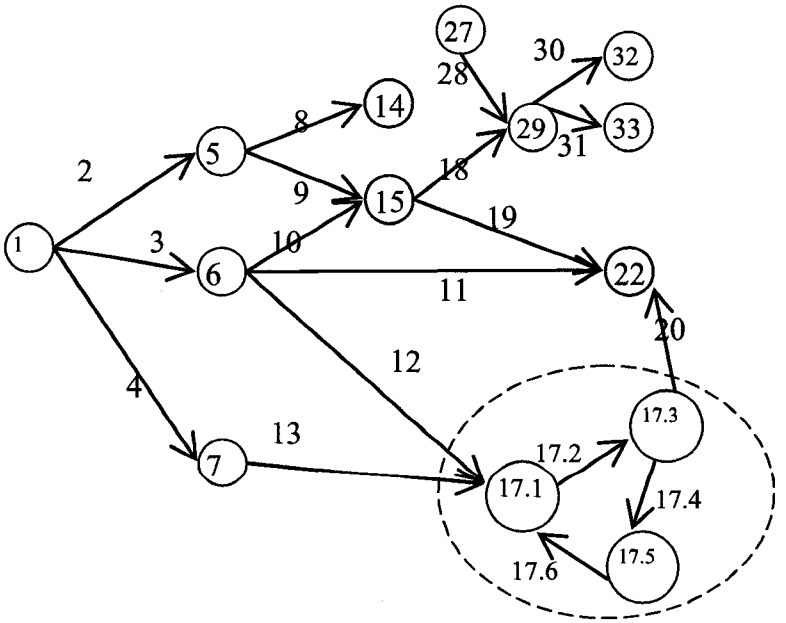



**Figure 4-25 Example of Extended Algorithm**

Again, the approach described here is only a conceptual idea; future work is still needed to figure out the algorithm for finding subsets and defining "beginning" and "ending" of a cycle.

# Chapter 5 Implementation and Verification

## 5.1 Implementation of Transformation

The transformation components can be described by the following class diagram:

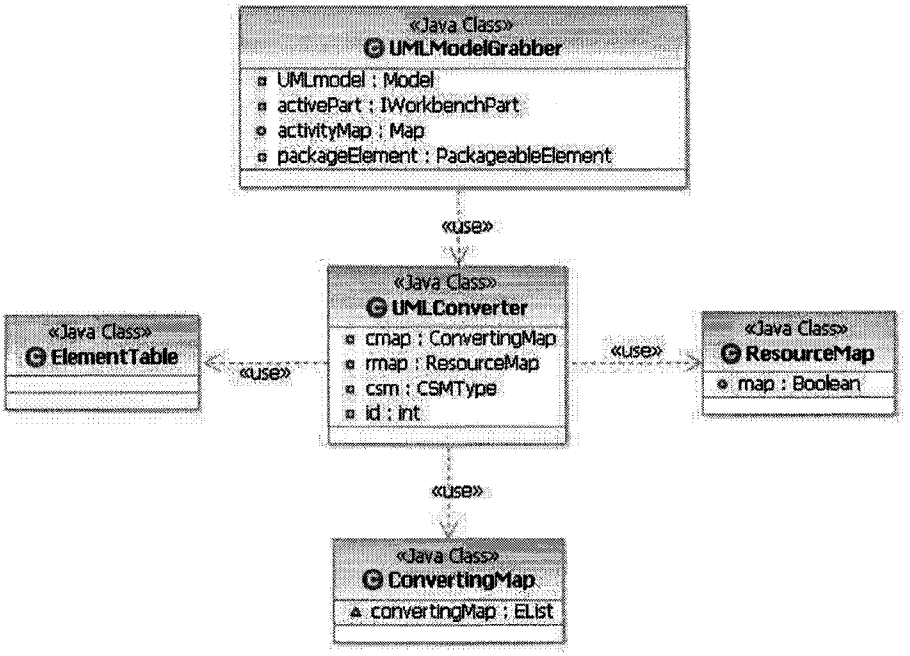



**Figure 5-1 Java Component of transformation**

**UML Model Grabber:**

UML Model Grabber detects UML2.0 model and activity diagrams. Using Eclipse Plug-in Development technology, it extracts the activity diagram from UML models in RSA and passes it to UML Converter.

**UML Converter:**

As the key component, UML Converter takes the activity diagram that Model Grabber passes, and converts it into CSM model. It uses an ElementTable to memorize and number elements in the UML activity diagram, a ConvertingMap to store the thranslation mapping between each UML element and CSM elements translated from the UML element, and a ResourceMap that records resources' acquire/release status. The converter does transformation according to ElementTable's numbering, and it refers to ResourceMap to determine the generation of ResourceAcquire and ResourceRelease for CSM, and ConertingMap to look up elements that have been translated already.

**Element Table:**

Element Table takes activity diagram, and numbers elements in it. Translation would be according to the table's numbering. The previous chapter has described the algorithm.

**Converting Map**

Converting Map is in charge of storing the information that each element is mapping to a list of CSM elements that it is translated into. Before a UML element being translated into a list of CSM ones, UML Converter uses Converting Map to look up for the previous element, and link the translated elements to it.

**Resource Map**

Resource Map is in charge of storing the information of resources and their acquire status. UML converter decides whether a resource should be acquired/released based on this map.

## 5.2Verification

The main purpose of this thesis was to develop a transformation algorithm from UML 2.0 annotated Activity 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 as complex ones. Test Cases (TC) were developed to test the transformation rules and verify the XML outputs. Not only are the expected output model elements present but also the XML output is also verified against Eclipse for the validity of the XML output being generated.

### 5.2.1 Features

For the transformation discussed in the previous chapter, we listed the following features of the transformation:

Feature1. Transformation of InitialNode;

Feature2. Transformation of FlowFinalNode;

Feature3. Transformation of ActivityFinalNode;

Feature4. Transformation of Join, Fork, Merge and Decision;

Feature5. Transformation of StructuredNode

Feature6. Transformation of ActivityEdge in Context1;

Feature7. Transformation of ActivityEdge in Context2;

Feature8. Transformation of ActivityEdge in Context3;

Feature9. Transformation of ActivityEdge in Context4;

