uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Shahbaz Maqbool
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Transformation of a Core Scenario Model and Activity Diagrams into Petri Nets

TITRE DE LA THÈSE / TITLE OF THESIS

Gregor Bochmann
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Daniel Amyot

Dorina Petriu

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

The undersigned recommend to the Faculty of
Graduate Studies and Research acceptance of the thesis

# TRANSFORMATION OF A CORE SCENARIO MODEL AND ACTIVITY DIAGRAMS INTO PETRI NETS

Submitted by Shahbaz Maqbool
in partial fulfillment of the requirements for the degree of
Masters in Computer Science

---

**Director, School of Computer Science**

---

**Dr. Gregor v.Bochmann, Thesis Supervisor**

University of Ottawa
September 2005

# TRANSFORMATION OF A CORE SCENARIO MODEL AND ACTIVITY DIAGRAMS INTO PETRI NETS

**By**

**Shahbaz Maqbool**

A thesis submitted in partial fulfillment
of the requirements for the degree of

Masters in Computer Science*

Under the auspices of the Ottawa-Carleton Institute for Computer Science

School of Information Technology and Engineering

University of Ottawa

Ottawa, Ontario, Canada
September 2005

---

* The Masters program in Computer Science is a joint program with Carleton University, administered by the Ottawa Carleton Institute for Computer Science.

© Shahbaz Maqbool, Ottawa, Canada, 2005

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

*Your file Votre référence*
*ISBN: 0-494-11341-3*
*Our file Notre référence*
*ISBN: 0-494-11341-3*

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

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

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

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

---

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

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

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

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

Canada

# Abstract

For any software development project it is important to capture the requirements in a clear and concise manner. Standardization efforts, such as the development of version 2 of the Unified Modeling Language (UML) by the Object Management Group, and the development of the User Requirements Notation (URN) by the International Telecommunication Union, propose visual languages for capturing requirements in terms of scenario notations. Activity Diagrams and Use Case Maps (UCM) are examples of such scenario languages in UML and in URN, respectively. The developers of these languages have concentrated on the visual notations and only a small amount of effort has been spent in defining precise and formal semantics for these languages.

Core Scenario Model (CSM) is a step towards defining formal semantics to the scenario based languages like UCM, Activity Diagrams and Interaction Diagrams. It includes common scenario information found in the UCM notation and in UML 2.0 Activity Diagrams and Interaction Diagrams, and has been developed as an intermediate language before transformation into formal languages like Petri Nets, Layered Queuing Network etc.

The thesis proposes a transformation method that takes UML Activity Diagrams as input and generates equivalent Petri Nets as output. The transformation approach takes into account the concurrency characteristics of Activity Diagrams.

The thesis also proposes a method for transforming a Core Scenario Model (CSM) representation into equivalent Petri Nets. A Java tool was designed and built for realizing the proposed transformation from CSM to Petri Nets. The application takes as input XML files produced by an existing tool, which contain CSM in XML format. The Petri Nets produced by our transformation is in XML format. It can be used for validating the original models by simulation. The results from this analysis can be traced back to improve design decisions.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

I am most grateful to my supervisor, Professor Gregor v.Bochmann, for his encouragement, guidance, support and patience throughout this research. Without his help, advice and trust, my studies in this area would not have been possible.

I would like to thank University of Ottawa, the School of Information Technology and Engineering for giving me the opportunity to do my thesis work. I would also like to thank the members of AIAS lab for creating a wonderful working environment and for their technical support.

I am very grateful to my wife Ambreen Shahbaz, for her moral support and constant encouragement. She and my son, Ahmad Shahbaz were very patient throughout many days and weekends where they missed fun because dad had some work to do. I am so grateful to my father, mother, father in law, mother in law and all of my brothers and sisters for their support and understanding. Their prayers were also a major factor in the success of my work. I would like to dedicate this thesis to my father, Maqbool Hussain. His dedication to raising us in the best way he could is the biggest reason of success of my brothers, sisters, and me.

I extend my appreciation to all my friends for their helpful comments and suggestions.

# Glossary

| | |
|---|---|
| **API.** | Application Program Interface |
| **CPN.** | Colored Petri Nets |
| **CSM.** | Core Scenario Model |
| **DOM.** | Document Object Model |
| **DTD.** | Document Type Declaration |
| **ITU.** | International Telecommunication Union |
| **LQN.** | Layered Queuing Networks |
| **MDA.** | Model Driven Architecture |
| **MOF.** | Meta Object Facility |
| **OMG.** | Object Management Group |
| **PN.** | Petri Nets |
| **SPE.** | Software Performance Engineering |
| **UCM.** | Use Case Maps |
| **UML.** | Unified Modeling Language |
| **URN.** | User Requirements Notations |
| **XMI.** | XML Metadata Interchange |
| **XML.** | Extensible Markup Language |
| **XSL.** | XML Style Sheet Language |

# Chapter 1 Introduction

## 1.1 Background / Context

For any software development project it is important to capture the requirements in a clear and concise manner. Requirements engineering addresses the development and validation of methods for eliciting, representing, analyzing, and confirming system requirements and with methods for transforming requirements into specifications for design and implementation. The last few decades have resulted in an evolution of software design methodologies towards requirements engineering and high-level design, where the errors are the most costly for software producers.

The success of large complex, reactive and safety- critical software is largely dependent on capturing precise and correct requirements. Standardization efforts, such as the development of version 2 of the Unified Modeling Language (UML) by the Object Management Group (OMG), and the development of the User Requirements Notation (URN) by the International Telecommunication Union (ITU-T), propose visual languages for capturing requirements in terms of scenario notations. Activity Diagrams (ADs) and Use Case Maps (UCMs) are examples of such scenario languages in UML and in URN, respectively. These scenario-based requirement specification techniques are very attractive and user-friendly. Their application to the requirements at early stages of the design process helps in

1) Preparing concise, descriptive, maintainable and consistent documents,
2) Preparing design specifications,
3) Developing models at lower cost and improving their correctness and traceability with respect to requirements,
4) Faster implementation at lower cost and improving their correctness and traceability with respect to requirements, and
5) Overall validation and evolution of the system.

These tasks pave the way to the generation of complete, consistent, and unambiguous specifications of system behavior that are well suited for design and implementation activities.

Although the above mentioned languages provide an accessible visualization of models, they lack formal precise semantics. Up till now significant research has been devoted to the development of visual requirements by the OMG and the ITU, and only a small amount of effort has been spent in defining precise and formal semantics for these visual languages. In my research work, the focus is to transform UCM (Use Case Maps) and UML (Unified Modeling Language) Activity Diagrams to Petri nets. Petri net is a graphical and mathematical modeling language. It consists of places, transitions, arcs and tokens. Input arcs connect places with transitions, while output arcs start at a transition and end at a place. Petri nets are a promising tool for describing and studying systems that are characterized as being 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, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems.

The Core Scenario Model (CSM) [24] notation has been developed for functional analysis and preliminary performance analysis of software systems. We were involved in the development of CSM and proposed some important concepts like refinement of a Step into a Scenario by proposing mapping of inputs and outputs of such Steps, alternative sets of inputs and outputs to Step etc, to be incorporated in the metamodel. With the CSM, a two-step transformation process is proposed. In the first step it proposes that a CSM be generated from the above mentioned scenario languages and in the second step this CSM can then be transformed into formal languages like Petri Nets, Layered Queuing Network etc.

Figure 1.1 represents different possibilities of transformation of activity diagrams and UCMs into Petri Nets. The transformation can either be in one step i.e. from UCM and

Activity diagrams to Petri nets or in two steps, first to CSM and then to Petri Nets. In a two step approach the performance information in the UML/UCM models can be gathered and added to the CSM and the resulting model can then be transformed to Petri Nets. The results achieved from the functional and performance analysis can be fed back for making critical design decision.

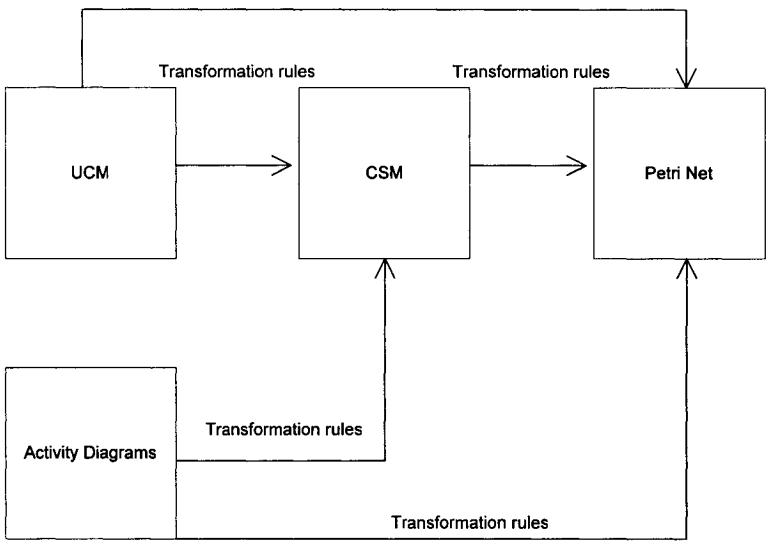



**Figure 1.1:** Different possibilities for transformations

## 1.2 Objective and Motivation

It is well known fact that the cost of requirements errors is much lower when found at the requirements and design stages and much higher when found during the implementation. So validation of requirements specifications at early stages of software development is very important. It is the process of checking whether the requirements specifications meet the intentions and expectations of the stakeholders. Several approaches have been proposed for validating the requirements of software projects but in many design processes we simply skip these tasks and move from informal requirements directly to component-based specifications. The use of notation based languages like Use Case Maps [1], UML Activity Diagrams [26] etc has increased a lot particularly in the last decade. These languages are either informal or semi-formal and cannot be used for

validation of the system. In order to validate the requirements it is necessary to translate the requirements from informal languages to formal languages like Petri Nets, Layered Queuing Networks etc. The results of the functional analysis should be fed back to have a precise requirements for the system to be built. This would allow the design process to focus on the main functional aspects of the system to be specified, and hence better cope with some of the complex problems related to the design, documentation, validation, and maintenance of systems and standards.

The goal of the thesis is to devise a method for transforming Activity Diagrams and Core Scenario Models into Petri Net. An implementation for automatic transformation of CSM to Petri Nets is also a part of this thesis. Both the CSM (input) and Petri Net (output) are represented in XML format.

## 1.3 Scope and Contribution of the Thesis

The thesis proposes a method to convert a UML Activity diagrams to Petri nets. It also proposes a tool for converting the functional aspects of the Core Scenario Model (CSM) to Petri Nets. The functional aspects include concepts like: when will a scenario start executing, when will it terminate, how the different activities are sequenced, are there any activities which can be done in parallel, are there any constraints on the activity (preconditions and postconditions), what will be the output of an activity, etc. CSM is discussed in detail in Chapter 4. The contributions of the thesis are summarized as follows:

- Define transformation rules from Activity Diagrams UML 2.0 to colored Petri Nets.
- Define transformation rules from the CSM to Petri Nets. Identify the metamodel classes/objects used to represent the CSM, and express each transformation rule in terms of Petri Net metamodel objects, their attributes and relationships.
- Design, implement and test a tool that realizes the CSM to Petri Net transformation. The tool takes as input XML files produced by an UCMNav tool, which contain CSM in XML format. The input CSM is transformed into

equivalent Petri Nets, which are expressed also in XML format according to the format for Colored Petri Net used with CPN tools [12].

- Provided some example applications.

## 1.4 Thesis outline

This thesis is organized as follows:

**Chapter 2** provides an overview of the background information required for this thesis, such as Use Case Maps, Unified Modeling language, UML metamodel, Model Driven Architecture, UML 2.0 Activity Diagrams, Extensible Markup Language, Document Type Definition, XML Schema, Document Object Model and Petri Nets. It also reviews some of the tools being used for XML parsing and for modeling colored Petri Nets.

**Chapter 3** presents an overview of the Core Scenario Model (CSM), its metamodel and XML representation. It also discusses the benefits of using CSM.

**Chapter 4** describes how the concepts introduced in Activity Diagrams UML 2.0 can be mapped on Petri Nets. It also describes some simplification rules.

**Chapter 5** describes how the concepts of Core Scenario Model can be mapped to Petri Nets.

**Chapter 6** presents a case study that is developed to validate and verify the transformation process.

**Chapter 7** concludes the thesis research, summarizes contributions of the thesis, and identifies directions for future research.

# Chapter 2 Literature Review

This chapter presents an overview of the background information related to the thesis, such as Use Case Maps (UCM), Unified Modeling Language (UML), the Extensible Markup Language (XML), Petri nets, UML design tools and the Core Scenario Model.

## 2.1 Use Case Maps

The Use Case Map (UCM) notation was developed at Carleton University by Professor Buhr and his team [11], and it has been used for the description and understanding of a wide range of complex applications since 1992. This section provides an overview of Use Case Maps.

### 2.1.1 UCM Notation

UCM is a very simple notation for requirements specification, design and testing of software systems. It represents scenarios as paths. A Path has a Start Point and End Point. Components are used to show logical or functional entities like hardware, software, actors, etc.

Figure 2.1 illustrates four basic notation elements of UCMs: start points, responsibilities, end points, and components.

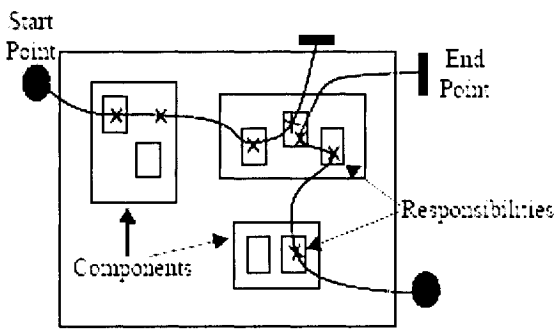



**Figure 2.1:** Basic Notations in UCM (taken from [2])

If a responsibility is shown inside a component it is said to be bound to a component and in this case, the component is responsible to perform the action, task, or function represented by the responsibility.

An OR-Join is used to merge two or more incoming paths together while an OR-fork is used to split a path into two or more alternatives. Alternatives may have guards represented as labels in between square brackets.

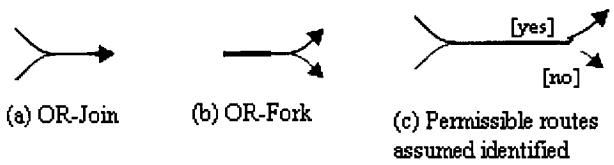



**Figure 2.2**: Shared Routes and OR-Forks/Joins (taken from [2])

An AND-join is used to synchronize two or more paths together while an AND-fork is used to split a path into two or more concurrent segments.

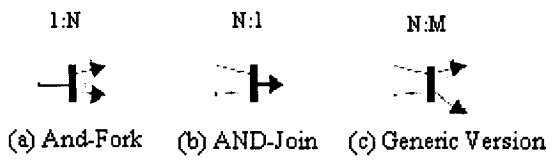



**Figure 2.3:** Concurrent Routes with AND-Forks/Joins (taken from [2])

There are different types of Components in UCM. Some of the important ones are illustrated in Figure 2.4. Rectangles are called teams and can contain any type of component. Parallelograms are called active components or processes, which usually imply a control thread. Rounded rectangles are called passive components or objects which are usually controlled. Dashed components are called slots and may be populated with different component instances at different times [1].

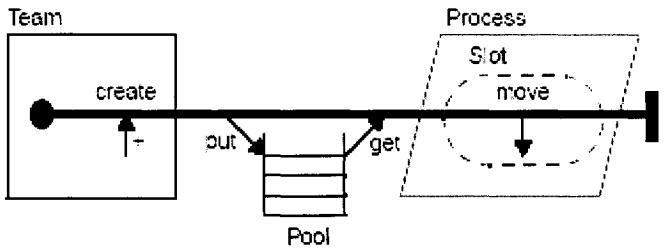



**Figure 2.4:** Dynamic Components and Dynamic Responsibilities (taken from [2])

If a UCM map becomes complex to be represented as a single map then Stubs can be used to define sub-maps. A stub contains separately specified sub-maps called plug-ins. A plug-in has further detail information about a given aspect of a scenario. Stubs are of two kinds as shown in Figure 2.5.

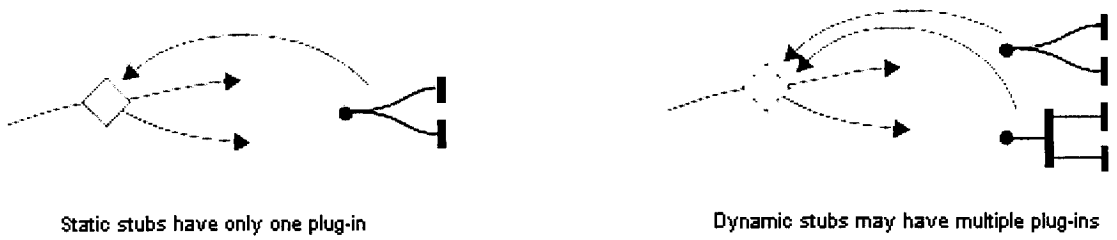



**Figure 2.5:** Stubs and Plug-ins (taken from [2])

- **Static stubs**: can contain only one plug-in and are represented as plain diamonds.
- **Dynamic stubs**: can contain more than one plug-ins and are represented as dashed diamonds.

Different paths in a UCM map may interact with each other synchronously or asynchronously. This has been shown in Figure 2.6

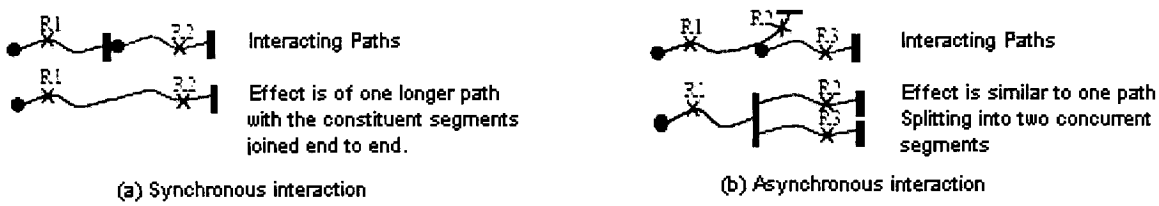



**Figure 2.6:** Path Interactions (taken from [2])

Some other important notational elements include

- **Timer**: are triggered by arrival of a specific event.
- **Abort**: are used to show termination of one path by another path.
- **Failure point**: are used to show potential failure points on a path.
- **Shared responsibility**: are used to represent a complex activity that involves negotiation between two or more components.

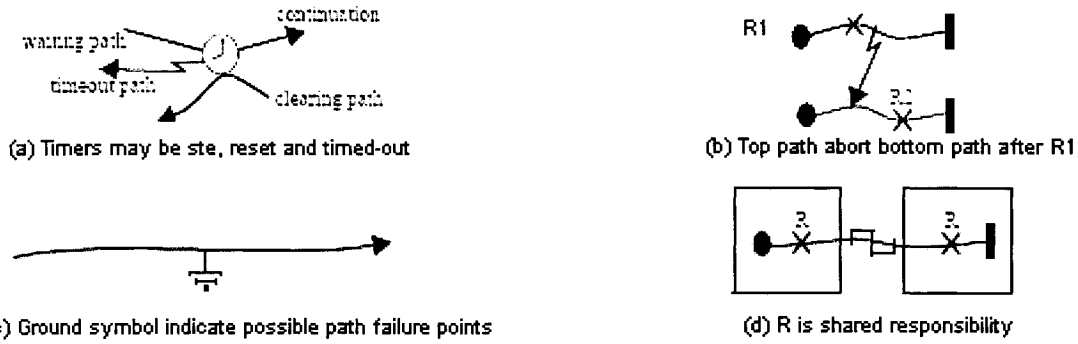



**Figure 2.7:** Timers, Aborts, Failures, and Shared Responsibilities (taken from [2])

### 2.1.2 UCM Navigator

UCMNav is the only available tool for editing Use Case Maps. It supports the UCM notation and the XML format [23]. It can be used for the creation, navigation, and maintenance of UCMs. A plug-in for generating Core Scenario Model from UCMs has also been developed recently [37]. This thesis uses the XML output of this plug-in and transforms it into an XML representation of Petri Nets. The output generated is compliant to the XML structure (DTD) shown in Appendix D.

## 2.2 Unified Modeling Language

The Unified Modeling Language (UML) developed by the Object Management Group (OMG) is one of the last decade's most important contributions to software engineering. In a relatively short time it has emerged as the industry standard for designing and visualizing software systems. It provides several kinds of diagrams, which allow the description of different aspects and properties of systems, like static and behavioral

aspects, interaction among system components and physical implementation details. Figure 2.8 shows the various version of UML introduced since its appearance.

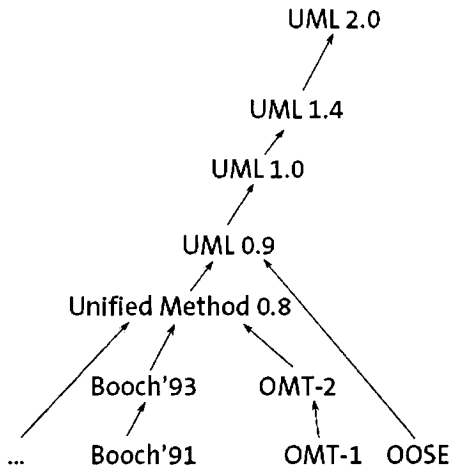



**Figure 2.8:** UML since inception

UML 2.0 diagrams can be divided into three major categories

1) **Structural Diagrams**

- Class diagrams
- Object diagrams
- Package diagrams
- Component diagrams
- Deployment diagrams
- Composite structure diagrams

2) **Behavioral Diagrams**

- Use case diagrams
- Activity diagrams
- State machine diagrams

3) **Interaction diagrams**

- ❖ Interaction overview diagrams
- ❖ Timing diagrams
- ❖ Communication (collaboration) diagrams
- ❖ Sequence diagrams

It is recognized that performance analysis should be integrated in the software development life cycle from the early stages. Reasons that are in favor of using performance analysis during the software development include the end user's expectations, cost control and the fact that performance requirements are better met if attention to performance problems is paid earlier rather than later. Since performance is a dynamic property, scenarios play a key role in determining a system's performance characteristics from its UML models. In UML, a scenario is an instance of a use case [29] and provides a means for the end user and the domain expert to state their expectations about the desired behavior of a system to its developers [9][10]. Scenarios are usually modeled either by activity diagrams or interaction diagrams. Activity diagrams provide the overall operations of a system.

Several revisions have been made since the original appearance of UML. The latest version UML 2.0 was officially adopted in October 2004 and provides some major improvements over UML 1.5.

### 2.2.1 Major Improvements in UML 2.0

As compared with earlier versions, UML 2.0 seems to have matured into a more complete language, with improved integration of the various parts. The major improvements in UML 2.0 are:

- ❖ New concepts for describing the internal architectural structure of Classes, Components and Collaborations by means of Part, Connector and Port.
- ❖ Introduction of inheritance of behavior in state machines and encapsulation of sub- machines through use of entry and exit points.

- An improved encapsulation of components through complex ports with protocol state machines that can control interaction with the environment.
- Integration of actions and activities and the use of flow semantics instead of state machines.
- Interactions are improved with better architectural and control concepts such as composition, references, exceptions, loops and alternatives and an improved overview with Interaction Overview Diagrams.

To deal with domain-specific issues UML defines three extension mechanisms: stereotypes, constraints and tagged values. These mechanisms can be used to extend the metaclasses defined by the UML metamodel. UML Profiles are packages that contain the definition of extended metaclasses. In other words profiles describe UML dialects. Profiles can be used to tailor the UML metamodel for different platforms (such as J2EE or .NET) or domains. A profile for dealing with the performance issues of the system was published in September 2003 [27].

### 2.2.1 UML Metamodel

The UML metamodel is defined as one of the layers of a four-layer metamodel architecture, depicted in Figure 2.9. The four layers are:

- $M_0$: domain-specific information
- $M_1$: model of the domain-specific information, e.g. in UML
- $M_2$: meta-model, e.g. definition of UML
- $M_3$: meta-meta-model, e.g. definition of the way that UML is defined, e.g. MOF.

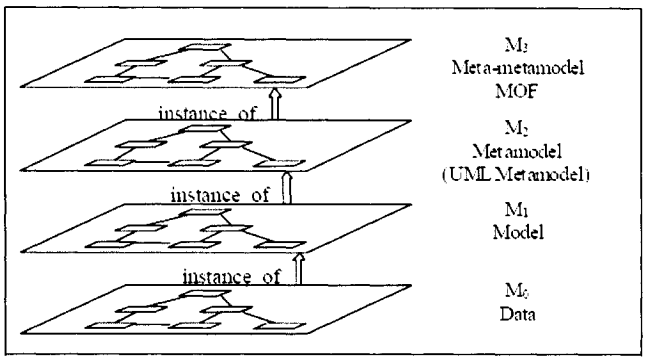



**Figure 2.9:** Four-layer Metamodel Architecture

The fundamental relationship between these layers is intended to be the instance-of relationship, which is clearly expressed in the UML specification. The $M_3$ level, Meta Object Facility (MOF), defines the basic concepts from which specific metamodels are created at the $M_2$ level. This includes the UML metamodel, which is regarded as being an instance-of the MOF meta-metamodel. Normal user models, created using the concepts of the UML, are regarded as residing at the $M_1$ level, and the ultimate run-time data is regarded as residing at the $M_0$ level.

## 2.3 Model Driven Architecture

The basic goal of model driven engineering (MDE) is that instead of writing code for an application, developers design their solutions in a modeling language. These designs will automatically generate code that with minimal customization can be mapped onto many different platforms. There will of course, be some platform-specific issues to be considered, but if successful, MDE promises to generate a big boost in productivity. Model-based code generation will make software faster to create, to update, to port, and to deliver.

In the last decade, there have been significant improvements in the ways in which software is built. Two important ones are

1) raised level of abstraction of the languages used to express behavior, and
2) increased level of reuse in system construction.

The Model Driven Architecture (MDA), also developed by the Object Management Group, is based on the above mentioned ideas for software development. It also introduces a new idea of design-time interoperability that ties these ideas together [31].

### 2.3.1 Raising the Level of Abstraction

An important development of last two decades is increased level of abstraction in software development. Initially programmers used to work with zeros and ones and with the introduction of higher level programming languages, like Java and C++, the level of abstraction were raised considerably. With this raised level of abstraction at which developers work, new tools to map from one layer to the next were also developed. Developers now write in a high-level language that can be mapped to a lower-level language automatically, instead of writing in the lower-level language that can be mapped to assembly language, just as our predecessors wrote in assembly language and had that translated automatically into machine language.

The next level of abstraction is the move to model-based development, in which we build platform-independent models.

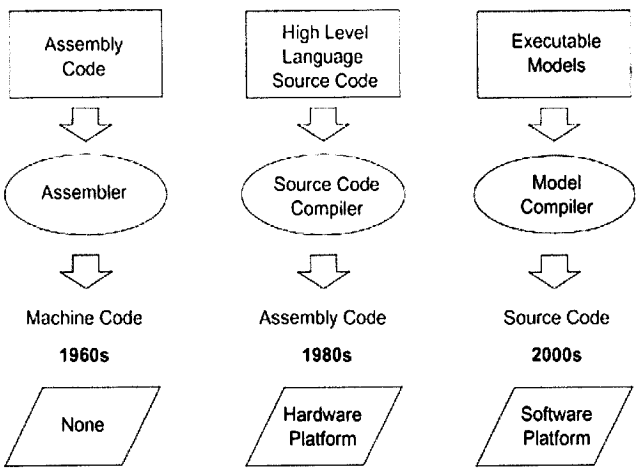



**Figure 2.10:** Raising the level of abstraction (taken from [31])

### 2.3.2 Raising the Level of Reuse

It is undoubtedly a fact that a major area of progress in the software industry has been reuse. In the object oriented paradigm, objects encapsulate a limited number of

subroutines and the data structures on which they operate [30]. By encapsulating data and subroutines into a single unit, the level of reuse is increased from the level of a single subroutine. A set of closely related objects, packaged together with a set of defined interfaces, form a component. A component enables reuse at a higher level, because the unit of reuse is larger.

Models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on relevant ones. All forms of engineering rely on models to understand complex, real-world systems. Models are used in many ways: to predict system qualities, reason about specific properties when aspects of the system are changed, and communicate key system characteristics to various stakeholders. The models may be developed as a precursor to implementing the physical system, or they may be derived from an existing system or a system in development as an aid to understanding its behavior. By dividing work into domain models we can expose interfaces at the level of rules.

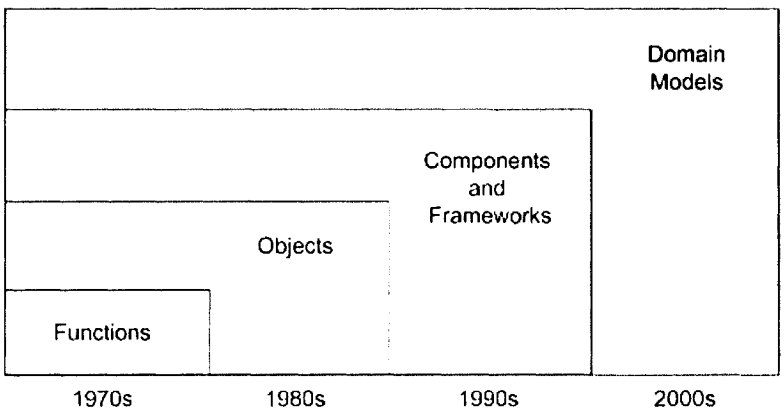



**Figure 2.11:** Raising the level of reuse (taken from [31])

If an application is built using models and some changes are required in the application, changes will be made in the application model by leaving implementation technologies alone. If a need to retarget an application to a different implementation environment arises just by selecting the models for the new environment we can regenerate code and

there will be no need to modify the application models. In this manner the costs of producing software will be lower; productivity will be higher, maintenance will be cheaper and each new model that gets built will be an asset that can be subsequently reused.

## 2.4 UML2.0 Activity Diagram

Activity diagrams are one of the behavioral diagrams in UML as already mentioned. They focus on the sequence, conditions, and inputs and outputs for invoking other behaviors. Activity Diagrams are primarily used to describe behavior of systems. They can be used to model an operation associated with a use case or a class. Figures 2.12 and 2.13 show the metamodel of Activity Diagram UML 2.0 [26]. Activities contain nodes connected by edges to form a complete flow graph. Control and data values flow along the edges and are operated on by the nodes, routed to other nodes, or stored temporarily. There are three kinds of nodes in activity models:

1. Executable / Action nodes operate on control and data values that they receive, and provide control and data to other actions.
2. Control nodes route control and data tokens through the graph.
3. Object nodes hold data tokens temporarily as they wait to move through the Activity diagram.

Activity nodes are connected by two kinds of directed edges:
1. Control flow edges connect actions to indicate that the action at the target end of the edge cannot start until the source action finishes. Only control tokens can pass along control flow edges.
2. Object flow edges connect objects nodes to provide inputs to actions. Only objects and data tokens can pass along object flow edges.

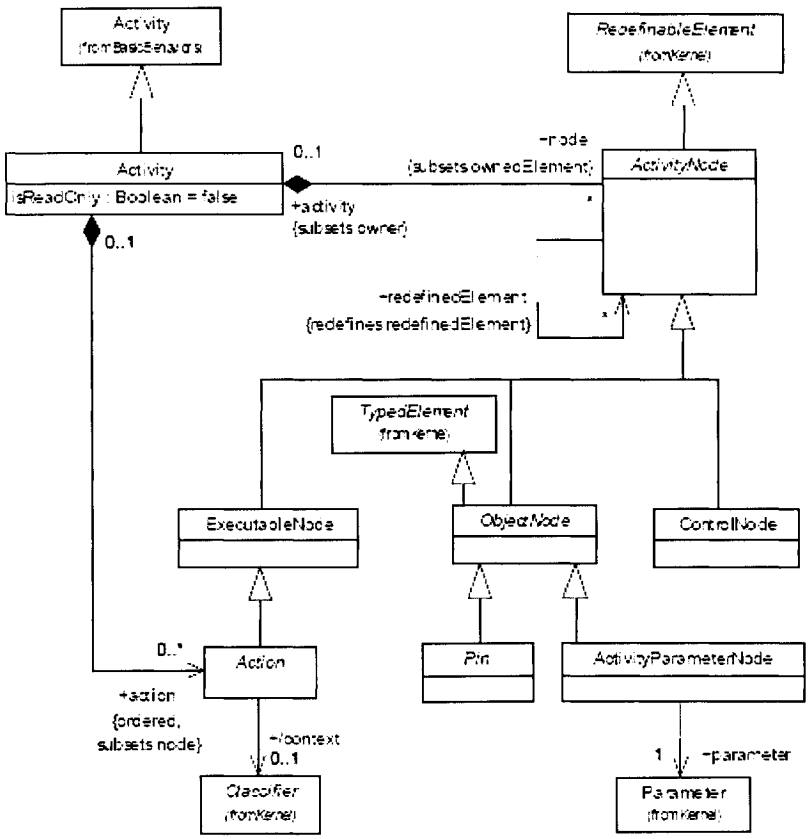



**Figure 2.12:** Class diagram for Activity Nodes in Activity Diagram (taken from [26])

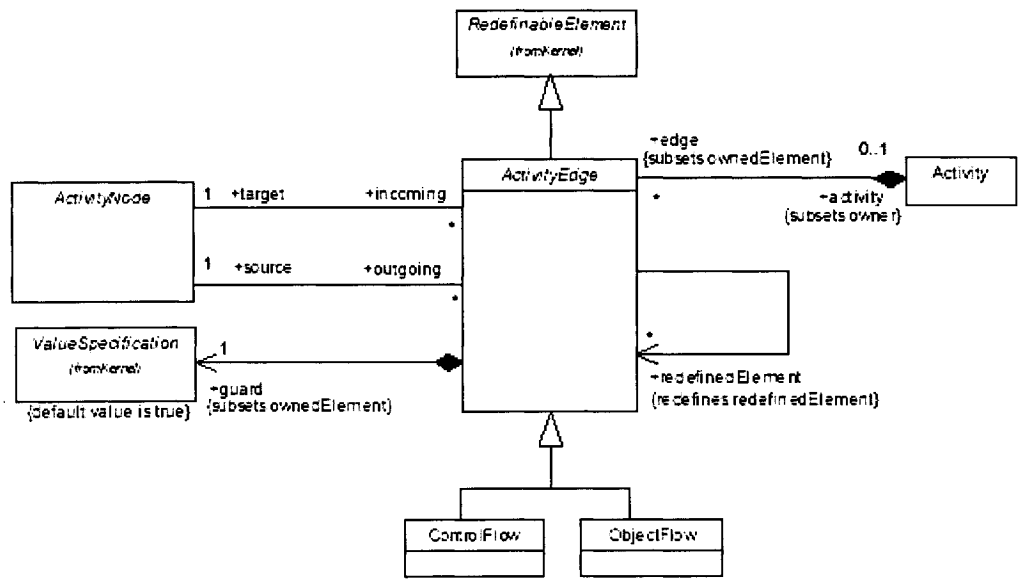



**Figure 2.13:** Class diagram for Activity Edges in Activity Diagram (taken from [26])

Important notations introduced in UML2.0 are

### 2.4.1 Action Nodes

An action as shown in the metamodel in Figure 2.12 is the only executable node in UML Activity Diagrams. It has been defined for invoking behaviors, either directly or through an operation. Actions are directly contained only in activities. Actions are notated with round-cornered rectangles. The names can be written inside the action.

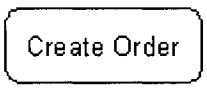



**Figure 2.14:** Action Notation

To begin executing, an action must know when to start and what its inputs are. These conditions are called control and data, respectively. Figure 2.15 shows control and data flow edges in UML2.0 directed towards an action. Data flow is distinguished from control by small rectangles on the action to show the type of data flowing into the action. These are called pins. An action may have sets of incoming and outgoing activity edges that specify control flow and data flow from and to other nodes. An action will not begin execution until all of its input conditions are satisfied. The completion of the execution of an action may enable the execution of a set of successor nodes and actions that take their inputs from the outputs of the action.

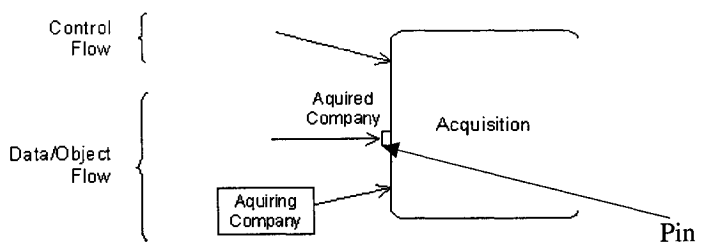



**Figure 2.15:** Control and data flow into an action (taken from [5])

An action that has control and data outputs is notated in the same way as inputs, except the flow arrows point in the other direction. An action terminates based on conditions

internal to itself, but when the action does terminate, data is posted to its output pins, and control values are placed on all its outgoing control flows. An action that has no control or data outputs can still terminate, but its termination cannot cause other actions to start [3].

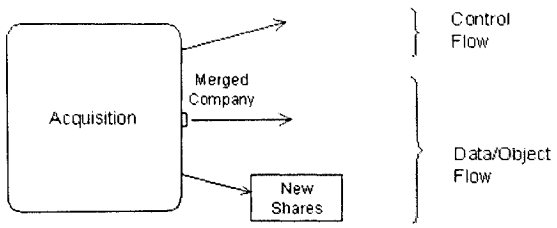



**Figure 2.16:** Control and Data Flow out of an Action (taken from [5])

### 2.4.2 Control Nodes

There are seven kinds of control node, with five notations, as shown in Figure 2.15. Contrary to the name, control nodes route both control and data/object flow.

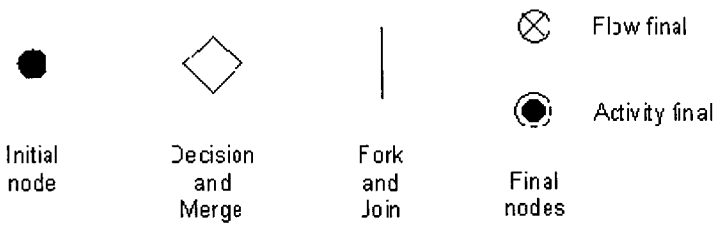



**Figure 2.17:** Control Nodes (taken from [6])

Each of these items is described briefly below.

#### 2.4.2.1 Initial Node

Initial nodes are used to show the start of an activity. They receive control when an activity is started and pass it immediately along their outgoing edges. No other behavior is associated with initial nodes in UML. Initial nodes cannot have edges coming into them. An activity can contain more than one initial node. If an initial node has more than one outgoing edge, only one of the edges will receive control, because initial nodes cannot copy tokens [26].

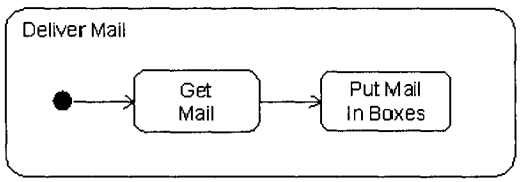



**Figure 2.18:** Initial Node (taken from [6])

#### 2.4.2.2 Decision Node

Decision nodes guide flow in one direction or another, but exactly which direction is determined by the constraints over the edges coming out of the node. Usually edges from decision nodes have guards, which are Boolean value specifications evaluated at runtime to determine if control and data can pass along the edge [4]. The guards are evaluated for each individual control and data token arriving at the decision node to determine exactly one edge the token will traverse. Decision node has one incoming edge and multiple outgoing edges.

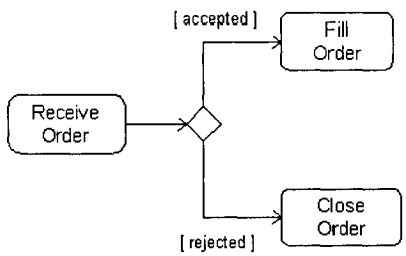



**Figure 2.19:** Decision Node (taken from [6])

#### 2.4.2.3 Merge Node

Merge nodes are used to bring together multiple flows. All control and data arriving at a merge node are immediately passed to the edge coming out of the merge. Merge nodes have the same notation as decision nodes, but merges have multiple edges coming in and one going out.

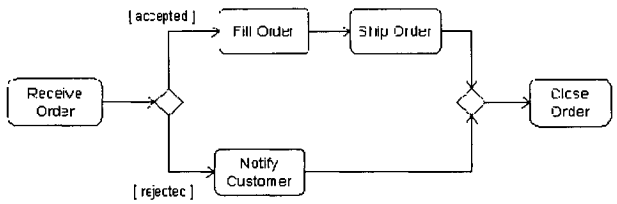



**Figure 2.20:** Merge Node with Alternate Flows (taken from [6])

#### 2.4.2.4 Fork Node

Fork node is used to split a flow into multiple concurrent flows. Data and control tokens arriving at a fork are duplicated and sent to the outgoing edges. There is no other behavior associated with fork nodes. In UML 2.0 it is not necessary to synchronize the behavior on concurrent flows which was required in UML 1.x activities. [4]. A fork node has one incoming edge and multiple outgoing edges.

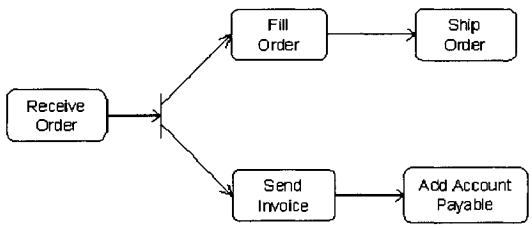



**Figure 2.21:** Fork Node (taken from [6])

#### 2.4.2.5 Join Node

Join node is used to synchronize multiple flows. Control or data must be available on every incoming edge so that it can be passed to the outgoing edge. Join nodes have the same notation as fork nodes, but joins have multiple edges coming in and one going out. Flows coming into a join are usually concurrent flows from an upstream fork. Join nodes take one token from each of the incoming edges and combine them according to these rules:

1. If all the incoming tokens are control, then these are combined into a single control token for the outgoing edge [6].

2. If some of the incoming tokens are control and others are data, then these are combined to provide only the data tokens to the outgoing edge. The control tokens are destroyed [6].

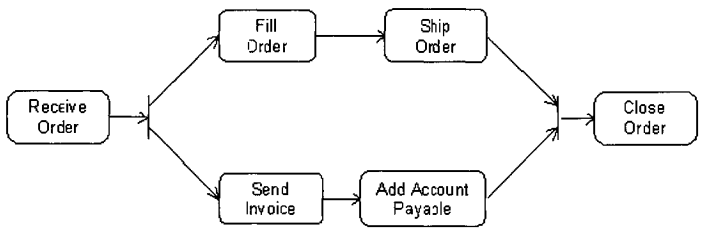



**Figure 2.22:** Join Node (taken from [6])

#### 2.4.2.6 Final Node

Flow in an activity ends at final nodes. There are two kinds of final nodes in UML2.0 activity diagrams

a) **Flow Final:** Each flow can have its own flow final. Final nodes cannot have outgoing edges so there is no downstream effect of tokens going into a node, which are simply destroyed. Activities terminate when all tokens in the graph are destroyed, so the activity shown in Figure 2.21 will terminate when both flows reach the flow final.

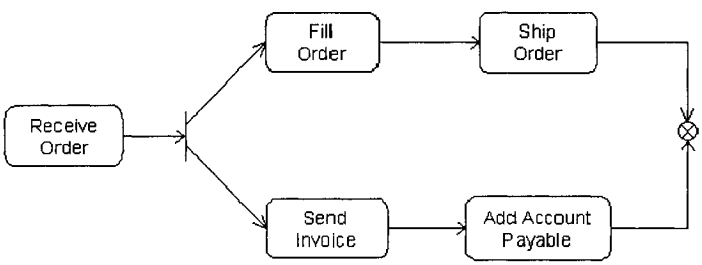



**Figure 2.23:** Flow Final Node (taken from [6])

**b) Activity Final**: nodes are like flow final nodes, except that control or data arriving at them immediately terminates the entire activity. This makes a difference if more than one control or data token might be flowing in the graph at the time the activity final is reached.

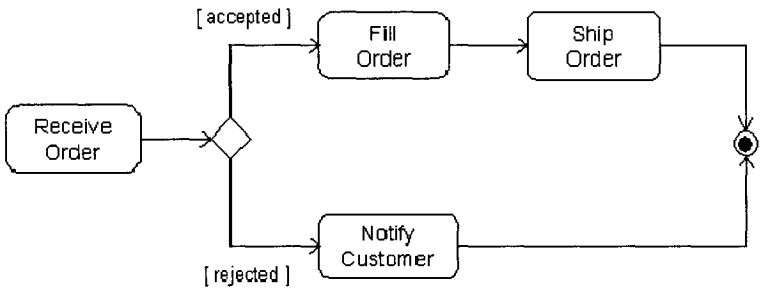



**Figure 2.24:** Activity Final Node (taken from [6])

There may be several final nodes in an activity.