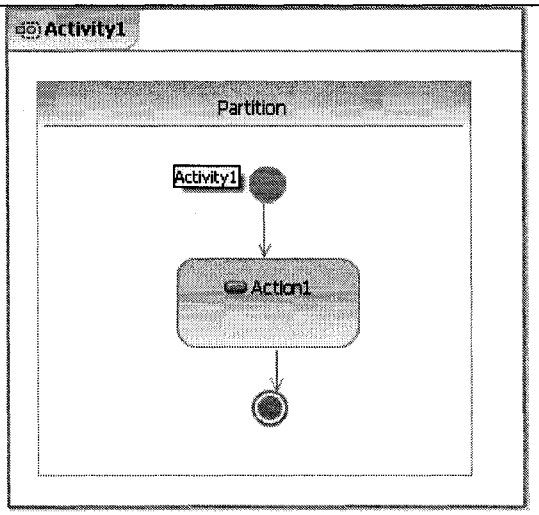
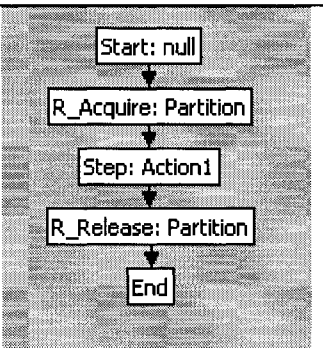
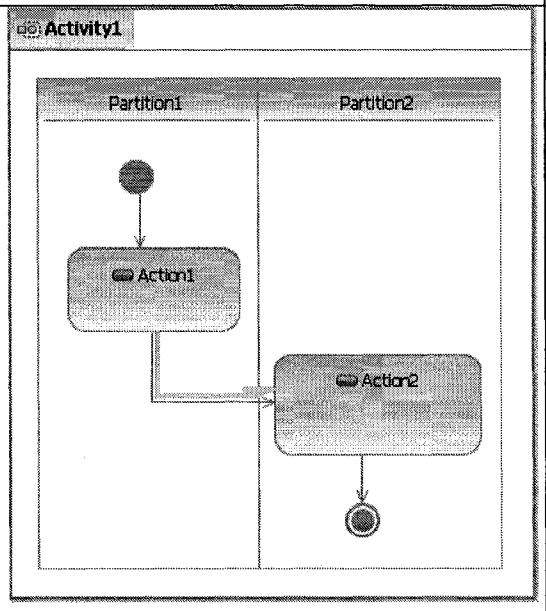
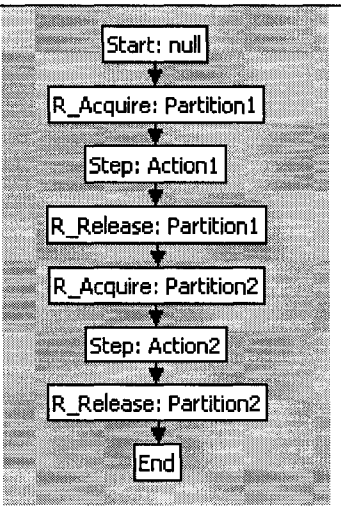
Feature10. Transformation of SendSignalAction and AcceptEventAction;

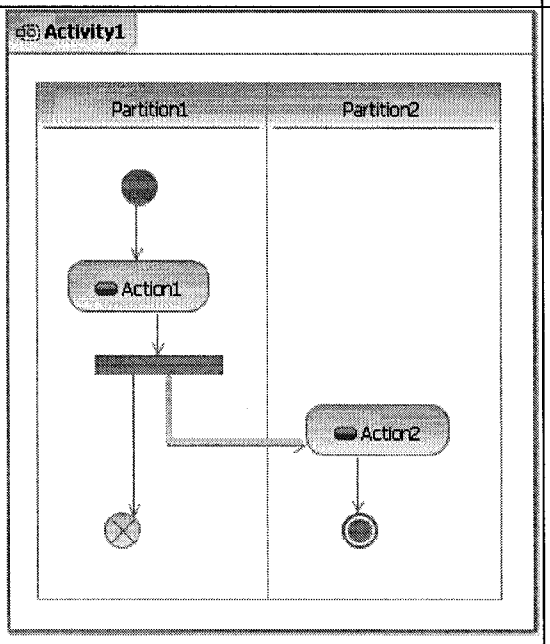
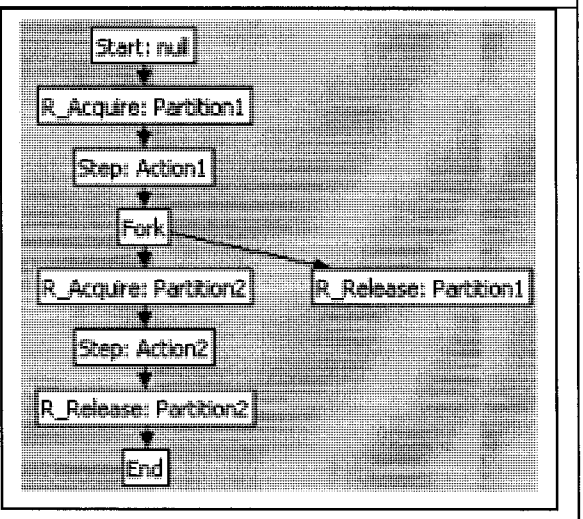
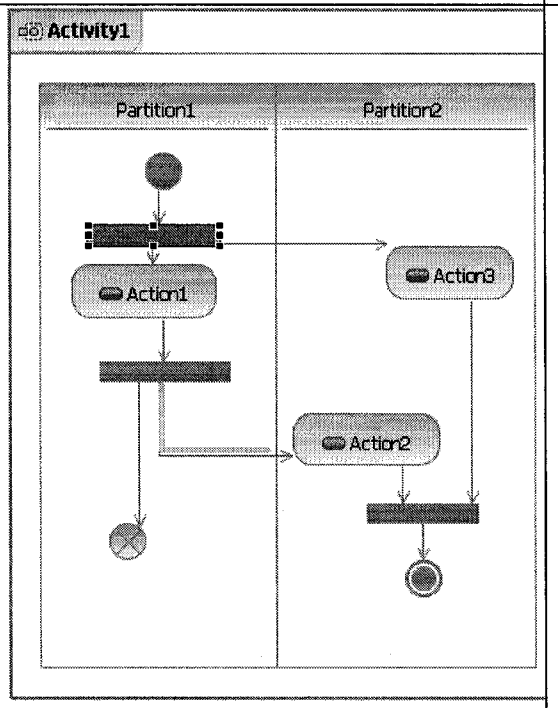
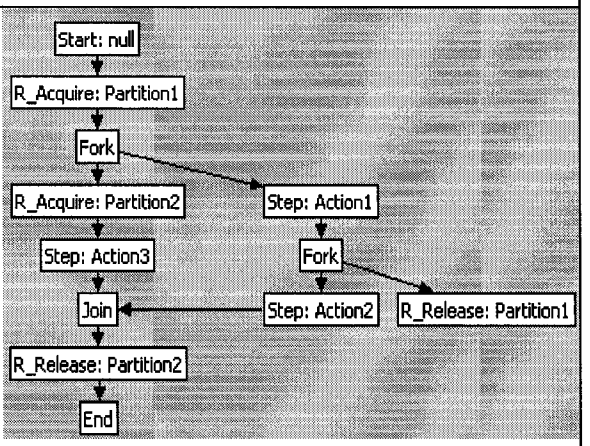
Feature11. Transformation of nodes with GaAcqStep/GaRelStep