### 2.4.3 Activity Edge

There are two kinds of activity edges in UML2.0

**a) Control Flow Edge**: A control flow is an activity edge that only passes control tokens. Tokens offered by the source node are all offered to the target node.

**b) Object Flow Edge:** An object flow is an activity edge that can have objects or data passing along it.

An activity edge is notated by a stick-arrowhead line connecting two activity nodes. If the edge has a name it is notated near the arrow [26].


**Figure 2.25:** Activity edge

### 2.4.4 Object Nodes

There are four kinds of object node in UML2.0 activity diagrams, as shown in Figure 2.26. All object nodes specify the type of value they can hold and if no type is specified, they can hold values of any type. Object nodes can hold more than one value at a time, and some of these values can be the same. Each object node specifies the maximum number of tokens it can hold, including any duplicate values, which is called the upper bound.

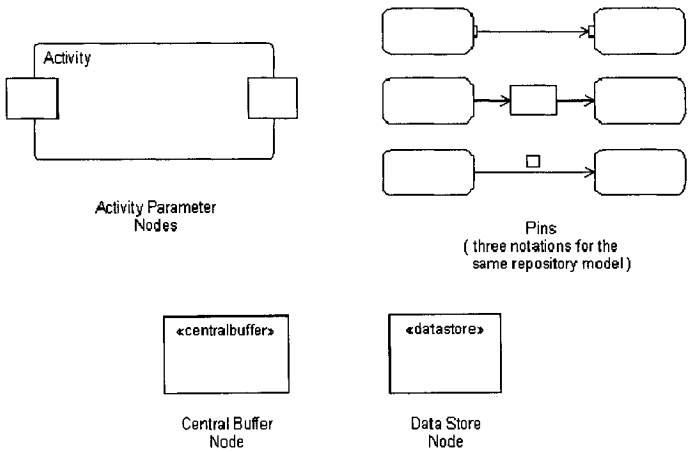



**Figure 2.26:** Object Nodes (taken from [7])

#### 2.4.4.1 Activity Parameter Node

Activity parameters are for accepting inputs to an activity and providing outputs from it. Activity parameter nodes must have either no incoming edges or no outgoing edges. When an activity in invoked, the inputs values are placed as tokens on the input activity parameter nodes. When the activity has finished the outputs of the activity must flow to output activity parameter nodes.

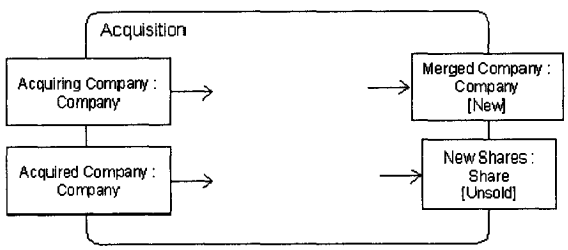



**Figure 2.27:** Activity Parameter Nodes (taken from [7])

### 2.4.4.2 Pins

Pins provide values to actions and accept result values from them and are connected as inputs and outputs to actions. An example of input pins is shown in Figure 2.28, in two of the notational forms. Whichever input value reaches a pin first is held there until the other arrives. When both pins have a value, the values are passed into the action and it starts.

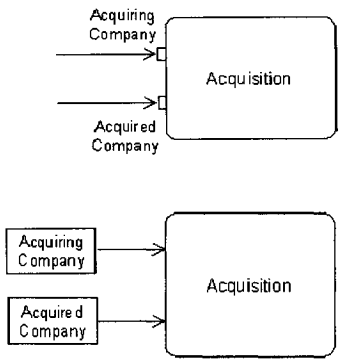



**Figure 2.28:** Input Pins (taken from [7])

A special kind of input pin called a value pin is defined for providing constant values such as numbers, or values.

Pins can be notated with the effect that their actions have on objects that move through the pin. Effect is one of the four values CREATE, READ, UPDATE, or DELETE.

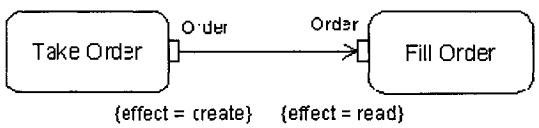



**Figure 2.29:** Pins showing effect of actions on objects (taken from [7])

A parameter node or pin may have multiple edges coming out of it, whereupon there will be competition for its tokens, because object nodes cannot duplicate tokens like forks. For example, Figure 2.30 shows parts being made, then painted at one of two stations, but not both.

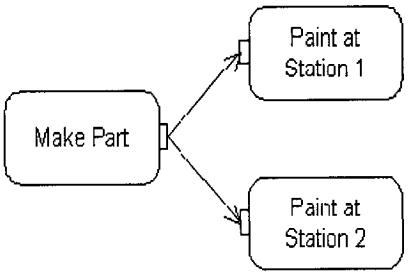



**Figure 2.30:** Token Competition (taken from [7])

#### 2.4.4.3 Central Buffer

Central buffers are for situations where tokens under competition arrive from multiple sources [5]. For example, Figure 2.31 shows parts arriving at a central buffer from two factories, which are then painted at two other factories. Pins cannot be used as central buffers, because pins have flows coming or going out, but not both.

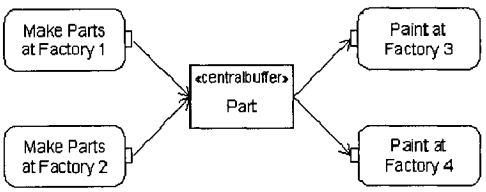



**Figure 2.31:** Central Buffer (taken from [7])

#### 2.4.4.4 Data Store

A data store node is for non-transient information. UML2.0 data store nodes are an attempt to support the earlier form of data flow and storage by providing a non-depleting specialization of object node. Tokens flowing out of data store nodes are copies of tokens that remain in the data store node. Tokens in a data store node cannot be removed, though values do not remain in the store after the containing activity is terminated, and a token arriving at a store that already has another token for the same object replaces that token.

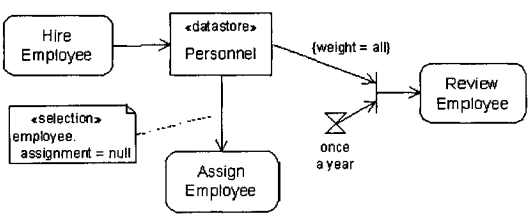



**Figure 2.32:** Data Store Node (taken from [7])

### 2.4.5 Partitions

Partitions are used to indicate what or who is responsible for actions grouped by the partition. The term responsible has a wide variety of a meaning, but the one defined by UML is that software class/component supports the behavior invoked by actions in the partition. Partitions often correspond to organizational units in a business model. An activity diagram may be divided visually into Partitions each separated from neighboring Partitions by vertical solid lines on both sides. Partitions do not have execution semantics. Partitions do not affect the token flow of the model. In UML2.0 the partitions can be hierarchical.

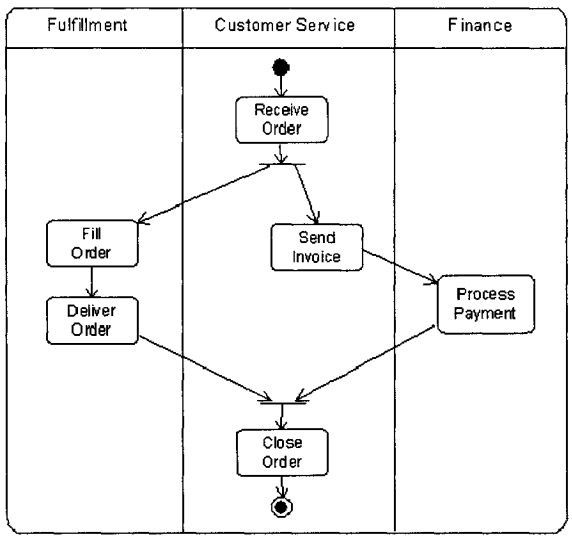



**Figure 2.33:** Partition Example, Swimlane Notation (taken from [8])

### 2.4.6 Usage of Activity Diagrams

Like any other diagrams in UML, activity diagrams have also strengths and weaknesses. The strength of activity diagrams lies on their ability to support parallel behavior, but their great disadvantage is that they do not make clear the links between actions and components, which are visualized better using interaction diagrams or state diagrams. However they are suitable mostly when we deal with the following situations:

#### 2.4.6.1 Modeling an operation

Activity diagrams can be used to model an operation associated with a use case or a class. In this case we are only interested in understanding what actions need to take place and what the behavioral dependencies are. Using an activity diagram to model an operation is simply like using it as a flowchart that supports concurrencies among threads in an operation.

#### 2.4.6.2 Modeling a workflow

Activity diagrams are best suited to model workflows across use cases that involve many actors or business organizations. In this case we focus on activities as seen by the actors that collaborate with the system.

## 2.5 Petri Nets

### 2.5.1 Definition

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 [16].

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. Figure 2.34 shows firing of an enabled transition. Transition t removes two token from place $H_2$ and one token from place $O_2$ and generate two $H_2O$ tokens,

$2H_2 + O_2 \rightarrow 2H_2O$

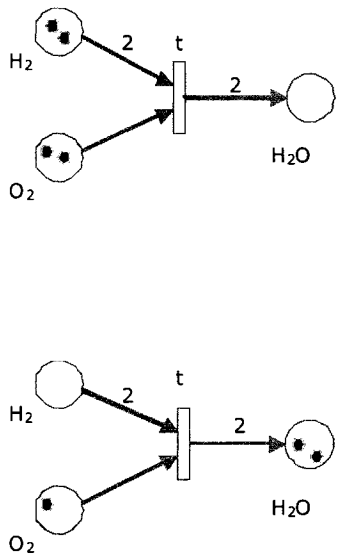



**Figure 2.34:** Firing example

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 [34].

### 2.5.2 Petri Net Metamodel

A simple Petri net is a set of places and transitions interconnected by directed arcs (an arc goes from a place to a transition or from a transition to a place). These concepts like Petri Net, Place, Transition and Arc have been used to define a Petri Net metamodel.

A Petri Net is composed of Transition, Place, input and output arcs. Places can be interface places or internal places of a Petri Net. The interface places are for Starting and Ending a Petri Net. If a Transition represents a complex operation and needs to be represented with more details, it can be represented as a Petri Net.

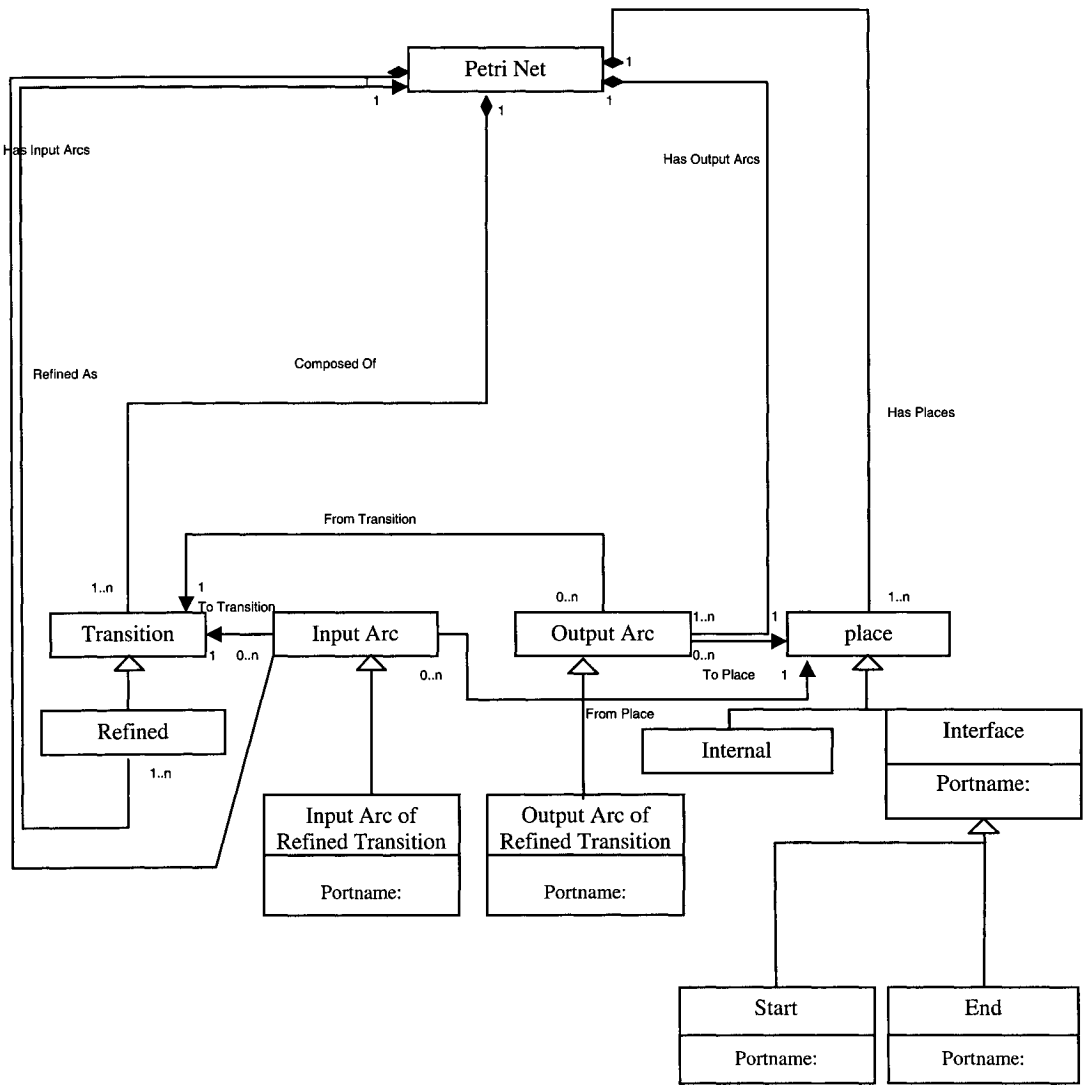



**Figure 2.35:** Petri Net metamodel

### 2.5.3 Colored Petri Nets

Colored Petri Nets (CPN) is a modeling language for design, specification, simulation and verification of systems. CPN combine the strengths of ordinary Petri nets with the strengths of a high-level programming language. Ordinary Petri nets provide the primitives for process interaction, while the programming language provides the primitives for the definition of data types and the manipulations of data values. Colored Petri Nets is a combination of Place/Transition Nets and Programming Languages.
In colored Petri Nets concurrency, control structures, synchronization, communication, and resource sharing between processes are described by means of Ordinary Petri Nets and data types and data manipulations are described by means of the functional programming language Standard ML.

CPN offer hierarchical descriptions, so large nets can be constructed by relating smaller CPN to each other, in a well-defined way. The hierarchy constructs of CPN play a role similar to that of subroutines, procedures and modules of programming languages. The existence of hierarchical CPN makes it possible to model large systems in a manageable and modular way.

A CPN model consists of a set of modules (pages) which each contain a network of places, transitions and arcs. The modules interact with each other through a set of well-defined interfaces, in a similar way as known from many modern programming languages. The graphical representation makes it easy to see the basic structure of a complex CPN model, i.e., understand how the individual processes interact with each other.

They also have a formal, mathematical representation with a well-defined syntax and semantics. This representation is the foundation for the definition of the different behavioral properties and the analysis methods.

Models with CPN can be made with or without explicit reference to time. Untimed CPN models are usually used to validate the functional/logical correctness of a system, while timed CPN models are used to evaluate the performance of the system. There are many

other languages which can be used to investigate the functional/logical correctness of a system or the performance of it. However, CPN is the only modeling language that is well-suited for both kinds of analysis.

Models can be simulated interactively or automatically with CPN. In an interactive simulation the user is in control. It is possible to see the effects of the individual steps directly on the graphical representation of the CPN. This means that the user can investigate the different states and choose between the enabled transitions. Automatic simulations are similar to program executions and the purpose is to be able to execute the CPN models as fast and efficient as possible, without detailed human interaction and inspection.

CPN also offers more formal verification methods, known as state space analysis and invariant analysis. In this way it is possible to prove, in the mathematical sense of the word, that a system has a certain set of behavioral properties. However, industrial systems are often so complex that it is impossible or at least very expensive to make a full proof of system correctness. The use of formal verification is often restricted to the most important subsystems or the most important aspects of a complex system.

Colored Petri Nets, as any description language with formal semantics, are used for three different but closely related purposes.

1) CPN model is a description of the modeled system, and it can be used as a specification (of a system to be built) or as a presentation (of a system to be explained to other people, or ourselves). By creating a model we can investigate a new system before we construct it. This is an obvious advantage, in particular for systems where design errors may jeopardize security or be expensive to correct.
2) The behavior of a CPN model can be analyzed, either by means of simulation (which is equivalent to program execution and program debugging) or by means of formal analysis methods (which are equivalent to program verification).
3) It should be understood that the process of creating the description and performing the analysis usually gives the modeler a dramatically improved

understanding of the modeled system - and it is often the case that this is more useful than the description and the analysis results themselves.

### 2.5.4 CPN Tools

A toolbox called "CPN Tools" has been developed at the Department of Computer Science, University of Aarhus, for editing, simulating, and analyzing Colored Petri Nets. Typical examples of application areas are communication protocols, distributed systems, automated production systems, work flow analysis and VLSI chips [19] [21].

"CPN Tools" is a major redesign of the "Design CPN" tool for editing, simulation and state space analysis of Colored Petri Nets. The new graphical interface is based on advanced interaction techniques, including bi-manual interaction, toolglasses and marking menus and a new metaphor for managing the workspace. The tool features incremental syntax checking and code generation which take place while a net is being constructed. A fast simulator efficiently handles both untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information such as boundedness and liveness properties. CPN Tools requires an OpenGL graphics accelerator and will run on all major platforms (Windows, Unix/Linux, and MacOS).

## 2.6 Work on Activity Diagrams into Petri Nets

As already described in Section 2.2, Activity Diagram is one of the behavioral diagrams in UML; which is a semi formal language for capturing different aspects and views of the system. Activity Diagrams are widely used in the software development process for modeling the dynamic aspects of the system.

Activity Diagram UML 2.0 is a complete rewrite of Activity Diagram UML 1.5, and many new concepts like data flow, exception handling, flow final, pins, datastore etc. have been introduced in the Activity Diagrams UML 2.0.

Many formal semantics for validation of activity models like Abstract State Machine semantics, Finite State Processes semantics and Petri Net semantics [35][20] have been

proposed. The Petri Net semantics defined by [35] [20] are based on UML 1.5 and cover aspects of control flow, concurrency and procedure call. The concepts of dataflow, object nodes and exception handling are not covered by [35] [20]. Also [35] and [20] discuss the transformations only at the conceptual level and propose manual translations from Activity Diagrams to Petri Nets.

The Object Management Group has defined an XML-based Metadata Interchange format (XMI) [28] which can be used as a model interchange format for UML. This allows UML modeling tools from different vendors to use XMI to exchange UML models. XMI can also be used to describe model transformations. For example if a UML model is to be transformed into a Petri Net model (which has a defined XML format); XMI can be used for this purpose. In this thesis we have introduced an algorithm for automatic transformation of Activity Diagrams to Petri Nets.

## 2.7 Extensible Markup Language, Document Type Definition, XML Schema, Document Object Model, XSLT

### 2.7.1 XML

XML (eXtensible Markup Language) is a simplified subset of SGML (Standard Generalized Markup Language) that maintains the features of validation, structure, and extensibility. Like SGML, XML is a metalanguage for describing the markup of different types of documents; however, XML's standardized text format was designed specifically for transmitting structured data to Web applications. But now it is also widely used for other applications. In addition, XML is easier to learn, use, and implement than full SGML. XML's development is guided by a working group of the W3C [36], and version 1.1 of the specification went to Recommendation status on 4th February 2004.

The XML specification describes XML documents, a class of data objects and partially describes the behavior of XML processor programs used to read such documents and provide access to their content and structure [33]. XML documents are composed of entities, which are storage units containing text and/or binary data. Text is composed of

character streams that form both the document character data and the document markup. Markups describe the document's storage layout and logical structure. A well-formed XML document is unambiguous, so that a browser or editor can read the tags and create a tree of the hierarchical structure without having to read its Document Type Definition.

XML provides a standard way for information providers to add custom markup to information-rich documents, so that complex documents can be rendered and published in a dynamic way.

W3C has recommended two specifications for defining the structure of XML documents

- Document Type Definition (DTDs)
- XML Schema Definition (XSD: discussed after DTD)

### 2.7.2 Document Type Definition

DTD is a set of rules about the structure of related XML documents. It defines the document structure with a list of legal elements. A DTD can be declared inline to XML document, or as an external reference. An XML document is said to be valid if it has an associated document type definition and if the document complies with the constraints expressed in it. A DTD can be used by independent groups of people who agree to use a common DTD for interchanging data. Applications can use a standard DTD to verify that data being received from the outside world is valid in respect to this DTD.

Let us consider the following example of a XML document with root element *note*. The *note* contains *to* for defining to whom this is being sent, *from* for defining who is sending it, *heading* for defining heading of note and *body* for defining body of note, with *body* having one paragraph.

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM
"note.dtd">
  <note>
      <to>Tove</to>
```

```
<from>Jani</from>
<heading>Reminder</heading>
<body>
<paragraph>Don't forget me this weekend.</paragraph>
</body>
</note>
```

The structure of such a note can be described by the following DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
 <!ELEMENT note (to,from,heading,body)>
 <!ELEMENT to     (#PCDATA)>
 <!ELEMENT from   (#PCDATA)>
 <!ELEMENT heading (#PCDATA)>
 <!ELEMENT body   (paragraph+)>
 <!ELEMENT paragraph   (#PCDATA)>
]>
```

Some elements contain other elements e.g. element *note* contains *to, from, heading* and *body* elements and body contains one or more *paragraph*. Other elements contain data in the form of text e.g. elements to, from, heading and paragraph contains #PCDATA (#PCDATA refers to parsed character data).

A DTD sets out a clear set of rules that allow documents to be validated. According to the above DTD, the document given as example, and any other that follows these rules, is a document of type known as Note.

The information provided by the DTD is called metadata and can be used to suggest how the document should be formatted for display or stored in a database. It also allows the data to be manipulated; a summary of the article could be produced by presenting only the note and heading elements. A more sophisticated DTD could allow the document to be easily searched or indexed. Because XML is a non-proprietary standard the

opportunity exists for varied applications to create and share documents without prior knowledge of their intended target.

### 2.7.3 XML Schema

Like a DTD, a XML Schema defines the structure of XML documents or data. A document which is compliant to the schema is called an instance document. A schema defines the way in which elements and attributes can be represented in a XML document. It also dictates that the given XML document should be of a specific format and specific data type. Major advantages of XML schema over DTD are the following:

- Unlike DTDs, XML Schema is an XML document so there is no real need to learn any new syntax.
- XML Schema supports Inheritance, where one schema can inherit from another schema. This is a great feature because it provides the opportunity for reusability.
- XML Schema provides the ability to define own data type from the existing data type.
- XML Schema provides the ability to specify data types for both elements and attributes.

The *note* example already discussed in the Section 2.7.2 can be represented in the XML schema format as shown below

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
>

<xsd:element name="note">
  <xsd:complexType>
    <xsd:sequence>
```

```
<xsd:element name="to" type="xsd:string"/>
<xsd:element name="from" type="xsd:string"/>
<xsd:element name="heading" type="xsd:string"/>

<xsd:element name="body" >
<xsd:complexType>
<xsd:sequence>
<xsd:element name="paragraph" type="xsd:string" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

**Explanation of important elements**

For an XML Schema, the root element is always <schema>. The XSD namespace declaration is provided with the <schema > to tell the XML parser that it is an XML Schema.

- xmlns:xsd="http://www.w3.org/2001/XMLSchema" indicates that the elements and data types used in the schema (schema, element, complexType, sequence, string, etc.) come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xsd:.
- targetNamespace="http://www.w3schools.com" indicates that the elements defined by this schema (note, to, from, heading, body, paragraph.) belong to the "http://www.w3schools.com" namespace.
- xmlns="http://www.w3schools.com" indicates that the default namespace is "http://www.w3schools.com".

An element declaration in XML Schema should have the name, type, minoccurs, maxoccurs attributes. An element can be of the following two types:

- Simple Type
- Complex Type

XML Schema provides a wide variety of base data types that can be used in a schema. Examples of base data types are xsd:string, xsd:int etc. An element is said to be of Simple Type if user defined data type is created from the base data types. If an element contains one or more child elements or if an element contains attributes, then the element is defined as Complex Type.

### 2.7.4 DOM

The W3C Document Object Model (DOM) is a platform and language-neutral interface that allow programs and scripts to dynamically access and update the content, structure and style of documents. A DOM document is a collection of nodes, or pieces of information, organized in a tree based hierarchy. This hierarchy allows a developer to navigate around the tree looking for specific information.

Analyzing the structure of a document requires the entire document and its hierarchy to be loaded into memory before any other processing is done. For exceptionally large documents, parsing and loading the entire document can be slow and resource intensive, so there are other means for dealing with such data. The event-based models, such as the Simple API for XML (SAX), work on a stream of data, processing it as it goes by. An event-based API eliminates the need to build the data tree in memory, but it does not allow a developer to actually change the data in the original document. The DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes at any point on the tree in order to create an application.

Working with the DOM involves several concepts that all fit together. A parser is a software application that is designed to analyze a document, in this case an XML file and

do something specific with the information. In DOM, the parser builds the data tree in memory. The complete working is shown in Figure 2.36.

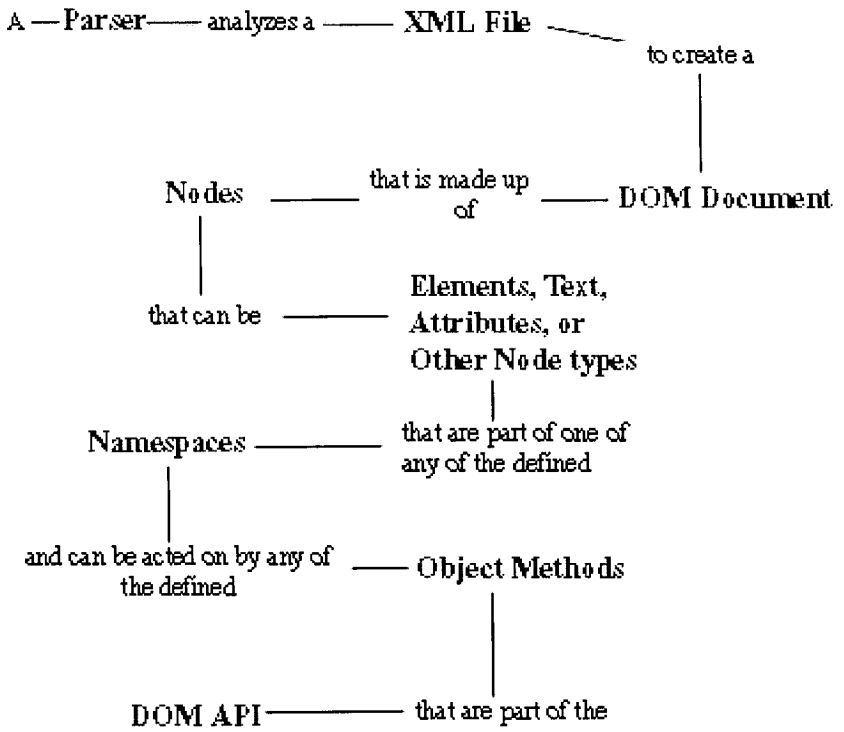



**Figure 2.36:** DOM Working [25]

**DOM Level 1 API:** The DOM Level 1 API was completed in 1998 and contains interfaces that represent all of the different types of information that can be found in an XML document, such as elements and text. It also includes the methods and properties necessary to work with these objects. Level 1 included support for XML 1.0 and HTML, with each HTML element represented as an interface. It included methods for adding, editing, moving, and reading the information contained in nodes.

**DOM Level 2 API:** Namespace support was added to the DOM Level 2. Level 2 extends Level 1, allowing developers to detect and use the namespace information that

might be applicable for a node. Level 2 also adds several new modules supporting Cascading Style Sheets, events, and enhanced tree manipulations.

**DOM Level 3 API:** It is the latest version of W3C Recommendations and adds enhanced namespace support, including two new recommendations, XML Infoset and XML Base; extended support for user interface events and support for DTD; XML Schema loading and saving abilities; and other features. Notably, it also adds support for XPath, the means used in XSL Transformations to locate specific nodes.

A DOM document is a tree structure, where each node represents one of the components (elements, their text, and their attributes are all known as nodes) from an XML document. All elements, their containing text and their attributes, can be accessed and manipulated through the DOM tree. However, the DOM standard is silent on the subject of how to create a DOM from an existing XML file. This problem is solved by many vendors and open source communities like Sun, Apache etc. We decided to use the API implemented by Sun which is a set of APIs to access and manipulate nodes in the DOM tree. It uses DocumentBuilderFactory and DocumentBuilder classes to parse an XML document and create a DOM tree from it. The graphical view is shown in the figure below.

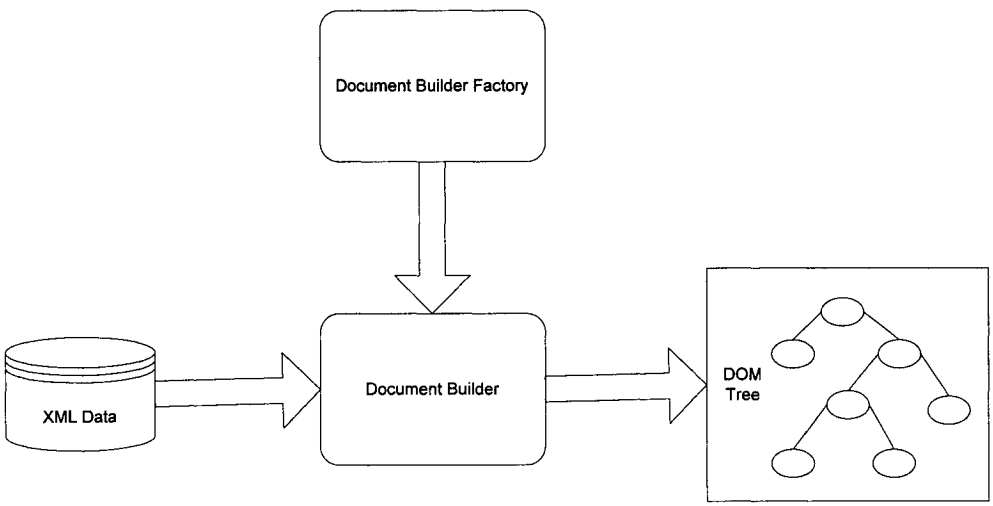



**Figure 2.37:** High level view of creating a DOM tree from XML documents

When the input file is parsed, the DocumentBuilder will return an org.w3c.dom.Document object.

```
DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document document = builder.parse(input file);
```

### 2.7.5 Extensible Stylesheet Language

XSL Transformations (XSLT) is a language for describing a scheme for converting XML documents into another format. An XSLT processor applies the rules from the XSLT style sheet to the XML document to create a new, third document in XHTML, WML, SVG, or almost any other XML format. Multiple XSLT files can translate a single XML document into multiple formats.

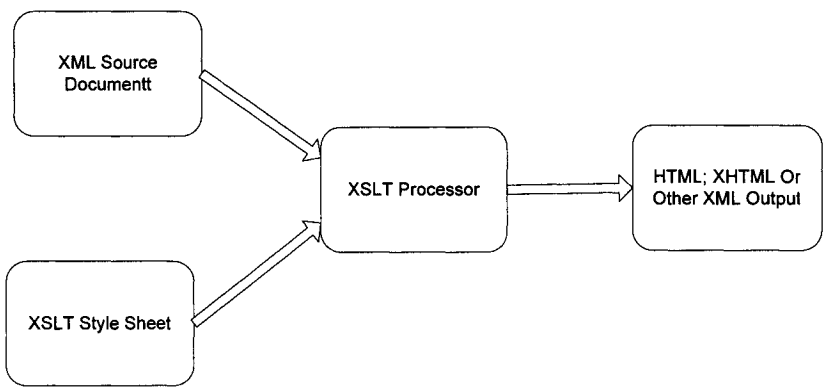



**Figure 2.38:** Transformation of XML document using XSLT

As a language, XSLT is best at manipulating the structure of the information as distinct from its content (data), e.g. string handling is much more laborious as compared with other programming language like Java. Also, for data conversion applications, XSLT processors hold all the data in memory while the transformation is taking place. The tree structure in memory can be as large as ten times the original data size, so a complex conversion can be quite time consuming.

Although XSLT can be used for manipulating complex data but it was designed mainly for the purpose of publishing. The most commonly used publishing method used today is World Wide Web and converting XML to HTML is the most common application for XSLT. Many XSL processors are available for use, but the most popular are Saxon and Xalan.

### 2.7.6 The Xerces XML Parser

Xerces is an open source XML parser developed by Apache. It can be used for parsing, validating and processing XML document using the Simple API for XML (SAX) or the Document Object Model (DOM) API. It is available in Java and C++ and is fully XML Schema-compliant since Xerces version 1.1.3.

Apache has developed two different parsers: Xerces Java 1 and Xerces Java 2. Xerces2 is a complete rewrite of the existing version 1. Xerces2 has a custom Xerces Native Interface (XNI), and its source code is said to be much cleaner, more modular, and easier to maintain than Xerces1. Xerces2 also implements the latest W3C XML Schema standards.

## 2.8 Chapter Summary

This chapter covers the background information that will be used throughout this thesis. The transformation from UML Activity Diagram notation and CSM to Petri Net, involves many popular specifications, notations, and languages.

Section 2.1 recalls the basic elements of the UCM notation. Section 2.2 introduces different UML diagrams and UML metamodel. Section 2.3 throws some light on the Model Driven Architecture and its advantages. Section 2.4 is an introduction to UML Activity Diagrams and various notations used in it. Section 2.5 discusses Petri Nets and its advantages. Section 2.6 discusses the work already done on translation of Activity Diagrams to Petri Nets. Section 2.7 provides a brief introduction to the Extensible Markup Language (XML), Document Type Definition (DTD), XML Schema, Extensible Style Sheet Language (XSLT) and the Document Object Model (DOM). The knowledge

about UCM, Activity Diagrams and Petri Nets, provides us the foundation to build CSM meta-model.

With the above background information, we are ready to compare different design notations, extract common properties, and finally explain the meta-model for the Core Scenario Model (CSM).

# Chapter 3 Core Scenario Model (CSM)

The Core Scenario Model (CSM) developed under the projects "Requirement-Driven Development of Distributed Applications" (RDA) and "Performance by Unified Model Analysis" (PUMA) will play an important part in deriving various functional and performance models from different design models. This chapter presents the motivations behind the CSM model, the definition of the CSM, and the benefits of using the CSM model in the transformation processes.

## 3.1 Overview

The use of formal methods in achieving correct software is absolutely necessary. With the use of formal methods it can be proven that the software fulfil its requirements. Formal specifications are unambiguous and analysable. However, formal methods are not widely used in software development. In most cases, this is because they are not suitably supported with development tools. Further, many software developers do not recognise the need for rigour. Different formal languages like Petri Nets and Layered Queuing Networks can be used for validating the software requirements.

Different methodologies for transforming software requirement to formal languages have been proposed. Williams and Smith [13] proposed a transformation approach from UML class diagrams, deployment diagrams, and sequence diagrams to Queuing Networks to describe the software architecture and software behaviour. Petriu et al. [15] proposed three pattern-based methodologies. They also specified the software architecture by using UML class, deployment, sequence, activity, and use case diagrams.

The approaches mentioned above use different combinations of UML diagrams and the UCM notation to describe software architectures and behaviours. Each approach develops its own transformation tool to automate the transformation process. When users use a specific transformation tool, theirs choices are limited to only one required specific design specification and one specific functional/performance language. On the other hand, the drawback of having many different transformation tools is the inconvenience

and uncertainty in choosing between different approaches. To use such a transformation tool, the software developer has to know in advance what is the target performance model, what is the input requirement, and so forth. Each transformation tool has its own input/output limitations and requirements. When using another transformation tool, software designers may have to change their software design models to meet the different input requirements. This obviously delays the transformation process and functional and performance evaluations.

At the same time, we can also see that all transformation processes have many common properties. The transformation tools are based on the availability of suitable abstraction of the final software system. As already discussed in Chapter 1, scenarios are generally used to describe software behaviours at the requirements level. Almost all the approaches mentioned above start from UML activity diagrams, UML sequence diagrams, or Use Case Maps. All these three design notations have common concepts. This provides us with a possibility to consolidate all the different transformation processes and get a unified approach.

Recently, Woodside *et al.* proposed an approach [14] to integrate early functional and performance analysis with software development life cycle. The main concept is a Core Scenario Model, which acts like a standard interface between different software specifications and different functional and performance models.

The major benefit of using the CSM is the separation of concern. The CSM acts as a unified intermediate interface for functional and performance models. It provides an entry point to different types of functional and performance models. Having an intermediate CSM representation can reduce the number of transformation tools needed. Suppose there are N1 different kinds of scenario design models to be transformed into N2 kinds of functional/performance models one by one, then we need N1 * N2 transformation tools to be developed. On the other hand, if we make use of the CSM, then we need N1 transformations between scenario design models and CSM, and N2 transformations

between CSM and functional or performance models. The total number of transformation tools needed becomes N1+N2.

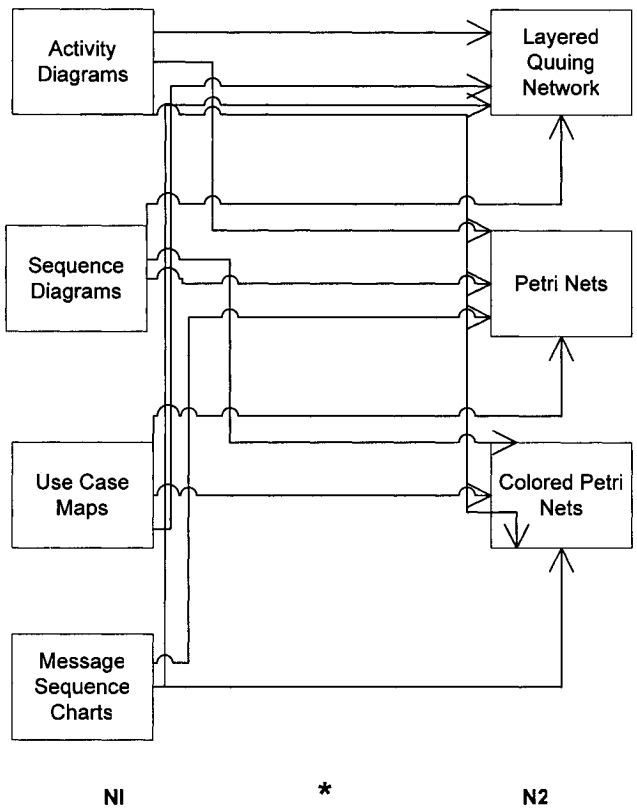



**Figure 3.1:** Transformations without Core Scenario Model

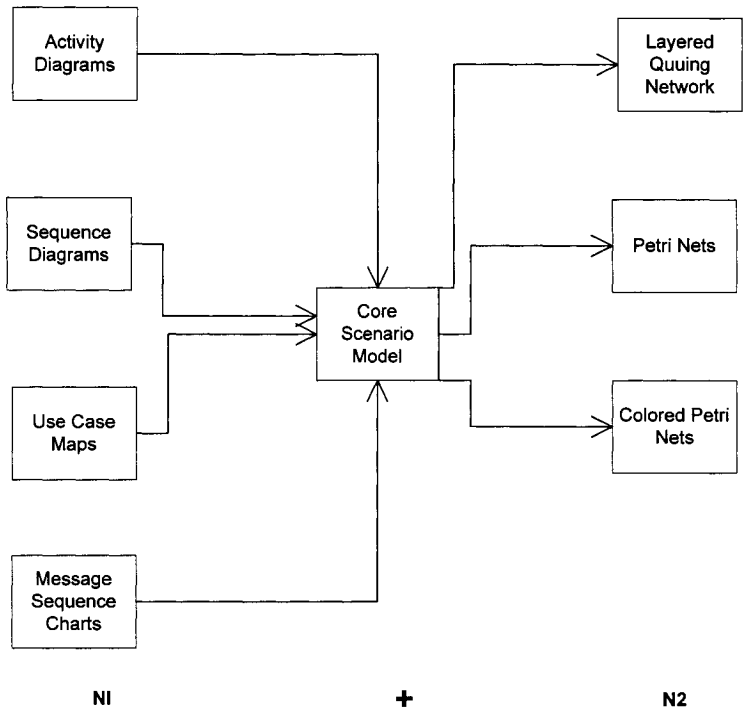



**Figure 3.2:** Transformations with Core Scenario Model

## 3.2 Comparison of UCM and UML Activity Diagrams

In order to provide a unified interface to different formal modeling languages, CSM should encompass the common properties among UML and UCM in order to allow different combinations of design specifications as inputs. Therefore, we first need to find out what are the common properties among different design notations. In the table below we compare the common properties among UCMs and UML activity diagrams.

| Concept | UCM | Activity Diagram | CSM |
|---|---|---|---|
| 1. Scenario representation | **Path (scenario):** A possible sequence of execution of actions; path chains responsibilities in a cause-effect sequence. | **Activity:** The flow of execution is modeled as activity nodes connected by activity edges | **Scenario** |
| 2. Entities | **Component:** | **Swimlane / Partition:** | **Component** |

| Concept | UCM | Activity Diagram | CSM |
|---|---|---|---|
| | Components indicate who is responsible for responsibilities. | Partitions indicate who is responsible for actions. | |
| 3.Function and action performed | **Responsibility:** Responsibilities are processing tasks (e.g. procedures, functions, actions, etc.) that are referenced by scenarios and by components | **Action:** An action is the fundamental unit of executable functionality, and an action represents a single step within an activity; activity is a grouping of actions, and an activity represents a behaviour which is composed of actions. | **Step** |
| 4. Scenario start and stop | **Start point & end point:** Indicate the start and end of a UCM. | **Initial Node & Final Node (Flow Final):** Indicate the start and end of flow. | **Start & End** |
| 5. Sub-scenario | **Stub:** Container for plug-in maps | **Sub activity:** Call behaviour action references an activity definition, in which case the execution of call action involves the execution of referenced activity and its actions. | **Refinement** |
| 6. Concurrency flow | **And Fork:** An AND fork splits a path into two or more concurrent segments | **Fork Node:** A fork node splits a flow into multiple concurrent flows. | **Fork** |

| Concept | UCM | Activity Diagram | CSM |
|---|---|---|---|
| 7. Alternative flow | **Or Fork:** An OR fork splits a path into two (or more) alternatives. | **Decision Node:** A decision node has one incoming edge and multiple outgoing activity edges, and token can transverse only one outgoing edge. | **Branch** |
| 8. Sequence flow | **Causal path sequence:** | **ActivityEdge:** There are two kinds of activity edges. 1) Control flow to model the sequencing of behaviours that does not involve the flow of objects. 2) Object flow to model the flow of data and objects in an activity | **Sequence** |
| 9.Alternative merge | **Or Join:** An OR join merges two or more overlapping paths | **Merge Node:** A merge node brings together multiple alternate flows. | **Merge** |
| 10.Synchronizing concurrent flow | **And Join:** An AND join synchronizes two or more paths together. | **Join Node:** A join node synchronizes multiple flows, and has multiple incoming edges and one outgoing edge. | **Join** |

**TABLE 3.1** Concept comparison of UCM and UML Activity Diagrams

The properties discussed above were used to determine what concepts are necessary to be supported by CSM. The concepts shown for Activity Diagrams and UCM's have been taken from UML Activity Diagrams and UCM metamodels [26] [37]. In the last column of the table 3.1 we have also shown the name of the corresponding concepts defined for CSM metamodel. Properties of UML Sequence Diagrams have also been kept in mind while designing CSM metamodel and Toqueer Israr from Carleton University worked on that part [32]. Detailed description of the CSM metamodel is given in Section 3.3.

## 3.3 Definition of the CSM

A metamodel for CSM has been designed by a team of people from Carleton University and University of Ottawa. The CSM metamodel captures the essential entities in the domain model which are required for building functional and performance models, and it makes explicit some facts which have to be inferred from UCM, UML and Schedulability, Performance, and Time (SPT) Profile data [14].

The CSM metamodel development went through many phases. Initially CSM was designed keeping in view only the performance aspects of the system and it was decided that the metamodel should consist of only two packages namely CSMMETA and XSD_Datatypes. CSMMETA was used for defining different classes for constructing a CSM and XSD_Datatypes to be used for defining various data types. Figure 4.3 and 4.4 show the details of the initially proposed metamodel [24].

The proposed CSM provides explicit representation of the scenario flow in a PathConnection type [14]. The Profile depends on a simple successor association between Steps (Actions in UML2.0). Here, there is a PathConnection object between each pair of Steps, with subtypes which correspond to the sequential relationship types common in path models for real-time software: *Sequence* for simple sequence, one step after another; *Branch* for an OR-fork with *Merge* for an OR-join, to describe alternative paths, and *Fork* and *Join* for describing parallel paths.