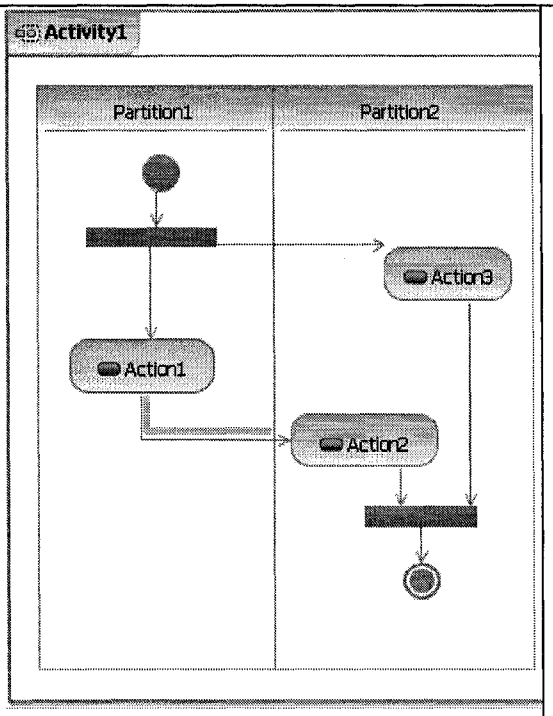
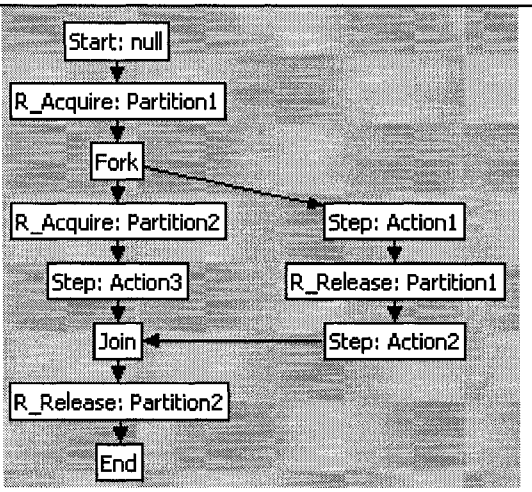
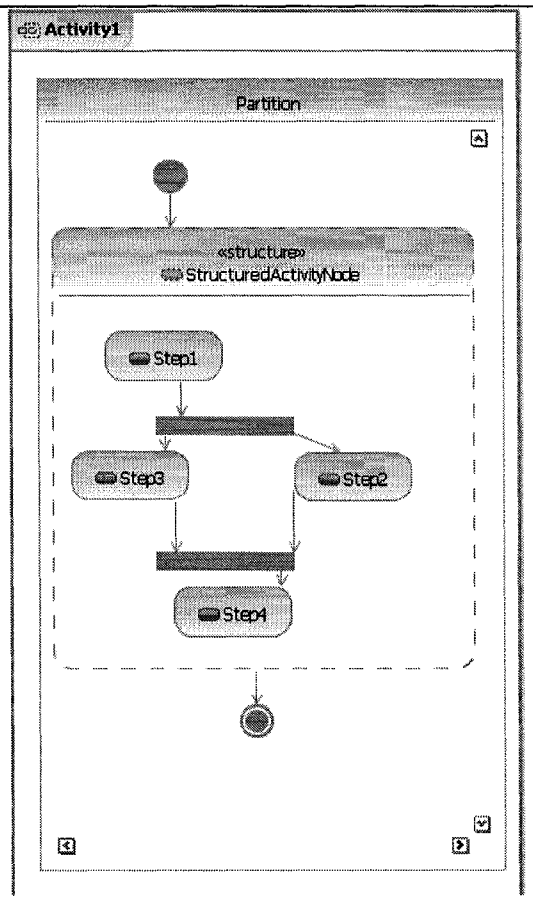
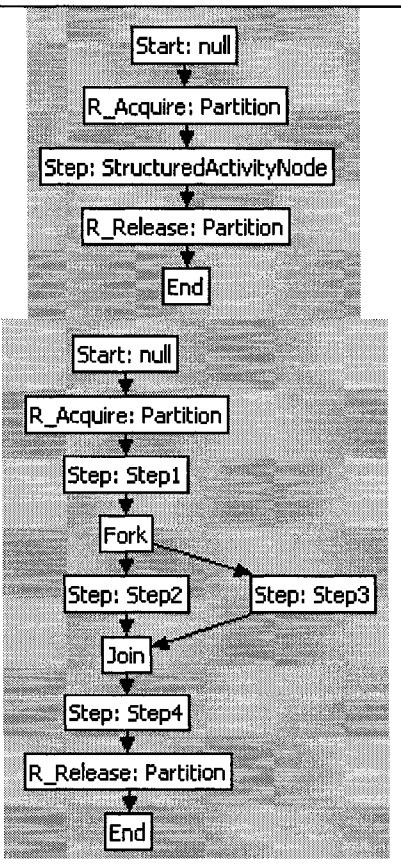
### 5.2.2 Test Cases

To verity the above features, the following test cases were designed, and a summary table shows how these test cases verify the features.

**Table 5-1** Test Cases and Transformation Results

| | Test Cases | Transformation Results |
|---|---|---|
| 1 |  |  |
| 2 |  |  <br>* Highlighted edge crossed between partitions, thus both ResourceAcquire and |

| | | |
|---|---|---|
| | | ResourceRelease are needed. |
| 3 |  |  <br>* Highlighted edge crossed between partitions, but only ResourceAcquire is needed. |
| 4 |  |  <br>* Highlighted edge crossed between partitions, but neither ResourceRelease nor ResourceAcquire is needed. |

| | | |
|---|---|---|
| 5 |  |  <br>* Highlighted edge crossed between partitions, but only ResourceRelease is needed. |
| 6 |  |  |