Each PathConnection subtype takes a different number of source steps, and successor steps, and these are labeled with the subtypes in the diagram. For example, a Sequence has exactly one source and one target Step, while a Fork or Branch has one source Step and multiple target Steps.

Active and passive resources represent the resources defined in the SPT Profile [27]. Active resources include devices (Processing Resources) and subsystems (captured by a placeholder called External Service). The latter are service operations executed by some resource outside the design document. Passive resources include operating system processes (or threads) identified as Components, and hosted by Processing Resources.

In the initially proposed metamodel the Start and End of a Scenario were proposed to be Steps which in our opinion was not inline with the UML Semantics, where Start (Initial Node in Activity Diagrams) and End (Final Node in Activity Diagrams) points are simply used for forwarding or destroying tokens and are described as Control nodes. So we proposed that the Start and End points should be PathConnections in the CSM metamodel. This proposal was discussed in the group meeting and was approved.


**Figure 3.3:** Package diagram

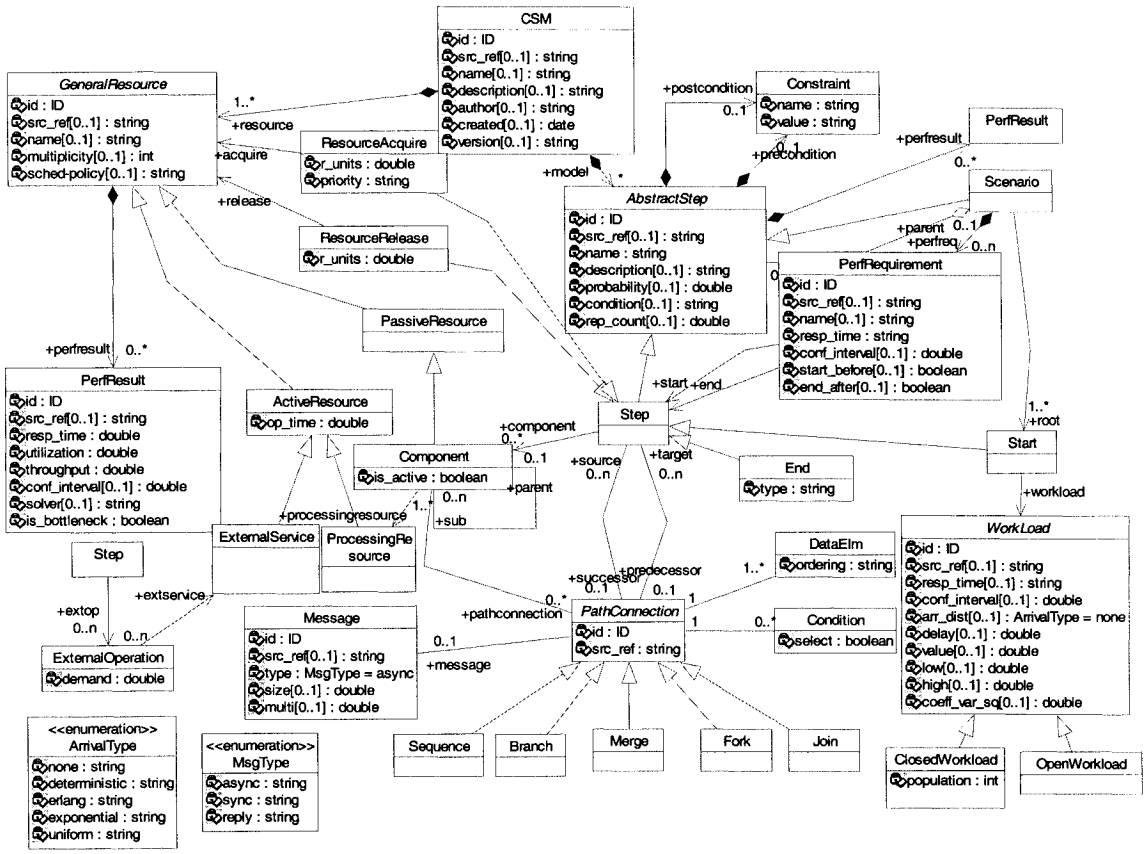



**Figure 3.4:** Initial CSMMETA Class diagram

Also the old version did not support completely the refinement of a Step (or action in activity diagrams) which is an integral part of UML models for representing complex scenarios. Although in the initial metamodel it was shown that a step can be represented as a Scenario but it did not take care of how the inputs to a Step needing refinement and Outputs generated by that Step will be mapped onto the Scenario. We proposed that refinement should be added to CSM and after team discussion it was also approved.

Although it was possible to construct purely functional models from this version, one had to go through all the classes to understand which are required for building the functional model. After long discussions it was decided that the CSM metamodel should be defined in such a manner that it will be easy to construct both a functional and a performance

models of a system. Based on our proposals the CSM metamodel was revised and is shown in Figures 3.5, 3.6, 3.7 and 3.8. It has been proposed that the CSM should consist of four packages and each package would lead to more detailed diagram. Figure 3.5 also shows the dependencies of the packages. The XSD_Datatypes package defines the primitive types, such as integer, double, string etc. It is a standard package, used for the schema validation purpose.

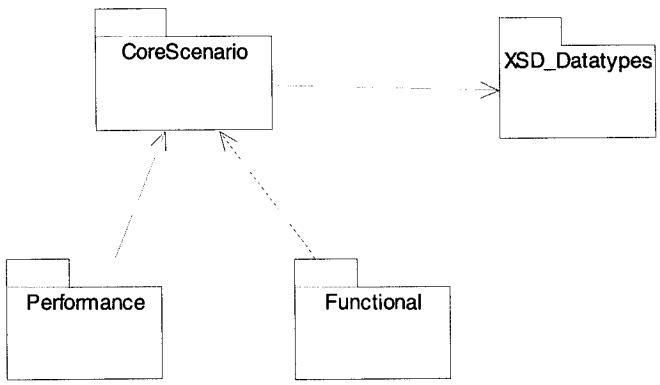



**Figure 3.5:** Package diagram for Core Scenario Model metamodel.

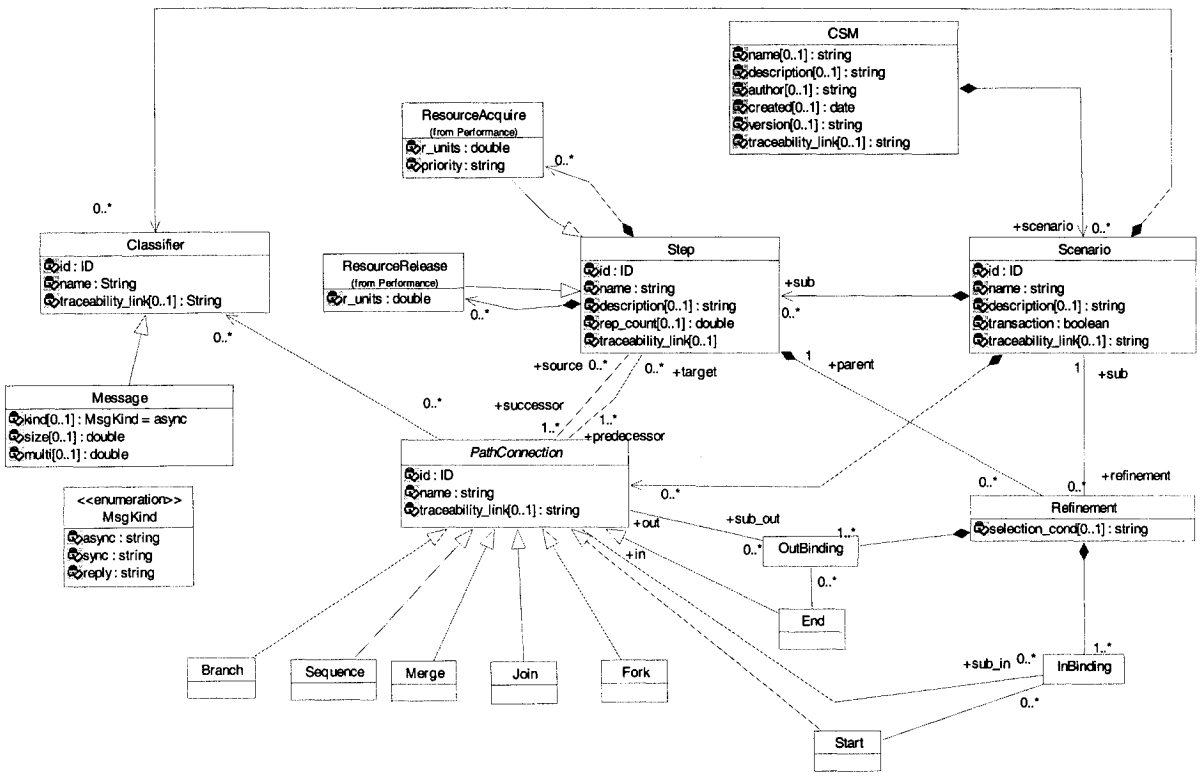



**Figure 3.6:** Class diagram for core package CSM metamodel.

Figure 3.6 shows the Classes which became part of the Core Package. The CoreScenario package is the most important package in the CSM; it defines the basic elements used to construct CSM models. Refinement class has been proposed in the Core Package for refining the Steps that are represented as independent scenarios. Two more classes, Inbinding and Outbinding, are defined for the inputs to such steps and outputs generated by them. Detailed information about each class in this package can be found in Section 3.3.1.

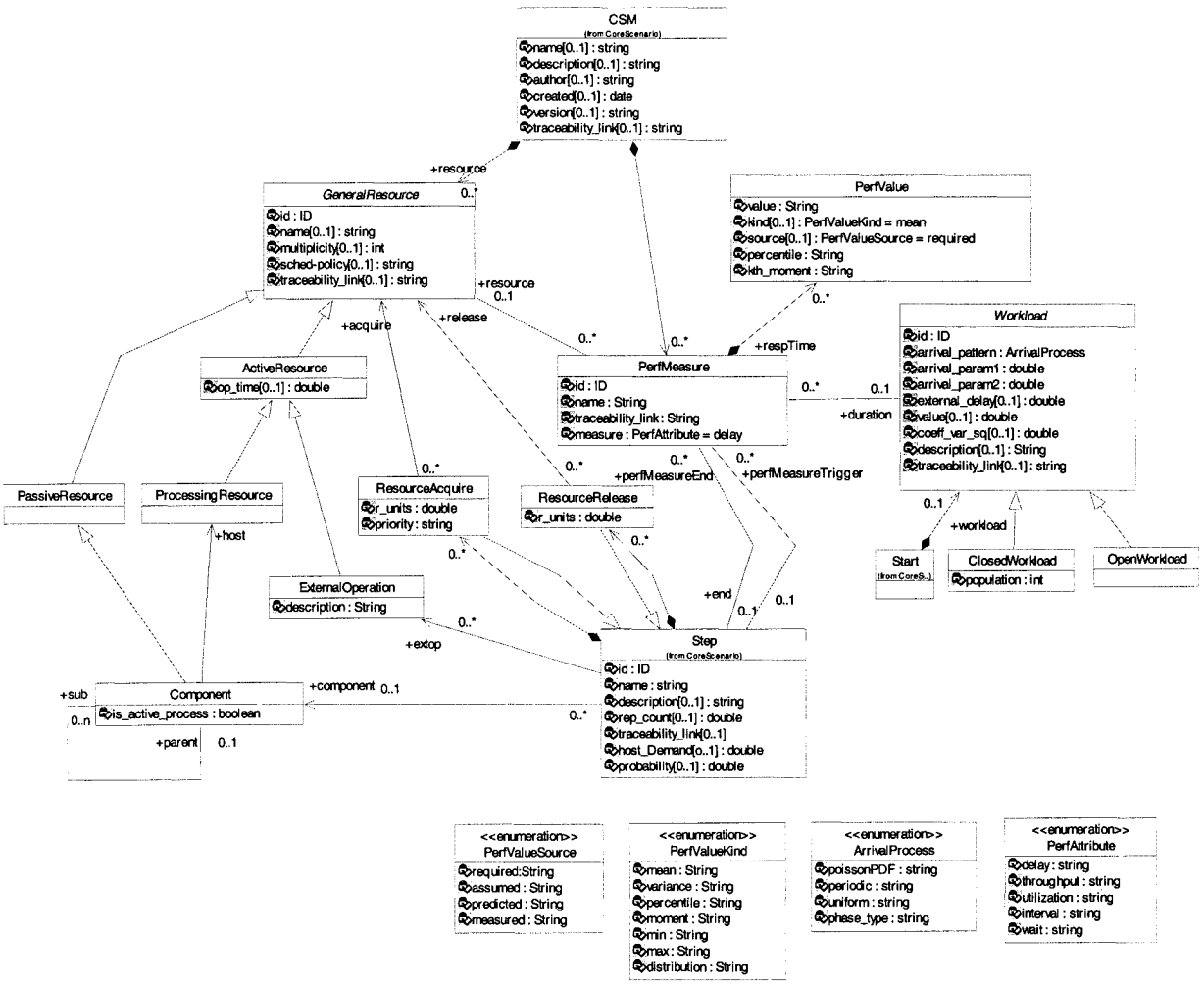



**Figure 3.7:** Class diagram for performance package of CSM metamodel.

The Performance package collects all the elements related to performance information, such as resource, performance requirement, performance result etc.

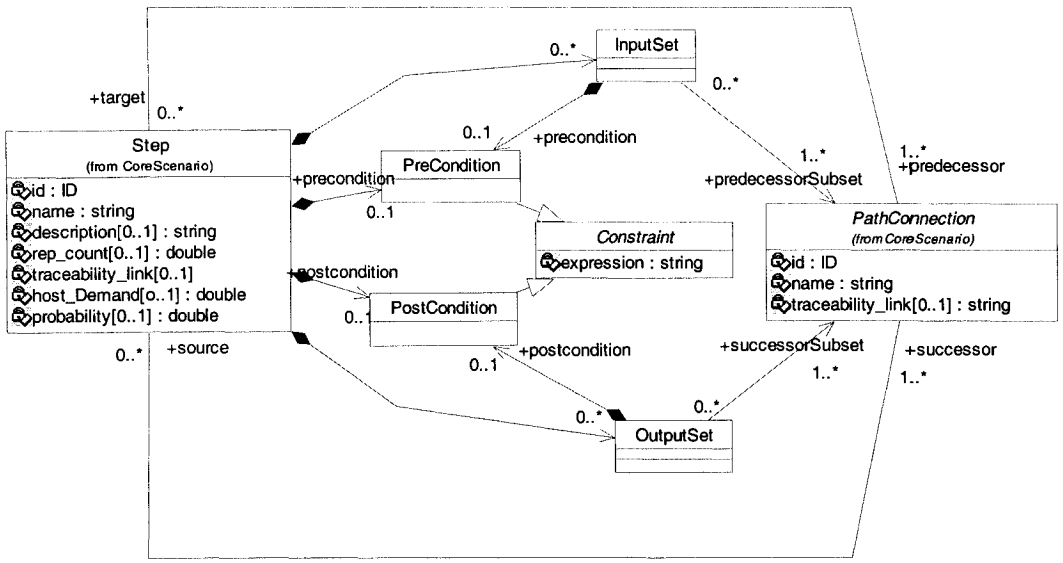



**Figure 3.8:** Class diagram for functional package of CSM metamodel.

Figure 3.8 shows the Function package of CSM. The Function package adds information about constraint and object flow. PreConditon and PostCondition classes have been proposed for conditions which must be met, before a Step can start executing and after a Step has executed, respectively. Input Set and Output Set classes have been proposed for one or multiple input and output sets of a Step. These two classes have been proposed to support alternative sets of inputs and outputs of a behavior, as introduced in UML 2.0 Activity Diagrams under the name ParameterSet. Detailed information about each class in the function package can be found in the following section.

### 3.3.1 Class Description of CSM Core Package

Class description for the core, functional and some important classes of performance packages is provided in the following sub sections.

#### 3.3.1.1 CSM

**Description:** A Core Scenario Model (CSM) is the top-level element in a scenario, which contains the core concepts from Use Case Maps (UCM) and UML 2.0 activity diagram and sequence diagram. It is composed of scenario model(s), and a collection of components, steps, path connections, resources etc.

**Attributes**

| | |
|---|---|
| name: string [0..1] | the model name |
| description: string [0..1] | detail information of the model |
| author: string [0..1] | the model author |
| created: date [0..1] | the date of creation |
| version: string [0..1] | version number |
| traceability_link: string [0..1] | for describing connections between models |

**Associations**

| | |
|---|---|
| scenario: Scenario [0..*] | scenario contained in the model |

**Constraints**

None

#### 3.3.1.2 Scenario

**Description:** Scenarios are the basic elements of a CSM model. Scenarios are used to define a behavior by specifying the elements that make up the scenario and the connections between elements.

**Attributes**

| | |
|---|---|
| id: ID | the identity of the scenario |
| name: string | the name of the scenario |
| description: string [0..1] | the detail information of the scenario |
| transaction: boolean [0..1] | indicates if scenario acts like a transaction |
| traceability_link: string [0..1] | links to the original model |

**Associations**

| | |
|---|---|
| sub: Step [0..*] | a collection of step(s) owned by a parent scenario |
| path: PathConnection [0..*] | a collection of path connection(s) owned by parent scenario |
| classifier: Classifier [0..*] | describes dataflow and is a set of instances that have features in common |
| refinement: Refinement [0..*] | a collect ion of steps, which need refinement |

**Constraints**

A scenario may not directly or indirectly own itself.

#### 3.3.1.3 Step

**Description:** A step is a fundamental unit of executable functionality. It describes the processing tasks (e.g. procedures, functions actions, etc.) that a system perform.

**Attributes**

| | |
|---|---|
| id: ID | the identity of a step |
| name: string | the name of a step |
| description: string [0..1] | detailed information on a step |
| traceability_link: string [0..1] | links to the original model |
| rep_count: double [0..1] | the count of repetitions |

**Associations**

| | |
|---|---|
| predecessor: PathConnection [*] | the predecessor(s) of current step |
| successor: PathConnection [*] | the successor(s) of current step |
| refinement: Refinement [0..*] | a collection of binding connections to refine current step |
| precondition: PreCondition [0..1] | constraint that must hold when the execution starts |
| postcondition: PostCondition [0..1] | constraint that must hold when the execution completes |
| inputset: InputSet [0..*] | alternative sets of input values to the execution |
| outputset: OutputSet [0..*] | alternative sets of result values from the execution |
| Component: Component [0..1] | the component that contains current step |
| Resourceacqurie: ResourceAcqurie [0..1] | the resource acquire action |
| Resourcerelease: ResourceRelease [0..1] | the resource release action |

**Constraints**

None

#### 3.3.1.4 Refinement

**Description:** In order to show the complex behavior a step can make use for the refinement class. A Refinement defines a connection between a single step and its referenced scenario(s). The execution of a single step can involve the execution of the

complex scenario and its steps. The same behaviour is captured as a call behavior action in UML 2.0 activity diagram or stub in Use Case Maps (UCM).

**Attributes**

| | |
|---|---|
| selection_cond: string [0..1] | A boolean condition for selecting scenario |

**Associations**

| | |
|---|---|
| parent: Step [1] | the invoking step |
| sub: Scenario [1] | the referenced scenario |
| inbinding: InBinding [*] | the inputs to the step needing refinement |
| Outbinding: OutBding [*] | the outputs from the step needing refinement |

**Constraints**

None

#### 3.3.1.5 InBinding

**Description:** An InBinding joins an entry point of a step which needs refinement as a scenario.

**Attributes**

| | |
|---|---|
| id: ID | the identity of an inbinding element |

**Associations**

| | |
|---|---|
| in: PathConnection [1] | the entry point of invocation action |
| start: Start [1] | the start point of the referenced scenario |

**Constraints**

None

#### 3.3.1.6 OutBinding

**Description:** An OutBinding joins an exit point of a step with an end point from a scenario.

**Attributes**

| | |
|---|---|
| id: ID | the identity of an outbinding element |

**Associations**

| | |
|---|---|
| out: PathConnection [1] | the exit point of invocation action |

| | |
|---|---|
| end: End [1] | the end point of the referenced scenario |

**Constraints**

None

#### 3.3.1.7 PathConnection

**Description:** A PathConnection element is an abstract class for the connections between two different elements of a scenario.

**Attributes**

| | |
|---|---|
| id: ID | the identity of path connection element |
| description: String [0..1] | detail information of path connection element |
| traceability_link [0..1] | the link to original model |

**Associations**

| | |
|---|---|
| source: Step [0..*] | step element from which the path connection starts |
| target: Step [0..*] | step element at which the path connection ends |
| classifier: Classfier [0..*] | classifiers can be attached to the path flow |
| subin: InBinding [0..*] | the entry point to sub-scenario |
| subout: OutBinding [0..*] | the exit point from sub-scenario |
| subset : InputSet[0..*] | alternative sets of inputs |
| subset: OutputSet[0..*] | alternative sets of outputs |

**Constraints**

(1) The source and target of a path connection element must be in the same scenario.

#### 3.3.1.8 Start

**Description:** A Start element is where scenario is invoked. A scenario may have more than one start element.

**Attributes**

Attributes are same as of PathConnection as Start is a PathConnection.

**Associations**

| | |
|---|---|
| inbinding: InBinding [0..*] | the input points of a sub scenario |
| workload: Workload [1] | the stream of requests attached to the scenario |

**Constraints**

(1) A start element has no incoming edges.

(2) A start element must have only one target edge.

#### 3.3.1.9 End

**Description:** An End element is where scenario is stopped. A scenario may have more than one end element.

**Attributes**

Attributes are same as of PathConnection as End is a PathConnection.

**Associations**

outbinding: OutBinding [0..*] the outputs from a sub scenario

**Constraints**

(1) An end element has no target edges.

(2) An end element must have only one source edge.

#### 3.3.1.10 Sequence

**Description:** Sequence elements specify the sequencing of steps.

**Attributes**

Attributes are same as of PathConnection as Sequence is a PathConnection.

**Associations**

None

**Constraints**

(1) The source and target of a sequence element must be in the same scenario.

#### 3.3.1.11 Branch

**Description:** A Branch represents alternative paths from a single source. The conditions can be attached to branch elements so that a choice is determined when the condition is true.

**Attributes**

Attributes are same as of PathConnection as Branch is a PathConnection.

**Associations**

None

**Constraints**

The branch element must have only one source and two or more targets.

#### 3.3.1.12 Merge

**Description:** A Merge represents the merging of two or more independent alternative paths. It is not used to synchronize concurrent paths but to accept one among several alternative paths.

**Attributes**

Attributes are same as of PathConnection as Merge is a PathConnection.

**Associations**

None

**Constraints**

The merge element must have two or more sources and only one target.

#### 3.3.1.13 Fork

**Description:** A Fork is an element that splits a path into two or more concurrent paths.

**Attributes**

Attributes are same as of PathConnection as Fork is a PathConnection.

**Associations**

None

**Constraints**

The fork element must have only one source and two or more targets.

#### 3.3.1.14 Join

**Description:** A Join is an element that synchronizes two or more concurrent scenario paths.

**Attributes**

Attributes are same as of PathConnection as Join is a PathConnection.

**Associations**

None

**Constraints**

The join element must have only one target and multiple sources.

#### 3.3.1.15 Classifier

**Description:** A Classifier describes a set of instances that have common features.

**Attributes**

| | |
|---|---|
| id: ID | the identity of classifier |
| name: string [0..1] | the name of classifier |

traceability_link: string [0..1] links to the original model

**Associations**

None

**Constraints**

None

#### 3.3.1.16 Message

**Description:** A Message is a kind of Classifier that defines the communications between steps.

**Attributes**

| | |
|---|---|
| kind: Msgkind [0..1] | the type of message |
| size: double [0..1] | the size of message |
| multi: double [0..1] | the multiplicity |

Other attributes are same as of Classifier as Message is a Classifier.

**Associations**

None

**Constraints**

None

### 3.3.2 Class Description of CSM Function Package

#### 3.3.2.1 Constraint

**Description:** A Constraint is a condition which must be met before / after the execution of Step.

**Attributes**

expression: string Condition to be met

**Associations**

None

**Constraints**

None

#### 3.3.2.2 PreCondition

**Description:** Condition that must be met before the execution of Step.

**Attributes**

expression: string Condition to be met prior to start of a Step

**Associations**

None

**Constraints**

None

#### 3.3.2.3 PostCondition

**Description:** Condition that must be met after the execution of Step.

**Attributes**

expression: string Condition to be met after completion of a Step

**Associations**

None

**Constraints**

None

#### 3.3.2.4 InputSet

**Description:** Alternative sets of input to a Step.

**Attributes**

None

**Associations**

| | |
|---|---|
| precondition : PreCondition [0..1] | constraint that must hold when the execution of step starts |
| predecessorSubset : PathConnection [1..*] | input set that must be available before the execution of a step starts. |

**Constraints**

None

#### 3.3.2.5 OutputSet

**Description:** Alternative sets of outputs from a Step.

**Attributes**

None

**Associations**

| | |
|---|---|
| postcondition :PostCondition[0..1] | constraint that must hold after the execution of step ends. |
| successorSubset: PathConnection[1..*] | output set that must be available after the execution of a step ends |

**Constraints**

None

### 3.3.3 Class Description of CSM Performance Package

#### 3.3.3.1 GeneralResource

**Description**

A GeneralResource is an abstract class that represents the available resources for execution of a Scenario.

**Attributes**

| | |
|---|---|
| id: ID | the identity of general resource element |
| name: string | the name of general resource element |
| multiplicity: int [0..1] | the property of being multiple |
| sched_policy: string [0..1] | the type of schedule policy |
| traceability_link: string [0..1] | links to the original model |

**Associations**

| | |
|---|---|
| perfMeasure: PerfMeasure [0..*] | the performance measurement attached to resource |

**Constraints**

None

#### 3.3.3.1 ResourceAcquire

**Description**

A ResourceAcquire element defines acquire step of a resource. It is a kind of step with additional attributes.

**Attributes**

| | |
|---|---|
| r_units: double | the number of resources being acquired |
| priority: string | the priority of acquiring action |

**Associations**

| | |
|---|---|
| acquire: GeneralResource[0..*] | resource that have been acquired |

**Constraints**

[1] ResourceAcquire element cannot acquire any resource that is not available.

[2] ResourceAcquire element must be paired with ResourceRelease element.

#### 3.3.3.3 ResourceRelease

**Description**

A ResourceRelease element defines release step of an acquired resource. It is a kind of step with additional attributes.

**Attributes**

| | |
|---|---|
| r_units: double | the number of resources being released |

**Associations**

release: GeneralResource[0..*] resource that have to be released

**Constraints**

[1] ResourceRelease step can not release any resource that is not acquired.

[2] ResourceRelease step must be paired with ResourceRelease element.

#### 3.3.3.4 ActiveResource

**Description**

An ActiveResource represents the available server in the performance model.

**Attributes**

op_time: double [0..1] service time

Other attributes are same as of GeneralResource as Active Resource is a subclass of GeneralResource.

**Associations**

None

**Constraints**

None

#### 3.3.3.5 PassiveResource

**Description**

A PassiveResource represents the resources that can be acquired and released.

**Attributes**

Attributes are same as of GeneralResource as Active Resource is a subclass of GeneralResource.

**Associations**

None

**Constraints**

None

#### 3.3.3.6 Component

**Description**

A component is the child of PassiveResource. Components represent abstract entities such as: actors, objects, resources, processes, containers, and so on.

**Attributes**

| | |
|---|---|
| Is_active_process: boolean [0..1] | indication of active process |

Other attributes are same as of GeneralResource as Component is a PassiveResource and PassiveResource is a GeneralResource.

**Associations**

| | |
|---|---|
| host: ProcessingResource [0..*] | the host of component |
| parent: component [0..1] | the parent of current component |
| sub: component [0..*] | the children of current component |

**Constraints**

None

#### 3.3.3.7 Workload

**Description**

A Workload is an abstract class that represents the load intensity applied to a scenario.

**Attributes**

| | |
|---|---|
| id: ID | the identity of workload element |
| arrival_pattern: ArrivalProcess | the pattern in which requests arrive |
| arrival_param1: double [0..1] | the parameter for workload |
| arrival_param2: double [0..1] | the parameter for workload |
| external_delay: double [0..1] | external delay |
| value: double [0..1] | the value |
| coeff_var_sq: double [0..1] | the coefficient |
| description: string [0..1] | the detail information |
| traceability_link: string [0..1] | links to the original model |

**Associations**

| | |
|---|---|
| perfMeasure: PerfMeasure [0..*] | the performance measurement attached to resource |

**Constraints**

None

#### 3.3.3.8 PerfMeasure

**Description**

A PerfMeasure represents the measurement such as delay, throughput and utilization during execution of a scenario.

**Attributes**

| | |
|---|---|
| id: ID | the identity of a performance measure |
| name: string | the name of a performance measure |
| traceability_link: string [0..1] | links to the original model |
| measure: perfAttribute=delay | the type of measurement |

**Associations**

| | |
|---|---|
| trigger: Step [0..1] | the step to start the performance measurement |
| end: Step [0..1] | the step to end the performance measurement |
| duration: Workload [0..1] | the duration in time |
| resource: GeneralResouce [0..1] | the resource involved in performance measurement |
| respTime: PerfValue[1] | the response time |

**Constraints**

[1] The trigger and end must be used in a pair.

[2] One of trigger, duration, or resource association is required.

#### 3.3.3.9 PerfValue

**Description**

A PerfValue represents the estimated value, which is calculated by a performance analysis tool, required value that comes from system requirements, assumed value, which comes from experience, or a measure value.

**Attributes**

| | |
|---|---|
| value: string | the calculated result |
| kind: PerfValuekind [0..1] | the type of result value |
| source: PerfValueSource [0..1] | the source of value |
| percentile: string [0..1] | the value of percentile |
| kth_moment: string [0..1] | the expected value of distribution |

**Associations**

None

**Constraints**

None

## 3.4 Advantages of Core Scenario Model

It is a very well known fact that the cost of functional/performance errors is much less if detected at the early stages of the software projects. CSM can help in validating the functional and performance needs of the software projects at initial stages. It has been defined to act as a unified intermediate interface for functional and performance models. It provides an entry point to different types of functional and performance models like Layered Queuing Networks, Petri Nets, etc. Having an intermediate CSM representation can reduce the number of transformation tools needed as explained in Section 3.1. The results achieved from the functional and performance analysis can be fed back for making critical design decision.

## 3.5 XML Schema for CSM

The CSM class diagrams in the above section represent the essential classes of the CSM metamodel and their relationships. They define the vocabulary used in our transformation. However, in a practical transformation process, we need an interchange

file format that provides a concrete syntax for the concepts in the metamodel. We use XML for this purpose. An XML schema to validate XML files was proposed for the CSM by Dorin Petriu, Prof Daniel Amyot and Yong Xiang Zeng [37].

For the automatic generation of an XML schema for the Core Scenario Model an Eclipse plug-in tool called HyperModel [18] was used. This tool can generate a schema from a UML Class Diagram. The XML schema was slightly modified manually to comply with validation tools such as XML Spy and Xerces parser. The XML schema is shown as Appendix A.

## 3.6 Chapter Summary

This chapter defines the Core Scenario Model (CSM) in detail and our contributions to it. In Section 3.2 similarities between UML Activity Diagrams and UCM are discussed which are the motivation behind the development of CSM. Section 3.3 defines the CSM metamodel and provides the description of different classes of the core and functional packages of the CSM. Section 3.5 explains the generation of XML Schema for CSM. This schema is used for validating the XML instance files for CSM. With this CSM, we are ready to support many transformations from different design specifications to CSM and from CSM to different functional and performance models. Work on transformation from UCM to CSM has been done by Prof Daniel Amyot's student, Yong Xiang Zeng [37] and the CSM instance files generated by him were used for testing the implementation done under this thesis. In the rest of this thesis, we will focus on the transformation from Activity Diagrams to Petri Nets and from CSM to Petri Nets.

# Chapter 4 Translation of Activity Diagrams into Petri Nets

## 4.1 Overview

Validation of requirements specifications is undoubtedly an integral part of requirements engineering. Validation is the process of checking whether the requirements specifications meet the intentions and expectations of the stakeholders. There are different approaches for validation of the requirements. One of the approaches is based on simulation/execution of system models that are derived from initial requirements specifications. As already mentioned in Chapter 2, Petri Nets is a formal language which can be used for validation of requirements by simulating/executing the system models.

In this chapter, we explain how the concepts introduced in Activity Diagrams UML 2.0 can be mapped to Petri Nets. This chapter also describes the concepts that are Activity Diagram specific and cannot be mapped to Petri Nets. These rules can be used for automated translation of Activity diagrams to Petri Nets using the XMI syntax for the UML diagrams. At the end we explain different possibilities of simplifications of Activity Diagrams before the transformation to Petri Nets to reduce the number of transitions, places and arcs in the resulting net.

## 4.2 Activity Diagrams, CSM and Petri Nets

The table below explains the mapping of concepts of Activity Diagrams to Petri Nets.

| Concept | Activity Diagram | Petri Nets |
|---|---|---|
| 1. Scenario representation | Activity | CP Net |
| 2. Entities | Swimlane / Partition | Will be modeled as a Place |

| Concept | Activity Diagram | Petri Nets |
|---|---|---|
| 3.Function and action performed | Action | Transition |
| 4. Scenario start and stop | Initial Node & Final Node (Flow Final) | A place without any incoming edge and a place without any outgoing edge, respectively. |
| 5. Alternative scenario | Sub activity | Subpage |
| 6. Concurrency flow | Fork Node | Will be modeled as a Transition |
| 7. Alternative flow | Decision Node | Will be modeled as a Place |
| 8. Sequence flow | ActivityEdge | Arc |
| 9.Alternative merge | Merge Node | Will be modeled as a Place |
| 10.Synchronizing concurrent flow | Join Node | Will be modeled as a Transition |
| 11. Objects | Object Node | Will be modeled as a Place |

**TABLE 4.1:** Concept Comparison Activity Diagrams and Petri Nets

## 4.2 Transformation Rules from Activity Diagrams to Petri Nets

### 4.2.1 Executable / Action Nodes

In Activity Diagrams, an action is the fundamental unit of behavior specification. An action takes a set of inputs and converts it to a set of outputs, though either or both sets may be empty. The outputs of one action may be provided as inputs to other actions using the activity flow model.

Like Actions in Activity Diagrams, Transitions are active components in Petri Nets. 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, which is similar to actions in UML 2.0.

It is proposed to map an Executable / Action Nodes in Activity Diagram to a Transition in Petri Nets as it seems to be a natural mapping.

### 4.2.2 Control Nodes

In Activity diagrams control nodes are used to route control and data through the activity graph. UML2.0 has seven concepts and five different notations for control nodes. Where as a Petri Net contains only three types of elements i.e. transition, place and arc. So it is necessary to map more than one concept of Activity Diagrams on a single concept in Petri Nets.

#### 4.2.2.1 Initial Nodes

Initial nodes receive control when an activity is started and pass it immediately along their outgoing edges. No other behavior is associated with initial nodes in UML. Initial nodes cannot have edges coming into them. The natural mapping of an Initial Node is a Place without any incoming edges, also called Source in Petri Nets.

#### 4.2.2.2 Decision Node

Decision nodes guide flow in one direction or another, but exactly which direction is determined by the constraints over the edges coming out of the node. It is proposed to map Decision node on a Place in Petri Nets.

#### 4.2.2.3 Merge Node

Merge nodes bring together multiple flows. All control and data arriving at a merge node are immediately passed to the edge coming out of the merge. Merge nodes have the same notation as decision nodes, but merge have multiple edges coming in and one going out, whereas it is the opposite for decision nodes. Flows coming into a merge are usually

alternatives from an upstream decision node. It is proposed to map Merge node on a Place in Petri Nets.

#### 4.2.2.4 Fork Node

Fork nodes in Activity Diagram are used to model concurrent flows. Control and data arriving at a fork are duplicated across the outgoing edges. No other behavior is associated with fork nodes. UML2.0 has introduced a concept of Pins which are associated with the Actions. If an action has more than one associated Output Pins, it can be used to model concurrent flow. Therefore, it is proposed to map a Fork Node on a Transition. If a Fork Node in Activity Diagram immediately precedes an Action Node we can simplify it. The simplification rules are defined in Section 4.3.

#### 4.2.2.5 Join Node

Join nodes synchronize multiple flows. In the common case, control or data must be available on every incoming edge in order to be passed to the outgoing edge. An action with more than one associated input Pins, depicts the same behavior as a join node. Therefore it is proposed to map a Join Node on a Transition.

#### 4.2.2.6 Final Node

There are two types of final nodes in Activity diagrams with the difference that Flow Final Node terminates only the local flow whereas an Activity Final node terminates the entire activity. In Petri Nets a Place with no outgoing edge is called a Sink and is like Flow Final Node in Activity Diagram. The concept of Activity Final Node cannot be mapped to sink as there is no notion of destroying tokens. This behavior can be captured programmatically while executing the Petri Nets.

### 4.2.3 Object Nodes

There are four different kinds of object nodes in UML2.0 Activity Diagrams (as explained in Section 2.4). Object nodes are used to hold data temporarily as they wait to move through the graph. They specify the type of value they can hold.

In Petri nets places are place holders for tokens. In Colored Petri Nets the places are typed for specifying the type of value a place can hold.

It is proposed to map an Object Node in Activity Diagram to a Place in Petri Nets.

### 4.2.4 Activity Edges

There are two kinds of activity edges in Activity Diagrams UML2.0, Control Flow Edges and Object Flow Edges. In Colored Petri Nets the edges are typed defining what kind of token can pass through the edge. Both kinds of edges in Activity Diagrams will be mapped on typed edges in Petri Nets.

As already explained in Chapter 2 that Petri Nets are bi-partite graphs meaning that a place cannot be preceded by a place and same is true for transitions. In Activity Diagrams or CSM there is no restriction of this type. The following rules apply during transformation of edges from activity diagrams to Petri Nets.

1) If source node of an activity edge is initial node and target nodes are action, fork or join then replace this activity edge with an arc.
2) If source of an activity edge is initial node and target nodes are decision, merge or final then replace the activity edge with an arc, a dummy transition and a dummy arc.
3) If source node of an activity edge is an action node and target nodes are action, fork or join then replace the activity edge with an arc, a dummy place and a dummy arc.
4) If source node of an activity edge is an action and target nodes are decision, merge, object or final then replace the activity edge with an arc.
5) If source node is fork or join and target nodes are action, fork or join then replace the activity edge with an arc, a dummy place and a dummy arc.
6) If source node of an activity edge is fork or join and target nodes are decision, merge or final then replace the activity edge with an arc.
7) If source node of an activity edge is decision, object or merge and target nodes are action, fork or join then replace the activity edge with an arc.

8) If source node of an activity edge is decision, object or merge and target nodes are decision, merge or final then replace the activity edge with an arc a dummy place and a dummy arc.

The rules are also presented in the following table.

| Source Node(s) of Edge | Target Node(s) of Edge | Transformation |
|---|---|---|
| • Initial Node or<br>• Decision Node or<br>• Merge Node or<br>• Object Node | • Action Node or<br>• Fork Node or<br>• Join Node | Arc |
| • Initial Node or<br>• Decision Node or<br>• Merge Node or<br>• Object Node | • Decision Node or<br>• Merge Node or<br>• Object Node or<br>• Final Node | Arc, dummy Transition and dummy Arc |
| • Action Node or<br>• Fork Node or<br>• Join Node | • Action Node or<br>• Fork Node or<br>• Join | Arc, dummy Place and dummy Arc |
| • Action Node or<br>• Fork Node or<br>• Join Node | • Decision Node or<br>• Merge Node or<br>• Object Node or<br>• Final Node | Arc |

**TABLE 4.2:** Translation rules of Activity Edges to Petri Nets

### 4.2.5 Activity Partition

Partitions are used to indicate what or who is responsible for actions grouped by the partition. There is no available notation for partitions in the Petri Nets, but they can be mapped as tokens in resource places as shown by Gregor v.Bochmann [17].

## 4.3 Simplification of Activity Diagrams before Transformation

Simplification rules are for decreasing the number of places and transitions in the generated Petri Net. It is quite possible that, while modeling with Activity Diagrams the modeler models a system in such a manner that these simplifications are never required. But if for better visual understanding of the system, the modeler opts to use the notations in the sequence described below, then we can simplify the description by applying the rules defined in the following section. At the end of this chapter we present examples of Activity Diagrams which have been taken from UML 2.0 Specification [26] and transformed in to Petri Nets after simplification of the Activity Diagram.

### 4.3.1 An Action Node followed by a Fork Node

In Activity Diagram, Fork Node is used to duplicate the tokens for concurrent flows and no other behavior is associated with it. In the proposed transformation rules, Fork Node has been mapped on a transition. So if an Action Node is followed by a Fork Node it can be replaced with an Action node with more than one outgoing edges as shown in the figure below.

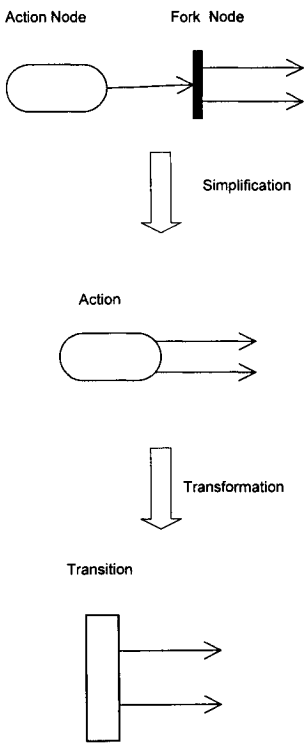



**Figure 4.1:** An Action node is followed by a Fork node.

### 4.3.2 A Fork Node followed by a Fork Node

If a Fork Node is followed by a Fork Node in an Activity Diagram it can be simplified as one Fork Node as the purpose of a Fork is to duplicate tokens.

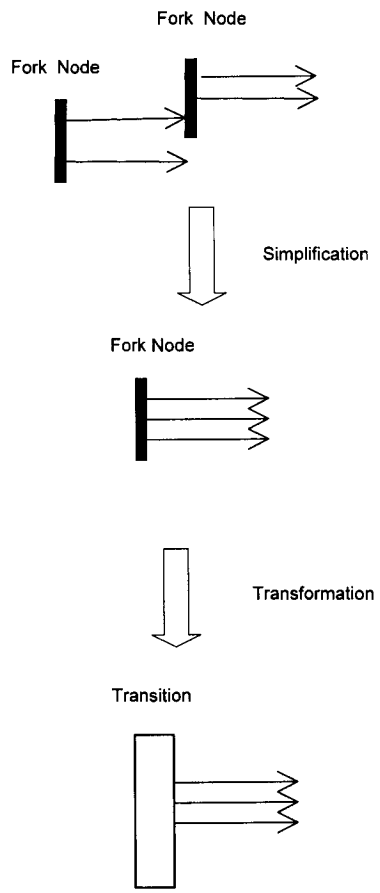



**Figure 4.2:** A Fork node is followed by a Fork node.

### 4.3.3 A Join Node followed by a Fork Node

A Join Node is used to synchronize multiple flows in Activity Diagrams. If a Join Node is followed by a Fork Node then in the simplification phase those can be replaced with Action Node as shown in the Figure 4.4

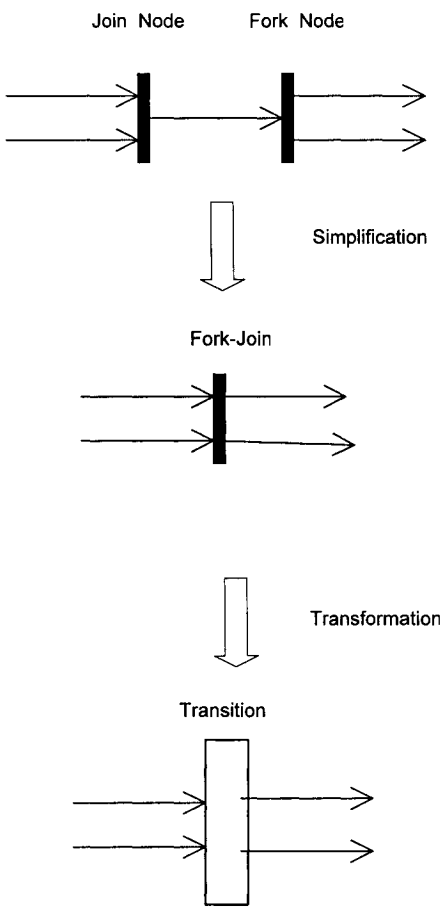



**Figure 4.3:** A Join node is followed by a Fork node.

### 4.3.4 A Join Node Followed by a Join Node

Join nodes are used to synchronize multiple flows. Join nodes take one token from each of the incoming edges and combine them as per rules described in Section 2.4.2.5. Then these tokens are offered to the out going edge. If a join node is immediately followed by another join, it means that the tokens coming to first join will be combined first and then that combined token will again be combined with the tokens coming at second join. So this process of combing can be replaced with one Join node combining all the tokens as shown in the figure below.