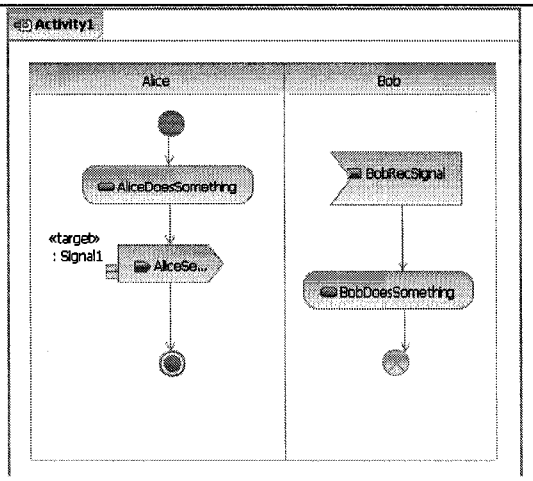
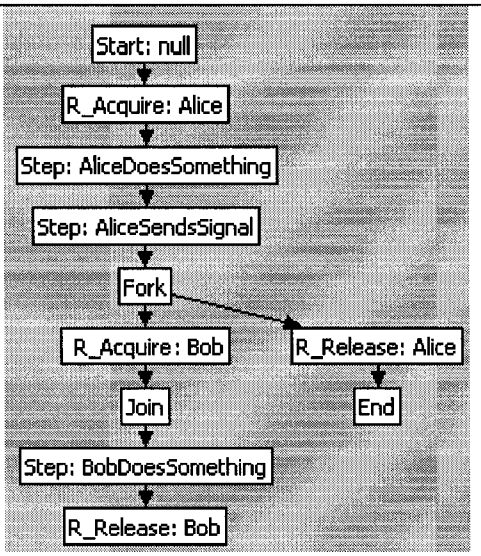
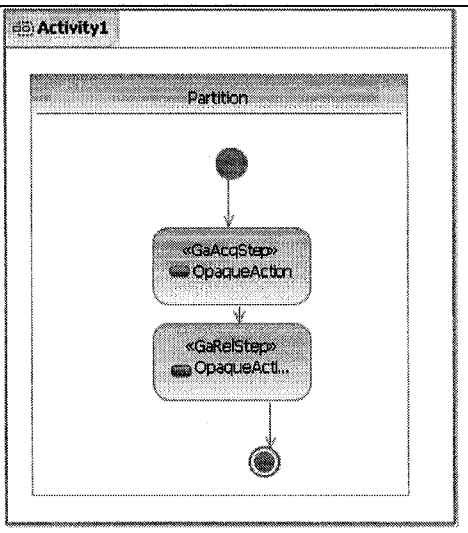
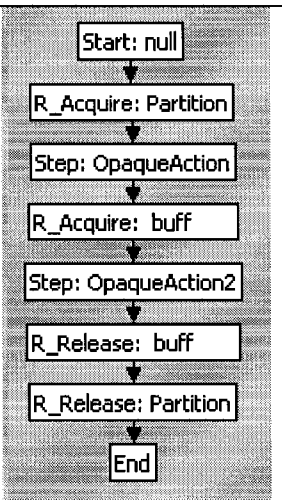
| | | |
|---|---|---|
| 7 |  |  |
| 8 |  <br>* Resource is assigned to buff by stereotype properties |  |

**Table 5-2** Verification Results of Test Cases and Features

| Features / Test Cases | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | | ✓ | | | | | | | | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **2** | ✓ | | ✓ | | | ✓ | | | | | |
| **3** | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | | |
| **4** | ✓ | ✓ | ✓ | ✓ | | | | | ✓ | | |
| **5** | ✓ | | ✓ | ✓ | | | | ✓ | | | |
| **6** | ✓ | | ✓ | ✓ | ✓ | | | | | | |
| **7** | ✓ | ✓ | ✓ | | | | | | | ✓ | |
| **8** | ✓ | | ✓ | | | | | | | | ✓ |

# Chapter 6 Case Study-Building Security System (With MARTE)

The previous chapters discussed the algorithm for the proposed transformation.. This chapters presents the application of the proposed UML2.0 to CSM transformation to a more a substantial example: a Building Security System that was introduced in [38] and used in other papers [41 ].

## 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 only takes one of the three scenarios as the example 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

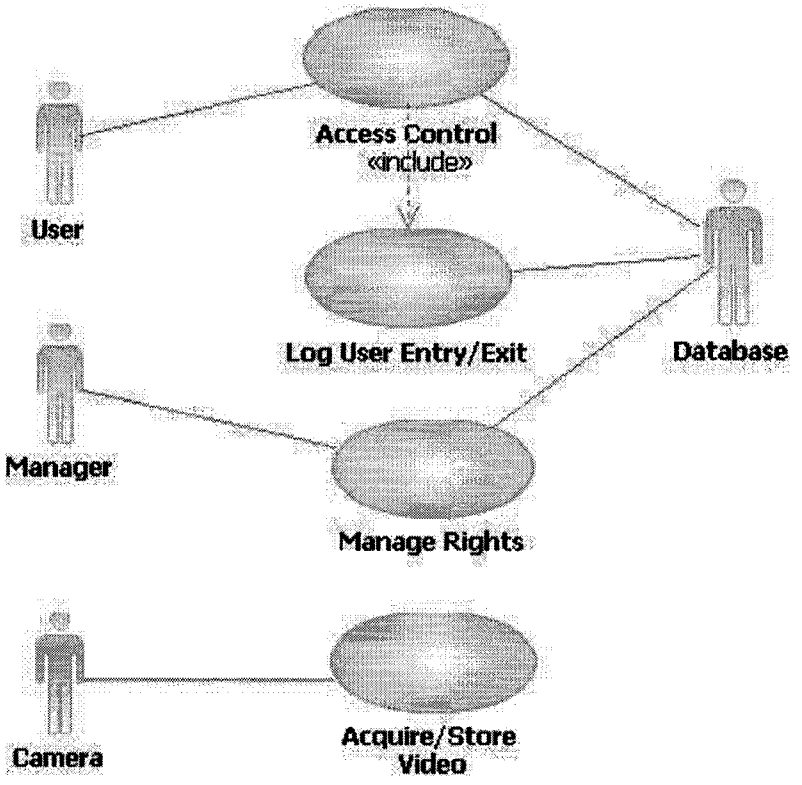



**Figure 6-1 Use Cases of Building Security System**

Two use cases in the system are selected for performance analysis, Acquire/Store Video and Access Control. 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. A performance requirement is that the system should be capable of taking care of 50 cameras.

## 6.3 Deployment Diagram

As illustrated in Figure 6-2, there are 3 nodes for running the Building Security System. There is a Camera node, Control node, and a DB node, all stereotyped as GaExecHost. In the deployment diagram the "SchedulableResource" stereotypes (drawn as triangles by the tool) are shown only for the Control artifact, but in fact apply to all the other artifacts (not shown to avoid diagram clutter). Examining the deployment diagram first, the nodes are stereotyped as ExecHost. As the Camera node is only symbolic, and is not represented in the design, it need not be stereotyped. The ProcessingRate attribute of the DataBaseNode is interpreted for performance as a factor, relative to a nominal processor, on which the hostDemand figures are based. The deployed objects are all artifacts, that are referenced by the Process stereotypes in the activity diagram. The reference to the artifact (rather than to the class or instance) is to resolve the deployment of active objects.

The bufferpool artifact stands for a set of buffers at run-time, and the number of buffers $Nbuffers is a significant parameter for the performance of the system.

Note that RSA does not show tagged values, and even has problems to show some of the stereotypes (for instance, it does not show stereotypes for CommunicationPath).

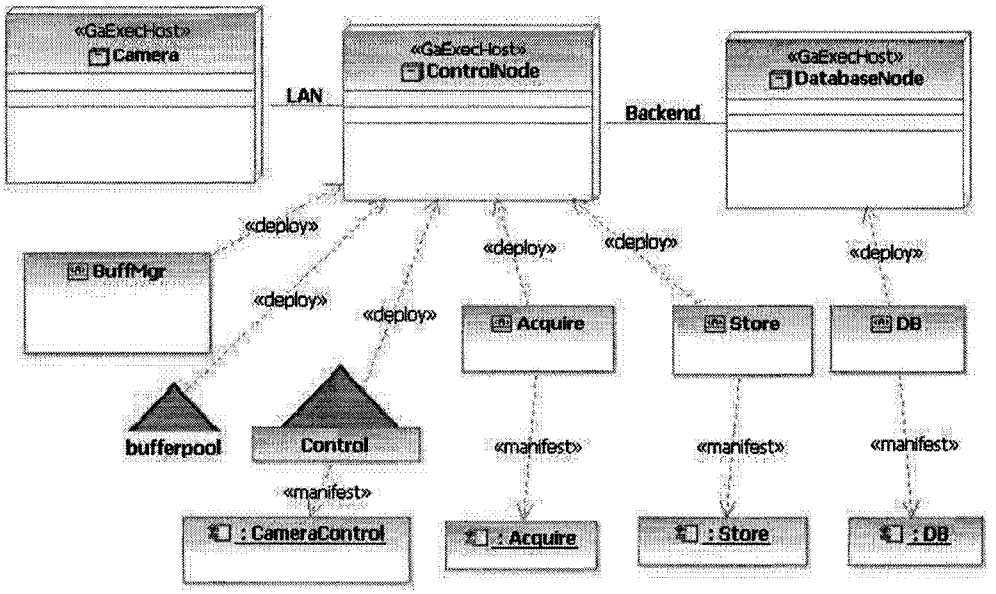



**Figure 6-2 Deployment Diagram of Building Security System**

## 6.4 Transforming Acquire/Store One Image scenario

As illustrated in Figure 6-3, the acquiring and storing of one image from the camera is described in a UML 2.0 activity diagram. There are 4 partitions, each representing a component instance, (annotated as a PaRunTInstance).

The figure illustrates the RSA version of the activity diagram. The tool does not show the attributes (tagged values) of stereotypes all at once; we can see instead the attributes of a selected stereotyped element in a special window. In order to show how the tagged values are transformed from UML stereotyped elements to CSM elements, Figure 6-4 illustrates displays the attribute and their values for action AllocBuf (on the left side) and the translated step in CSM expressed in XML (on the right side). The whole transformed CSM model is shown as Figure6-5.