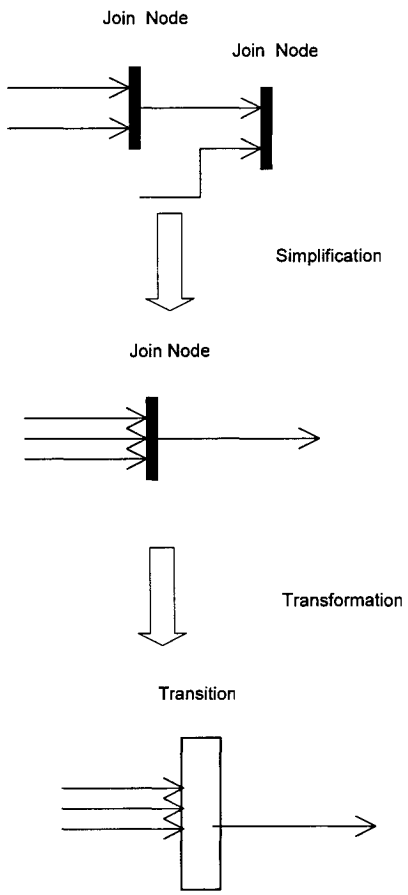



**Figure 4.4:** A Join node is followed by a Join node.

### 4.3.5 A Decision Node Followed by a Decision Node

Decision nodes are used for guiding flows in one direction or another depending on the guards over the edges coming out of the node. The guards over edges are evaluated for each individual control and data token arriving at the decision node to determine the edge the token will traverse. If a decision node is followed by another decision it means that guards will be first checked on the outgoing edges of the first decision and then the same token will again be checked against other guards over the edges on second decision node. It is possible to check these guards at one place so a decision node followed by another decision can be replaced by one Decision Node.

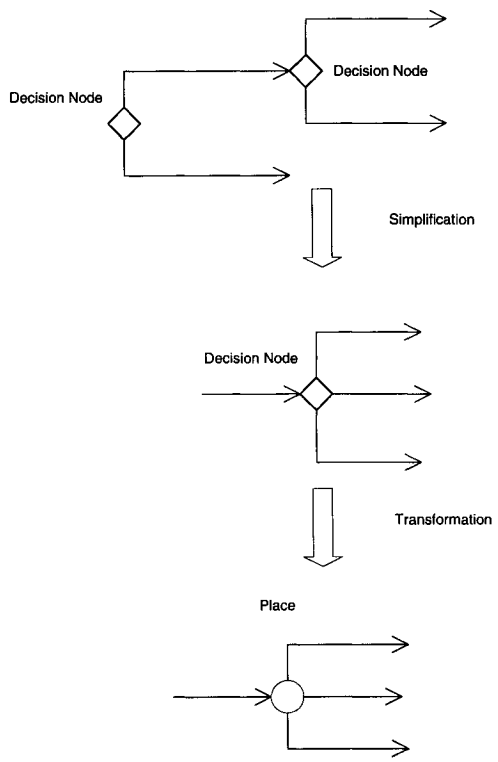



**Figure 4.5:** A Decision node is followed by a Decision node.

### 4.3.6 A Merge Node Followed by a Decision Node

Merge nodes are used to bring together multiple flows. All control and data arriving at a merge node are immediately passed to the edge coming out of the merge. If a merge node is followed by a decision node it can be replaced by one place as the token which arrives first will be evaluated against the guards on edges and this operation of arriving tokens and guard evaluation can be done at one node.

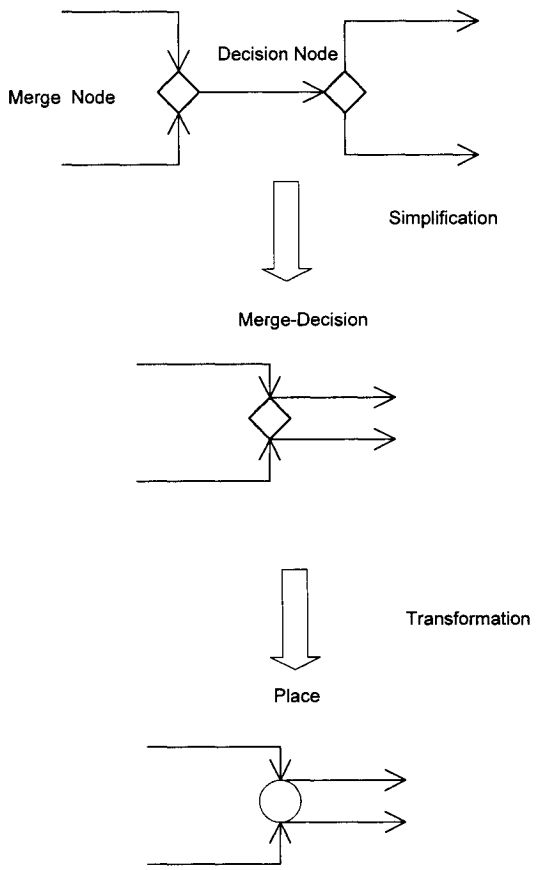



**Figure 4.6:** A Merge node is followed by a Decision node.

### 4.3.7 A Merge Node Followed by a Merge Node

If a merge node is followed by a merge node it can be replaced by one merge as the token which arrives first will be immediately forwarded to the outgoing edge if we assume that the processing time at the nodes is negligible.

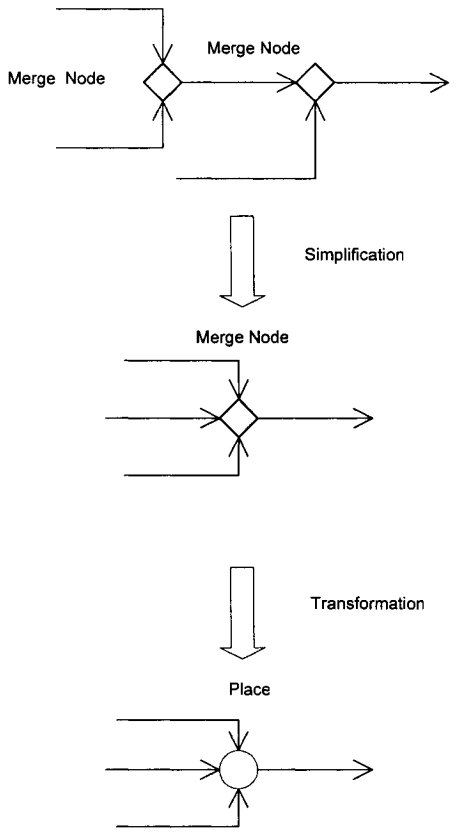



**Figure 4.7:** A Merge node is followed by a Merge node

## 4.4 Examples

The activity diagram in Figure 4.8 represents the definition of Order Processing for a business. A Parameter node for receiving the order has been shown of the left hand side on the border of the diagram. As soon as an order is received the Receive Order action is triggered. A decision node after Received Order illustrates branching based on order rejected or order accepted conditions. Fill Order is followed by a fork node which passes control both to Send Invoice and Ship Order. The join node indicates that control will be passed to the merge when both Ship Order and Accept Payment are completed. Since a

merge will just pass the token along, Close Order activity will be invoked. (Control is also passed to Close Order whenever an order is rejected.) When Close Order is completed, control passes to an activity final. Figure 4.9 shows the activity diagrams after applying the simplification rules. The Fill Order action and the following Fork (shown within a rectangular box on Figure 4.8) has been represented by the Fill Order action in Figure 4.9.

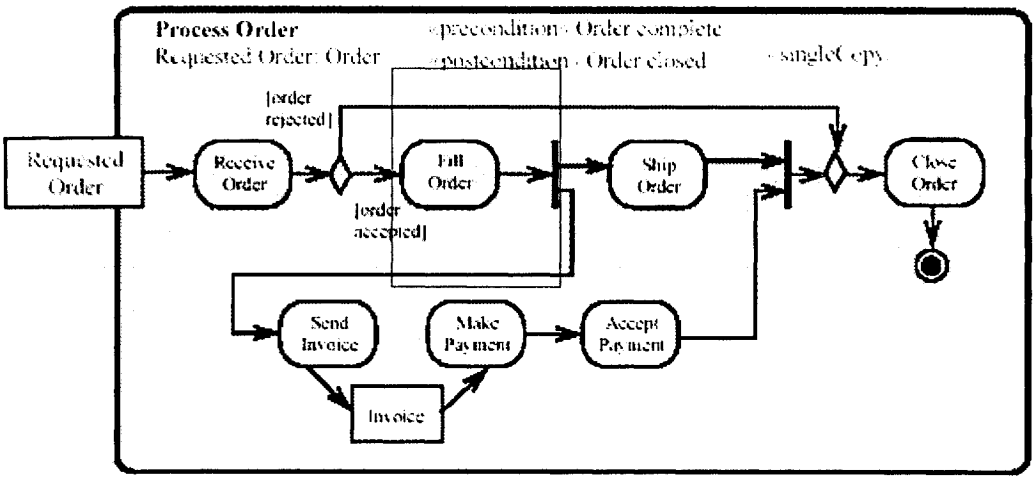



**Figure 4.8:** Activity Diagram for order processing

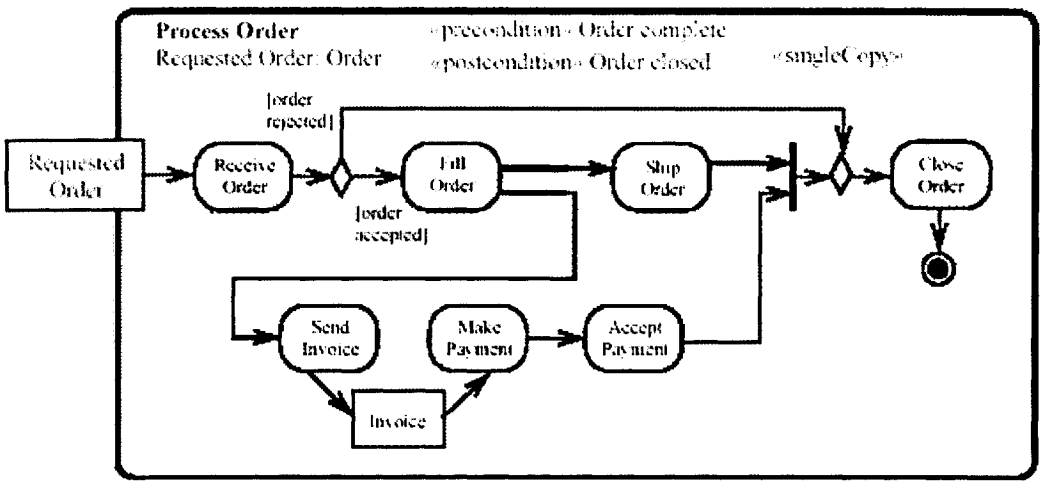



**Figure 4.9:** Activity Diagram after simplification for order processing

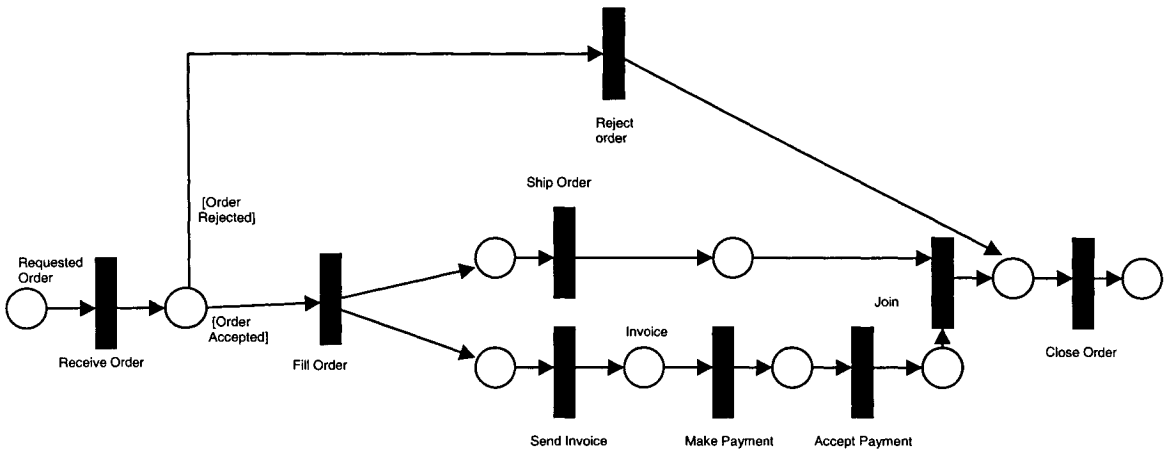



**Figure 4.10:** Petri Net after transformation for order processing

The activity diagram in Figure 4.11 is for the trouble ticket problem and has also been taken from the UML2.0 Specification [26]. The trouble ticket scenario covers quality assurance teams or customer support teams. If a "bug" or "problem" is identified; it must be recorded; the record must be checked for accuracy; from a single instance of a problem, the underlying cause is identified; a resolution is identified, which must be communicated back to the original party with the problem. Figure 4.12 shows the activity diagrams after applying the simplification rules. The decision node followed by decision node (shown within rectangular boxes on Figure 4.11) has been translated as one decision node.

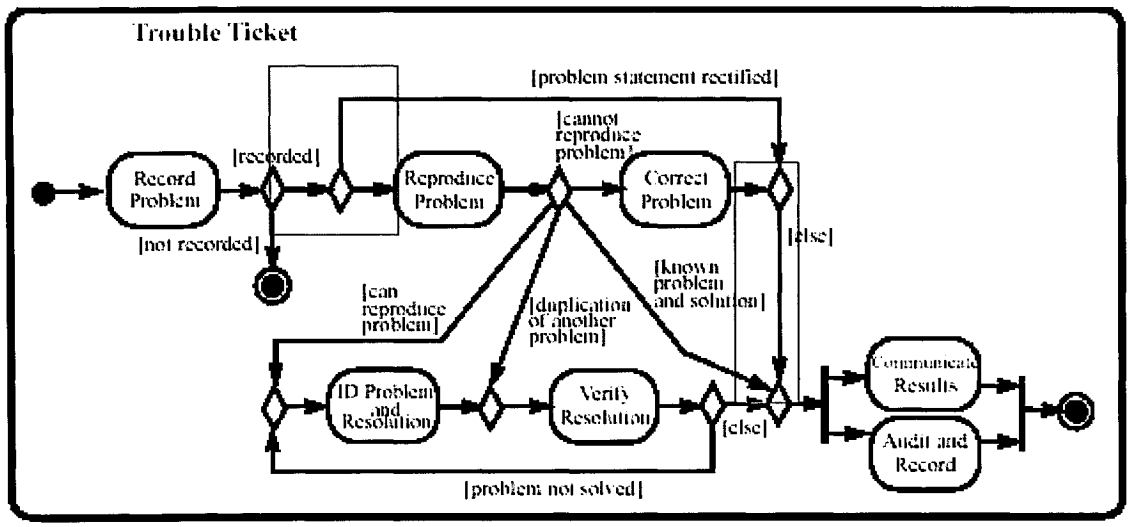



**Figure 4.11:** Activity Diagram for trouble ticket

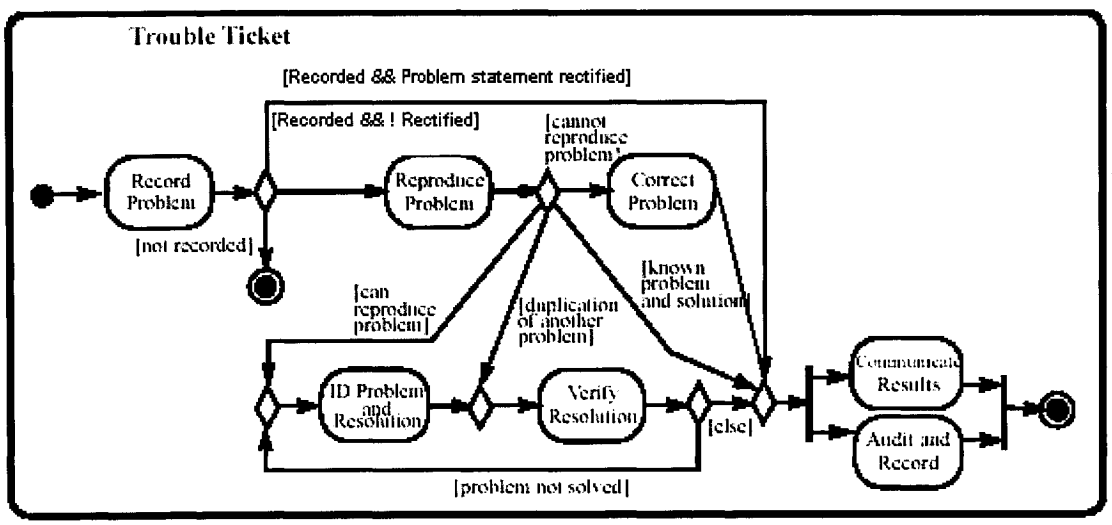



**Figure 4.12:** Activity Diagram after simplification for trouble ticket

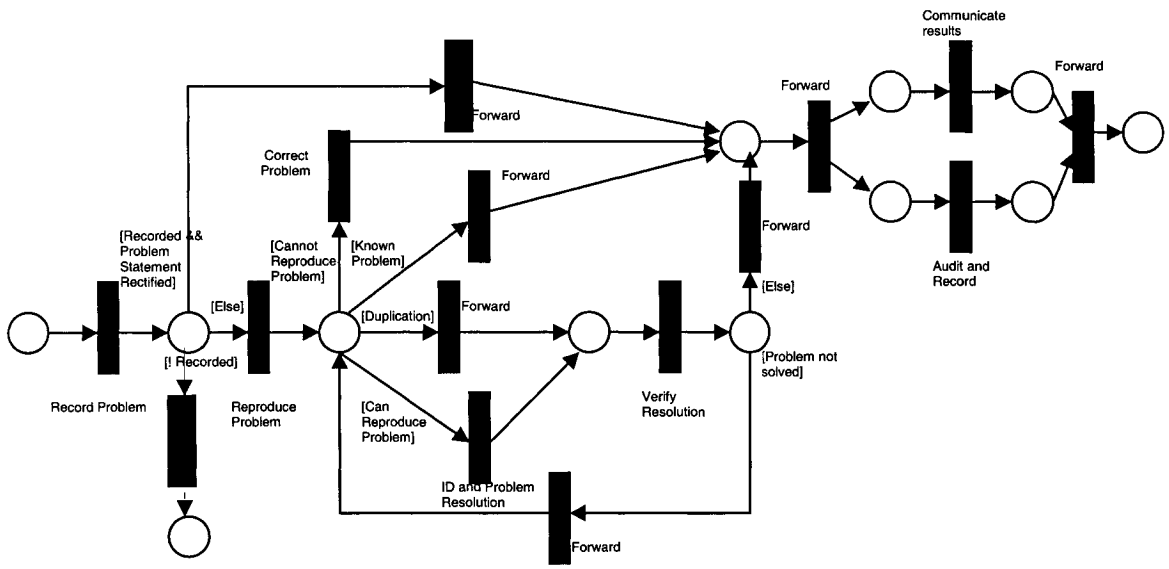



**Figure 4.13:** Petri Net after transformation for trouble ticket

## 4.5 Elements Not Mapped

The reader should notice that Petri Net is not intended to duplicate all the information found in Activity Diagrams. A consequence is that there are some concepts that are unique to Activity Diagrams, and are difficult to map into Petri Nets.

The list of the elements not mapped is as under

**Activity Final Node:** is a node that stops all the flows in an activity and immediately terminates the entire activity. In Petri Nets, there is no notation for destroying tokens.

**Interruptible Activity Region:** is for supporting termination of tokens flowing in the portions of an activity. There is no notation for destroying tokens in Petri Nets and so it cannot be mapped.

**Exception Handler:** specifies a body to execute in case of specified exception. There is no notion of exception handling in Petri Nets.

## 4.6 Chapter Summary

This chapter describes the transformation process of Activity Diagrams to the Petri Nets. We provided a mapping from Activity Diagram & CSM concepts to Petri Nets. After the transformation rules from AD to PN, we have defined some rules for simplification of Activity Diagrams before transformation to Petri Nets. These simplification rules are inline with the UML2.0 Activity Diagram Specifications. At the end we have discussed few concepts which are unique to activity Diagrams and cannot be mapped to Petri Nets.

# Chapter 5 CSM to Petri Nets

## 5.1 Overview

As already explained in Chapter 3, the Core Scenario Model (CSM) is an intermediate notation that can be transformed into formal functional and preliminary performance models for software systems. Validating the system models is an essential part of the requirements validation process. Functional validation analysis can be effective in avoiding functional disasters and generating formal requirements for the software to be built. Preliminary performance analysis can be helpful in determining the potential performance issues of the software to be built. The UML Profile for Schedulability, Performance and Time (SPT) [27] was developed for assisting to capture the performance data, and automation of the model-building step.

The figure below is a high level view of the work being done at the University of Ottawa which is part of a bigger project being done jointly with Carleton University. Our work in this chapter is the translation of a CSM XML file, generated from a UCM, into an equivalent Petri Net representation.

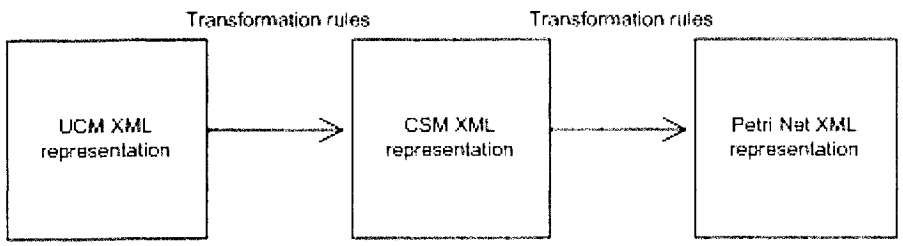



**Figure 5.1:** High level view of transformation from UCM to CSM and from CSM to Petri Nets

In this chapter we explain how the functional concepts of CSM can be mapped into Petri Nets. Also we define an algorithm for the automatic transformation of functional concepts of CSM to Petri Nets.

## 5.2 Mapping from CSM to Petri Nets

A mapping of the concepts from Activity Diagrams to Petri Nets has been shown in Table 4.1, and in Table 3.1 we have shown how the concepts of UML 2.0 Activity Diagrams have been mapped to CSM. The above mentioned tables can be used for defining a mapping of a CSM into Petri Nets. The automatic transformation can be achieved by making use of the XML Schema for CSM and Petri Net DTD which are shown in Appendix A and B. There was no documentation available for the Petri Net DTD. In order to understand the structure of this DTD, graphical views were prepared using XMLSpy and have been shown in Appendix C.