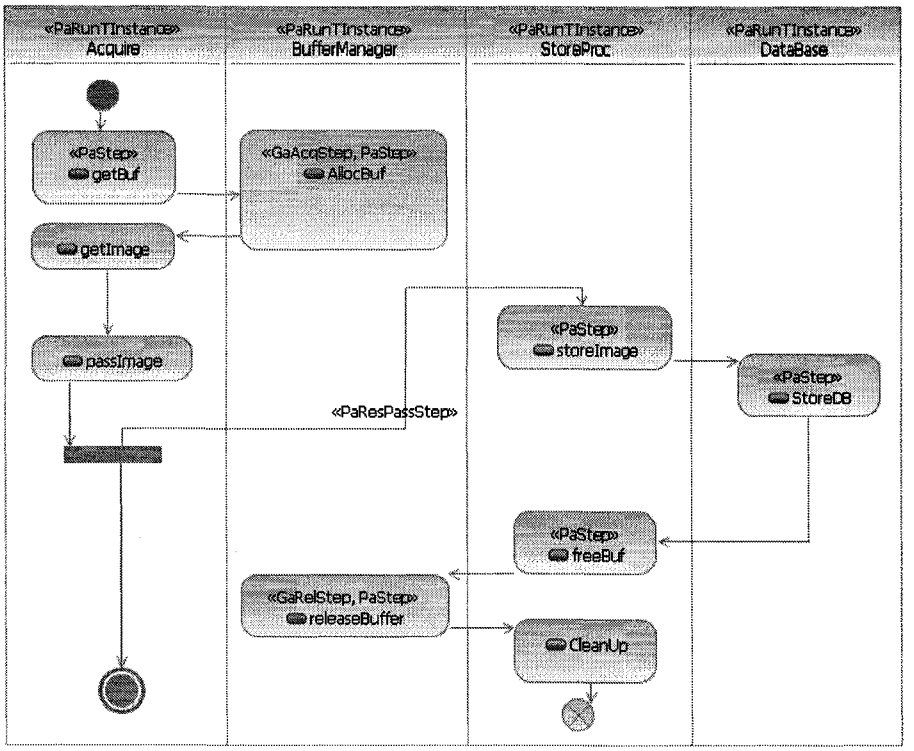



**Figure 6-3 Activity Diagram (RSA Version) of Scenario of Acquire/Store**

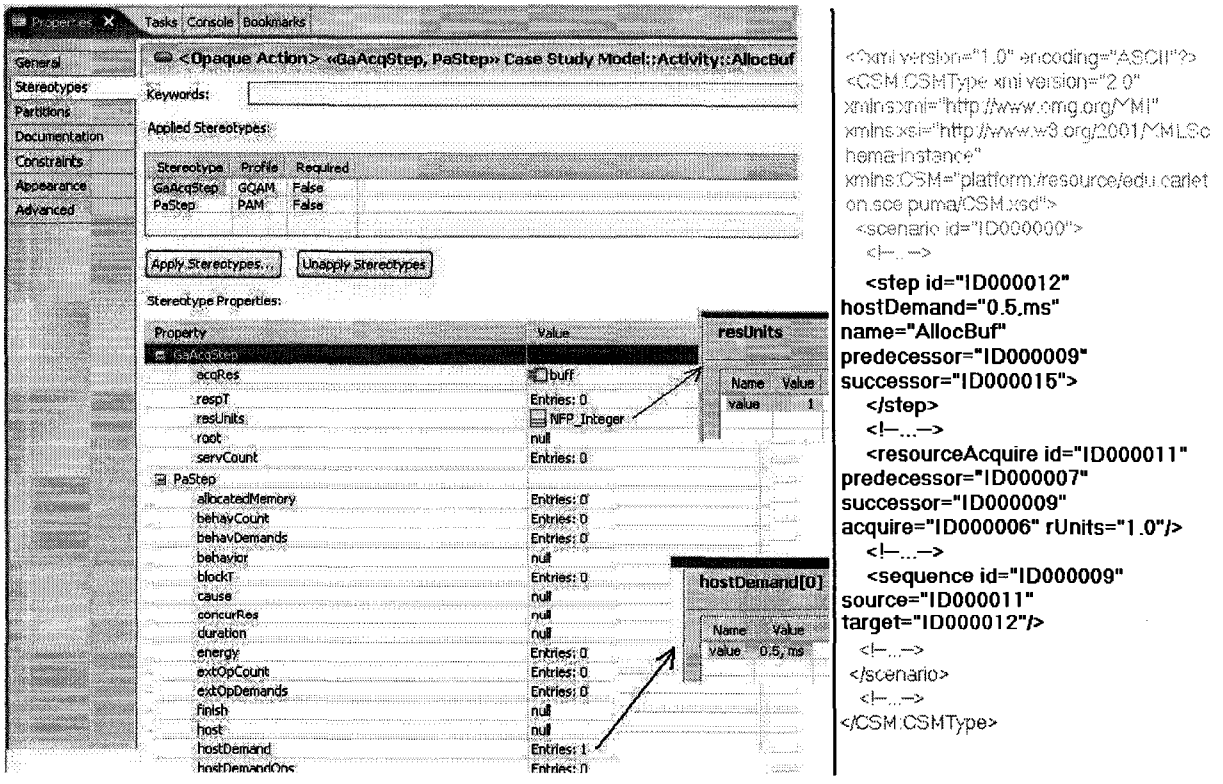



**Figure 6-4 Example of Tagged Value Transformation for activity AllocBuf**

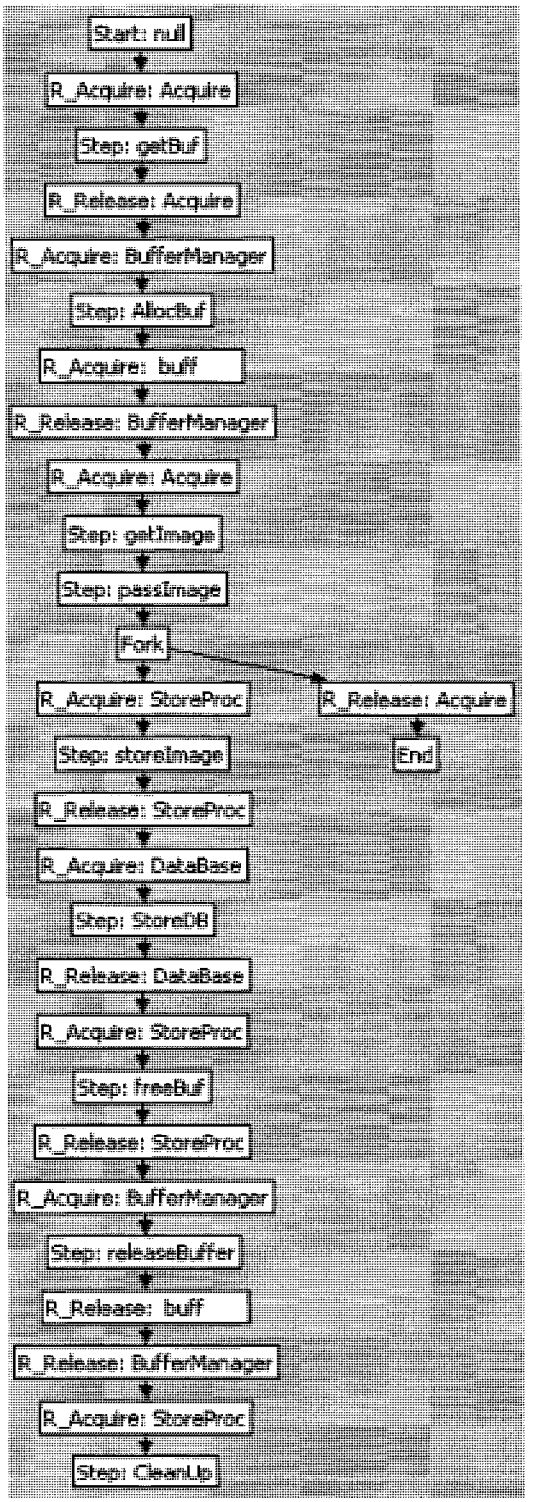



**Figure 6-5 Transformation Result (CSM) of Scenario of Acquire/Store**

## 6.5 Transforming the AccessControl scenario

The Access Control scenario describes processing a user request to enter the building. The user inserts a card into the CardReader which reads the card information and passes it onto the AccessController. TheAccessController checks the card information against the access rights it gets from the Database (which may require reading from the Disk some of the time). Depending on how the user's access rights check out, the AccessController signals either the DoorLock or the Alarm and then logs the access attempt with the Database. In the case of a successful access the DoorLock sends an enterBuilding notification to the user. The Activity Diagram for the scenario is shown in Figure 6-6. The result of the transformation is the CSM scenario shown in Figure6-7.

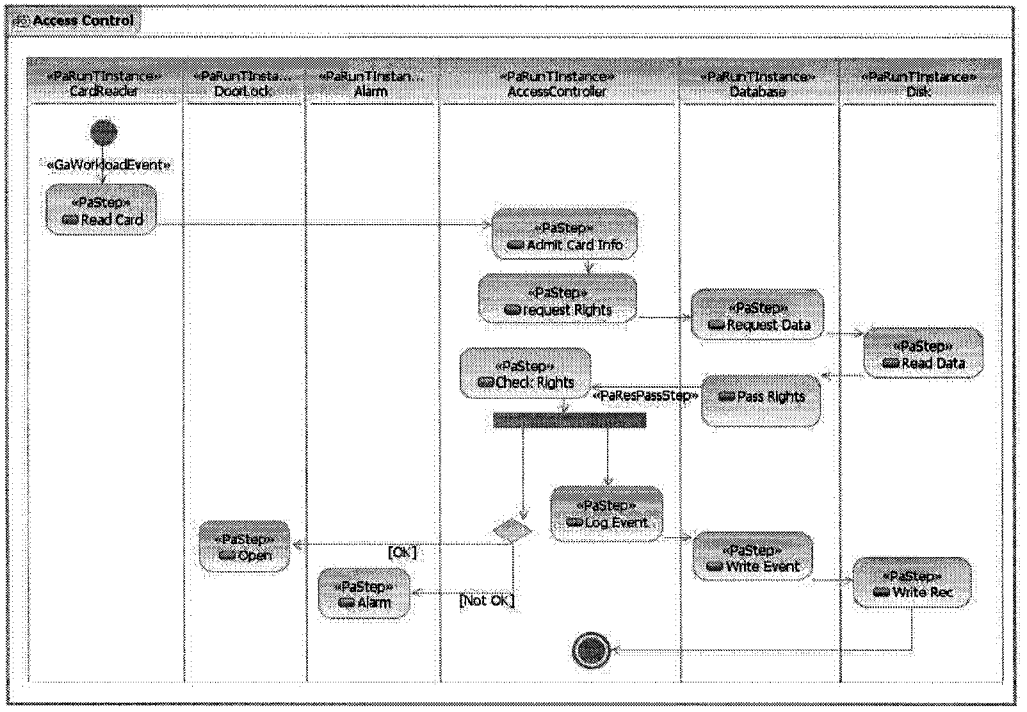



**Figure 6-6 Activity Diagram (RSA Version) of the Scenario of Access Control**

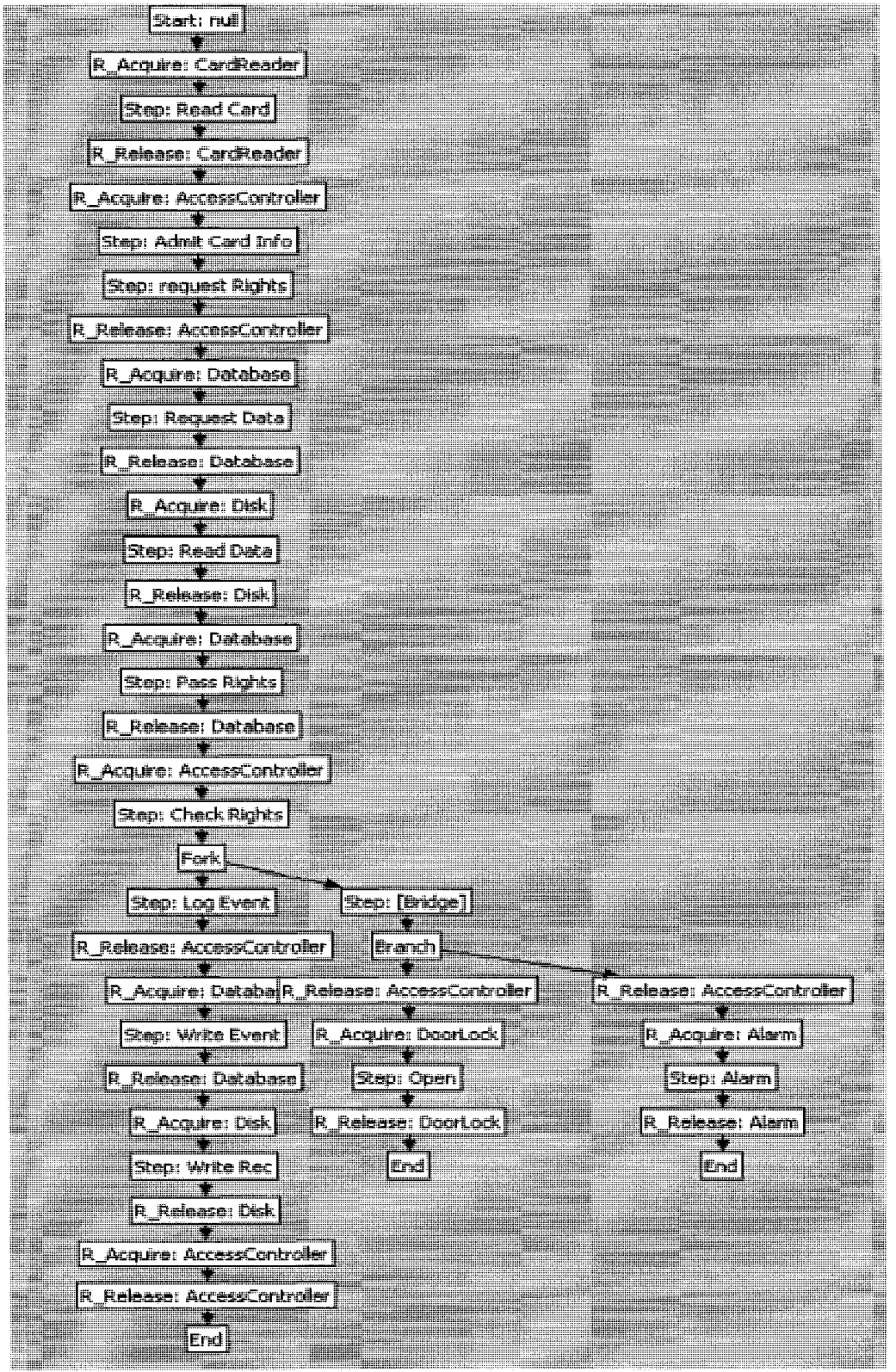



**Figure 6-7 Transformation Result (CSM) of the Scenario of Access Control**

# Chapter 7 Conclusions

## 7.1 Completed Work

The thesis proposed and implemented a Java based algorithm for the transformation from UML 2.0 annotated activity and deployment diagrams to the Core Scenario Model (CSM) as part of the PUMA project [41]. 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 using the recently adopted MARTE Profile for annotating the UML model with performance information.

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 4.2, which invokes the transformation rules presented in section 4.3. We assume that the construction of the input models follows the patterns presented in section 3.1.4. If the input model does not follow these 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 + MARTE 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.

This work is a step in the larger PUMA research project, aiming at deriving Core Scenario models from UML 2.0+MARTE models. Although the UML 2.0 standard, the XMI standard, the Core Scenario Model Schema and the UML Profile for MARTE are still evolving, the conceptual approach proposed in this thesis will be applicable to any future versions.

## 7.2 Limitation and Future Work

Although the work done for this thesis is quite thorough in realizing the transformation from UML 2.0 Activity Diagrams to Core Scenario Models, the following is a list of limitations, which also represent possibilities for future work.

As mentioned before, only the subset of UML2.0 Activity Diagram satisfying specific assumptions is guaranteed to have expected transformation results. These assumptions include one-dimensional partition, which brings limitations to the source models. Future work needs to be done in order to deal with multi-dimensional partitions. Another limitation is the fact that each activity diagram has to be acyclic in order for the traversing algorithm described in Chapter 4 to work. A conceptual approach for removing this limitation is described at the end of Chapter 4.

The research deals with most metaclasses within the UML2.0 metamodel for Activity Diagrams. However, one main feature of UML2.0, Exception Handling, is not discussed in this thesis. This is a main limitation of the research and future work should take care of it.

Some limitations are due to the RSA tool, such as that structured nodes are detailed on the same diagram, and that swimlanes are not allowed in structured nodes. These restrictions limit not only the application of the transformation, but also the modeling power of activity diagrams. Future work related to multiple and nested scenarios needs to be done based on newer RSA versions, or on other tools.

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.

As mentioned in Chapter2, there is a new MOF-QVT standard proposing a specialized language for model transformations. A direction for future research is to use specialized model transformation languages rather than a general purpose language as Java for the transformation from UML+MARTE to CSM, and then to different performance models.

# References

[1] Booch, G., Rumbaugh, J., Jacobson, I., "The Unified Modeling Language User Guide", Addison Wesley, 1999

[2] Smith, C.U., "Performance Engineering of Software Systems", Reading Mass., Addison Wesley, 1990

[3] Woodside, C.M., Franks, G., and Petriu, D.C., "The future of software performance engineering," Future of Software Engineering 2007, May 2007.

[4] Object Management Group, UML Profile for Schedulability, Performance, and Time,version 1.1, See http://www.omg.org/cgi-bin/doc?formal/2005-01-02

[5] Object Management Group, UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Beta 1, http://www.omg.org/docs/ptc/07-08-04.pdf, August 2007

[6] Woodside, C.M., Neilson, J.E., Petriu, D.C. & Majumdar, S., "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-Like Distributed Software", IEEE Transactions on Computers, Vol. 44, Nb.1, January 1995

[7] Williams, L.G., & Smith, C.U., "Performance Evaluation of Software Architectures" in Proc of WOSP'98, Santa Fe, New Mexico, USA, 1998

[8] King, P., & Pooley, R. "Derivation of Petri Net Performance Models from UML Specifications of Communication Software", Proc. Of XV UK Performance Engineering Workshop, 1999

[9] Pooley, R., "Using UML to Derive Stochastic Process Algebra Models" Proc of XV UK Performance Engineering Workshop, 1999.

[10] UML 2.0 Superstructure 2002, http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-02.pdf

[11] Rational Software Architect, See http://www-306.ibm.com/software/awdtools/architect/swarchitect/

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

[13] Booch, G., Object Oriented Analysis and Design with Applications (2nd Ed.), Benjamin/Cummings, Redwood City, 1994.

[14] Jacobson, I., Christerson, M., Johnsson, P. and Overgaard, G. Object-Oriented Software Engineering: A Use Case Driven Approach, Prentice–hall, Englewood Cliffs, NJ, 1992

[15] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorensen, W., Object-Oriented Modeling and Design. Prentice-Hall, Englewoods Cliffs, 1991

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

[17] Cook, S., and Daniels, J., Designing Object Systems: Object-Oriented Modeling with Syntropy, Prentice-Hall, Hemel Hempstead, 1994

[18] Object Management Group, UML 2.0 Infrastructure 2003, http://www.omg.org/cgibin/apps/doc?ptc/03-09-15.pdf

[19] Object Management Group: The MDA Guide Version 1.0.1, document OMG/2003-06-01, 2003.

[20] Kleppe, A., Warmer J and Bast, W, (2003). MDA explained: the model driven architecture: practice and promise. Addison-Wesley, Boston

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

[22] Smith, C.U. and Williams, L., Performance Solutions: A practical guide to creating responsive, scalable software, Addison-Wesley, 2001

[23] http://en.wikipedia.org/wiki/Performance_engineering

[24] Garrido, J.L., Gea, M. A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling In Interactive Systems. Design, Specification, and Verification, Revised Papers of the 9th International Workshop (DSV-IS 2002), Rostock, Germany, June 12-14, 2002, pages 16-28. Volume 2545 of Lecture Notes in Computer Science ( P. Forbrig, Q. Limbourg, B. Urban, J. Vanderdonckt ,Eds.) , Springer Verlag, December 2002

[25] Reisig, W., Humboldt University of Berlin; An Informal introduction to Petri nets, in Lecture Notes in Computer Science, Vol. 1491: Lectures on Petri Nets I: Basic Models. Springer-Verlag, 1998.

[26] Gerard, S., (edited by): Report on SIVOES' 2004-SPT Workshop on the usage of the UML profile for Scheduling, Performance and Time Mai 25th, 2004, Toronto, Canada.

[27] Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), RFP. 2005. OMG document: realtime/05-02-06.

[28] Object Management Group. MDA Guide Version 1.0.1. 2003.

[29] Object Management Group: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. 2004. OMG document ptc/04-09-01.

[30] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, "Model-based performance prediction in software development: a survey" IEEE Transactions on Software Engineering, Vol 30, N.5, pp.295-310, May 2004.

[31] Cortelessa, V., Mirandola, R., Deriving a Queueing Network based Performance Model from UML Diagrams, in Proc. of 2nd ACM Workshop on Software and Performance, pp.58-70, Ottawa, Canada, Sept. 2000.

[32] Petriu, D.B., Woodside, C.M., "Software Performance Models From System Scenarios", Performance Evaluation, Vol No 61, pp.65-89, 2005.

[33] Kahkipuro, P., UML-Based Performance Modeling Framework for Component-Based Distributed Systems, in: R.Dumke et al.(eds), Performance Engineering, LNCS Vol. 2047, Springer, Berlin Heidelberg New York (2001) 167-184.

[34] Lopez-Grao, J. Merseguer, J. Campos, "From UML Activity Diagrams To Stochastic Petri Nets: Application To Software Performance Engineering," in 4th Int. Workshop on Software and Performance (WOSP 2004), Redwood City, CA, (2004), 25-36.

[35] Cavenet, C., Gilmore, S., Hillsron, J., Kloul, L. and Stevens, P. "Analysing UML 2.0 activity diagrams in the software performance engineering process," in Proc. 4th Int. Workshop on Software and Performance (WOSP 2004), pp. 74-83, Redwood City, CA, Jan 2004.

[36] Amer, Hoda, "Automated Transformation of UML Software Specifications into LQN Performance Models using Graph-Grammar Techniques", M.Eng. Thesis, May 2001.

[37] Schurr, A., "Introduction to PROGRES, an Attributed Graph Grammar Based Specification Language", In: Nagl, M., (ed): Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Sciences, Vol. 411, pp 151-165, 1990

[38] Petriu, D.C., Shen, H., "Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications", in Computer Performance Evaluation - Modelling Techniques and Tools, (T.Fields, P. Harrison, J. Bradley, U. Harder, Eds.) LNCS Vol. 2324, pp.159-177, Springer, 2002.

[39] Gordon Gu, D.C. Petriu, "XSLT transformation from UML models to LQN performance models", Proc. of the 3rd ACM Workshop on Software and Performance WOSP'02, pp.227-234, Rome, July 2002.

[40] Ping Gu, Gordon, Petriu, Dorina C., "From UML to LQN by XML algebra-based model transformations", Proc. of the 5th ACM Workshop on Software and Performance, pp. 99-110, Palma, Spain, July 11-14, 2005.

[41] C.M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, J. Merseguer, "Performance by Unified Model Analysis (PUMA)", Proc. of The 5th ACM Workshop on Software and Performance, pp. 1-12, Palma, Spain, July 11-14, 2005.

[42] Object Management Group, Meta Object Facility (MOF) Specification, Version 1.4, see http://www.omg.org/cgi-bin/doc?formal/2002-04-03

[43] Object Management Group, MOF QVT Final Adopted Specification, 2005, see http://www.omg.org/docs/ptc/05-11-01.pdf

[44] Lazowska, E.D., Zahorjan, J., Graham, G.S. and.Sevcik, K.C., Quantitative System Performance, Computer System Analysis Using Queueing Network Models, Prentice-Hall, Inc., February1984.