## 5.3 Transformation Algorithm

The algorithm reads an XML files containing the input Core Scenario Model and transforms it into an output Petri Net. The input file is parsed and converted into a DOM tree. Then the tree is traversed and transformation rules are applied on the nodes and the resultant nodes are stored into an arraylist. At the end, this arraylist is traversed and results are written into a CPN DTD compliant XML file.

While designing the application we had two options; either to update the original DOM tree created from the input file or create a new XML tree. We opted to create a new tree due to the reason that tracking information in the CSM elements is stored on all the nodes. If we update a node and later on we need some tracking information of the original node, it won't be there. An example of this problem is shown below. We have a Start node in a CSM file which has two target nodes h1 and h2. If we traverse the start node and update it to a Petri net place, the target information has to be deleted, but we need this information at the later stage when creating arcs for the complete transformation to work.

**CSM Element**

<Start id="h0" name="Start" target= "h1 h2" >

</Start>

**Equivalent Petri Net Element**

```
<place id="h0">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled=""/>
<lineattr colour="Black" thick="1" />
<textattr colour="Black" bold="false"/>
<text>Start</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" />
</place>
```

The main steps of the transformation algorithm (at a high level of abstraction) are as follows:

1) Parse the input xml document and create a DOM tree;
2) Find the root node and from there find the Scenario node. The Scenario node is translated as a Page in the Petri Net representation; *Note: Node in the algorithm refers to elements of the input XML file after the creation of the DOM tree.
3) Traverse through the children of Scenario node (found in step 2) in the source DOM tree and apply the following transformation rules;
4) If the node is of type Start, End, Branch or Merge then create a place and if the node is of type Step, Fork or Join then create a transition as a child of the Page in the resulting Petri Net representation and store it in the arraylist data structure.
5) If the node is of type Sequence then apply the rules shown in Table 4.2. The resulting nodes are children of the Page and stored in the arraylist data structure.
6) If a step needs a refinement repeat from step 2 to step 5.
7) After complete traversal of the source DOM tree, write the resulting output stored in the arraylist data structure to the output XML file.

The most interesting part is the middle of the algorithm that does the actual transformation. Primarily what the transformation does is to create an output model that

represents the Petri Net from the input model, based on the transformation rules that were described in Chapter 4. These rules can be roughly seen as a transformation algorithm expressed at the metamodel level. Each basic case reflects a facet of the algorithm in certain situations. In this section the transformation algorithm has been described at the implementation level, showing how the objects in the input file can be manipulated to create an output document which is compliant to the Petri Net DTD.

## 5.4 Details of the implementation

The transformation algorithm described above is at very high level and the actual implementation is much more complex than that. In the actual implementation we have to add additional Transitions, Places and Arcs along with tracking information (information about the source and target of the node under parsing) in the resulting Petri Net. The new Transitions, Places and Arcs are to be added due to the property that a Petri Net is a bi-partite graph. So we cannot have two Places or Transitions in a row. In this section, we explain the implementation and further details of the algorithm, such as handling the tracking information etc.

The implementation consists of three main parts:

1) XML input
2) Transformation
3) XML output

For the implementation of the algorithm, following options were available:

1) Using XSL Transformations
2) Using a programming language to implement the transformation

As already explained in Chapter 2, handling data (e.g. strings) is very cumbersome in XSLT as compared with other programming language like Java. XSLT is mainly used for publishing/presentation purpose. So, we decided to implement the transformation in a programming language and decided to use Java as it has very well defined Application

Programming Interfaces (APIs) for XML. We decided to use the Document Object Model (DOM) which allows random access to nodes and is a W3C standard. The Simple API for XML was another option for transformation but it allows sequential access only and so far it is not a standard of W3C.

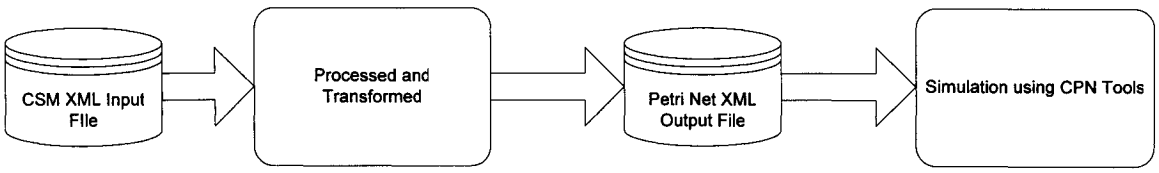



**Figure 5.2:** High level view of applications working

As already mentioned the tracking information in Petri Nets is only contained in the arcs which connect a place and a transition, unlike in CSM where the tracking information is stored in all the nodes. In our implementation we traverse the DOM tree to find the desired Node and then create an equivalent node along with its attributes, add new nodes if required as per the CPN DTD, and then store the results into an arraylist. This arraylist, after complete traversal of the document, is used to create a new CPN Tools DTD compliant XML file.

The implementation generates an XML file which did not contain the display information for the Petri Nets. Also the CPN tools do not have auto-layout feature. It would be very useful if the generated Petri Net could be visualized.

## 5.6 Chapter Summary

This chapter described the transformation process of functional concepts in CSM to the Petri Nets. We provided an algorithm for automated translation of CSM to Petri Nets. We also discussed various options available for implementation of the algorithm and reasons for choosing a programming language.

# Chapter 6 Case studies

For validation of the application developed by implementing the algorithm it was tested for all concept defined in the CSM for building functional models. For validation of the system built, black box testing was done, where we provided different inputs to the system and verified that the results produced are as desired. The inputs provided were XML representations of the CSM and the outputs generated were XML representation of the Petri Nets. Following test procedure was used for all the tests

1) Open the application in Java Editor.
2) Run the application and provide the name of input XML file.
3) Provide the name of output XML file.
4) Open the resulting output file and compare the result with the expected result.

## 6.1 Case Study 1

To illustrate the transformation from UCM to CSM and then from CSM to Petri Nets, we extend the simple call request shown in Figure 6.1 below.

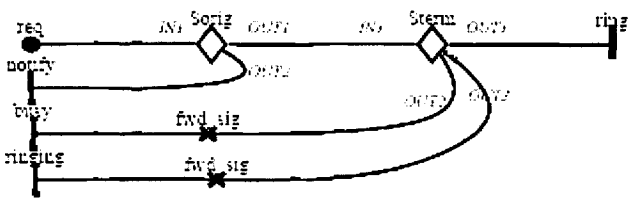



**Figure 6.1:** Call Root Map in UCM

### 6.1.1 UCM Representation of Simple Call Request

The Call Root Map shows a simple unbound UCM where a user attempts to establish a telephone call with another user through some network. The scenario contains two stubs.

The first stub (Sorig) contains the Originating plug-in whereas the second (Sterm) contains the Terminating plug-in. The plug-ins, generated with the UCM Navigator, are presented in Figures 6.2 and 6.3.

The caller first sends a connection request to the network. This request is checked by Originating Call Screening (OCS), which checks whether the call should be denied or allowed. The binding relationship for Sorig, which connects the input/output segments of a stub to the start/end points of its plug-in, is {< IN1, start >, < OUT1, success >, < OUT2, fail >}.

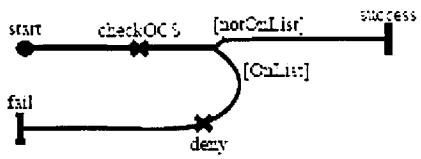



**Figure 6.2:** OCS Plug-in in UCM

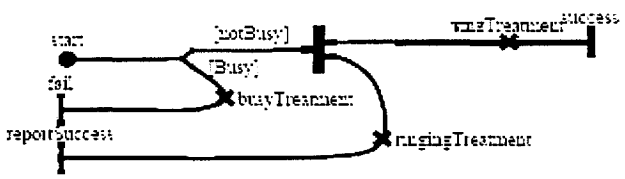



**Figure 6.3:** Terminating Plug-in in UCM

In the terminating Plug-in, it is first checked if the called number is busy or not. If the called number is busy the caller is notified and if the number is not busy then a ring message is sent to the called number and a notification is also sent to the caller. The Terminating side contains a Sterm plug-in. The binding relationship is {< IN1, start >, < OUT1, ring >, < OUT2, busy >, < OUT3, ringing >}.

### 6.1.2 UCM to CSM Transformation

Work on the transformation from UCM to CSM has been done by Yong Xiang Zeng, in his Master Thesis [37]. The resulting CSM XML file for the simple call request is shown below. The binding information for steps that need refinement is shaded grey.

```
<?xml version="1.0" encoding="utf-8"?>
<CSM name="root" description="" >
  <Scenario id="m0" name="root" description="This example is for a Simple Call
System." >
    <Start id="h0" name="Request" target= "h1" ></Start>
        <Sequence id="h1" name=" " source= "h0" target= "h2" />
        <Step id="h2" name="Sorig" description=" " predecessor= "h1" successor= "h3
h4">
         <Refinement parent="h2"  sub="m1">
                    <InBinding id="in1" start="h17" in="h1"/>
                    <OutBinding id="out1" end="h25" out="h3"/>
                    <OutBinding id="out2" end="h27" out="h4"/>
          </Refinement>
        </Step>

    <Sequence id="h4" name=" " source= "h2" target= "h5" />
    <End id="h5" name="Notify" source= "h4" />
    <Sequence id="h3" name=" " source= "h2" target= "h6" />
        <Step id="h6" name="Sterm" description=" " predecessor= "h3" successor= "h7
h9 h13">
         <Refinement parent="h6"  sub="m2">
                    <InBinding id="in2" start="h28" in="h3"/>
                    <OutBinding id="out3" end="h35" out="h7"/>
                    <OutBinding id="out4" end="h41" out="h9"/>
                    <OutBinding id="out5" end="h44" out="h13"/>
```

```
                    </Refinement>
        </Step>
        <Sequence id="h7" name=" " source= "h6" target= "h8" />
    <End id="h8" name="Ring" source= "h7" />
    <Sequence id="h9" name=" " source= "h6" target= "h10" />
    <Step id="h10" name="Fwd_sig" description=" " predecessor= "h9" successor=
"h11" />
    <Sequence id="h11" name=" " source= "h10" target= "h12" />
        <End id="h12" name="Busy" source= "h11" />
        <Sequence id="h13" name=" " source= "h6" target= "h14" />
    <Step id="h14" name="Fwd_sig" description=" "  predecessor= "h13" successor=
"h15" />
    <Sequence id="h15" name=" " source= "h14" target= "h16" />
        <End id="h16" name="Ringing" source= "h15" />
  </Scenario>

<Scenario id="m1" name="OCS Plug In" description="OCS Plug In" >
    <Start id="h17" name="Start" target= "h18" ></Start>
        <Sequence id="h18" name=" " source= "h17" target= "h19" />
        <Step id="h19" name="Check OCS" description=" " predecessor= "h18"
successor= "h20"> </Step>
        <Sequence id="h20" name=" " source= "h19" target= "h21" />
        <Branch id="h21" name=" " source= "h20" target= "h22 h26" />
        <Sequence id="h22" name=" " source= "h21" target= "h23" />
        <Step id="h23" name="Deny" description=" " predecessor= "h22" successor=
"h24"> </Step>
        <Sequence id="h24" name=" " source= "h23" target= "h25" />
    <End id="h25" name="Fail" source= "h24" />
    <Sequence id="h26" name=" " source= "h21" target= "h27" />
        <End id="h27" name="Success" source= "h26" />
  </Scenario>
```

```
<Scenario id="m2" name="Terminating Plug In" description="Terminating Plug In" >
 <Start id="h28" name="Start" target= "h29" ></Start>
      <Sequence id="h29" name=" " source= "h28" target= "h30" />
      <Branch id="h30" name=" " source= "h29" target= "h31 h32" />
       <Sequence id="h32" name=" " source= "h30" target= "h33" />
      <Step id="h33" name="Busy Treatment" description=" " predecessor= "h32"
successor= "h34"></Step>
      <Sequence id="h34" name=" " source= "h33" target= "h35" />
  <End id="h35" name="Fail" source= "h34" />
  <Sequence id="h31" name=" " source= "h30" target= "h36" />
      <Fork id="h36" name="Step" source= "h31" target="h37 h38"/>
      <Sequence id="h38" name=" " source= "h36" target= "h39" />
      <Step id="h39" name="Ringing Treatment" description=" " predecessor= "h38"
successor= "h40"></Step>
  <Sequence id="h40" name=" " source= "h39" target= "h41" />
      <End id="h41" name="Report Success" source= "h40" />

  <Sequence id="h37" name=" " source= "h36" target= "h42" />
      <Step id="h42" name="Ring Treatment" description=" " predecessor= "h37"
successor= "h43"></Step>
      <Sequence id="h43" name=" " source= "h42" target= "h44" />
      <End id="h44" name="Success" source= "h43" />
 </Scenario>
</CSM>
```

Based on the notation used by Woodside et al. [24] we have shown the results produced by Zeng [37] for the Simple Call request in Figure 6.4, 6.5 and 6.6.

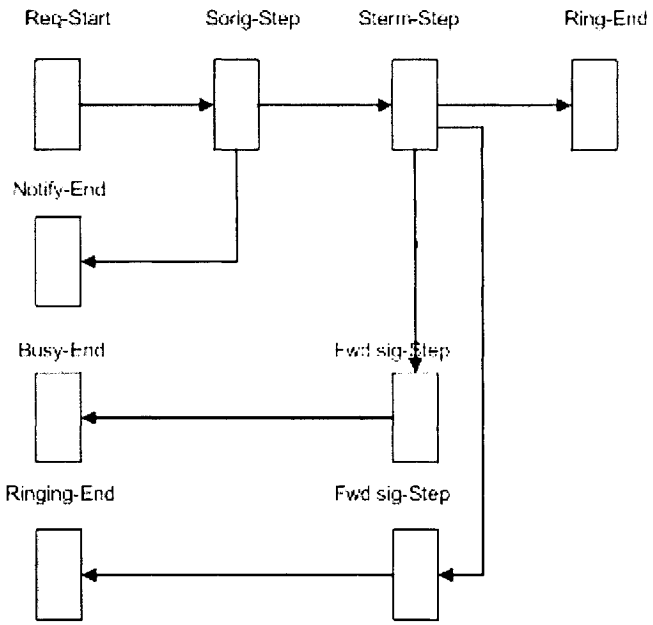



**Figure 6.4:** Basic Call Root Map in CSM

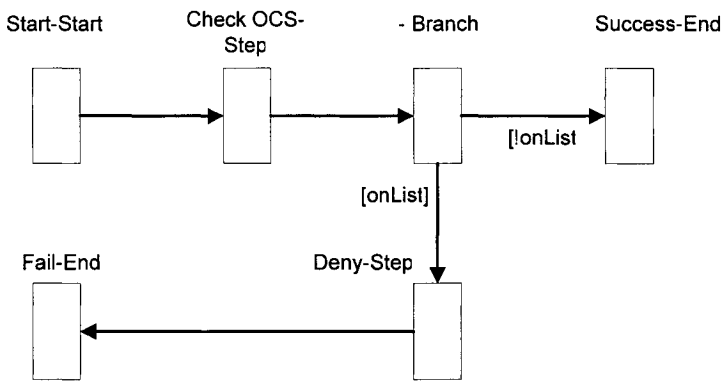



**Figure 6.5:** OCS Plug-in CSM

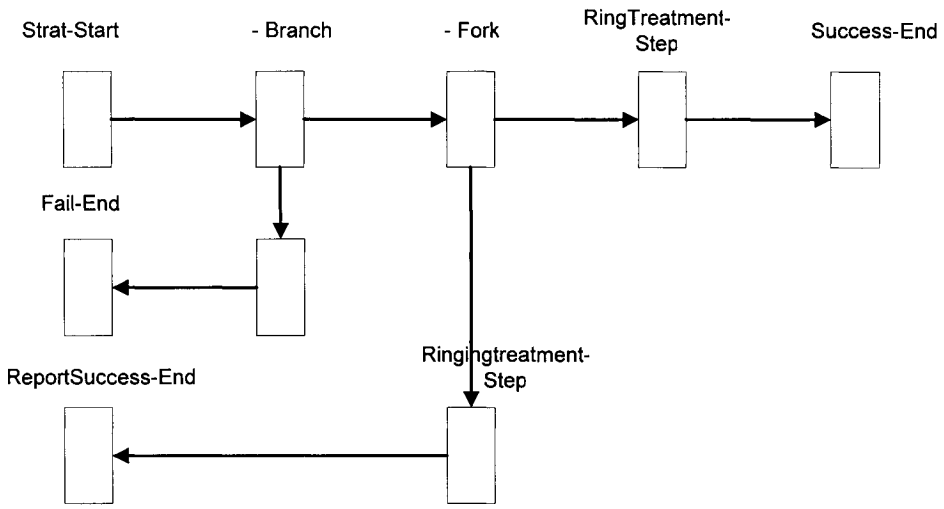



**Figure 6.6:** Terminating Plug-in in CSM

### 6.1.3 CSM to Petri Net Transformation

The Petri Net obtained by the translation from the CSM above is shown in Figures 6.7, 6.8 and 6.9. The corresponding Petri Net XML file is shown in Appendix D.

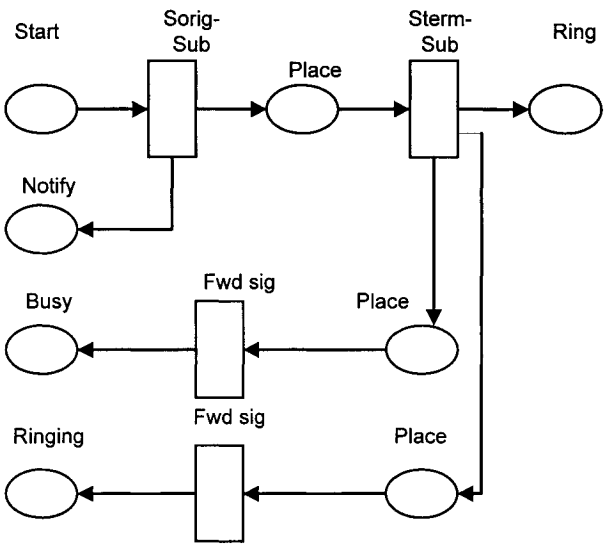



**Figure 6.7:** Basic Call Root Map in Petri Net

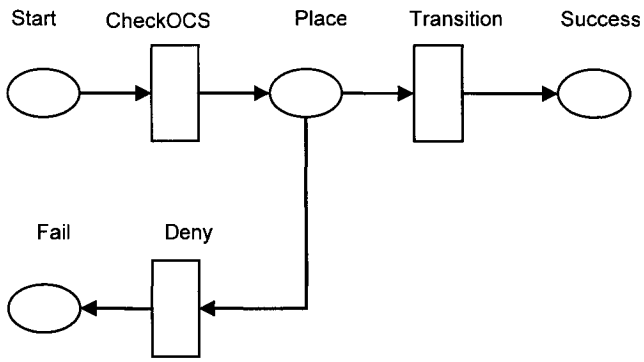



**Figure 6.8:** OCS Plug-in Petri Nets

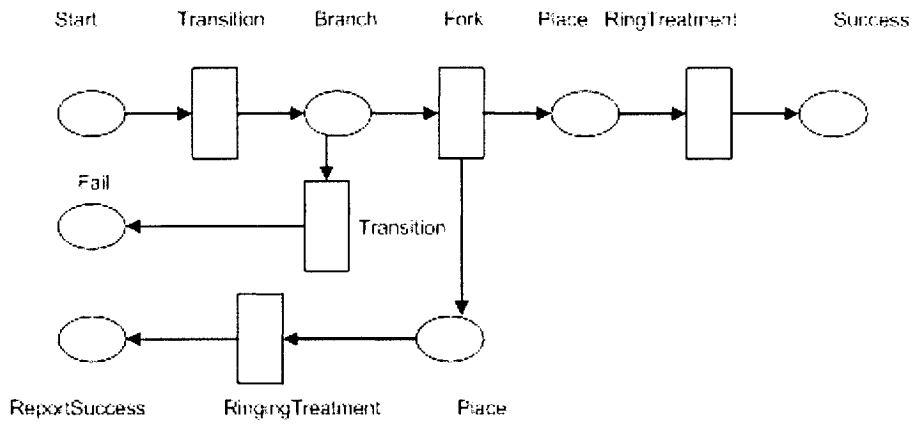



**Figure 6.9:** Terminating Plug-in in Petri Net

### 6.1.4 List of Elements/Associations/Attributes Covered

In the simple call example following elements/attributes/associations in CSM were successfully translated to equivalent Petri Net

1) CSM
2) Scenario/Sub Scenario
3) Start
4) Step
5) Sequence
6) Refinement
7) Inbinding
8) Outbinding
9) Branch
10) Fork

11) End
12) Id
13) Name
14) Source/Target
15) Successor/Predecessor
16) Perent
17) Sub
18) In/Out

## 6.2 Case Study 2

For validation of remaining functional concepts of CSM like Marge, Join etc, we used the wireless telephony example as shown in Figure 6.10.

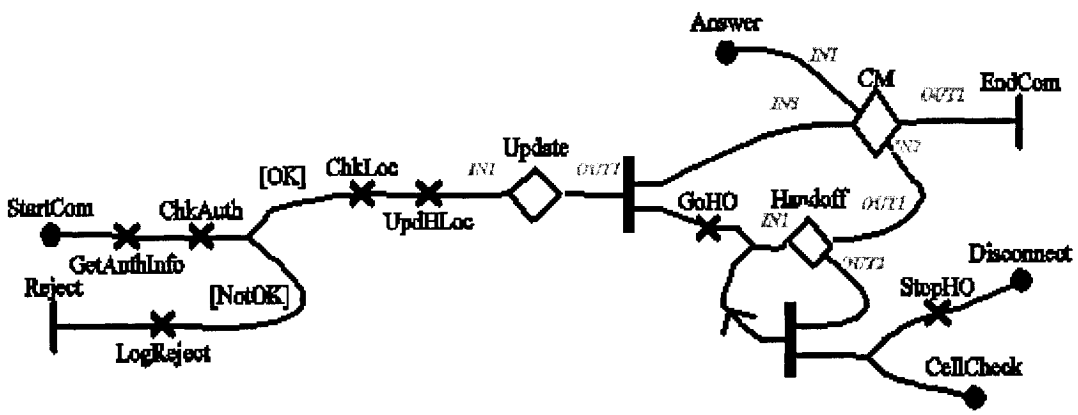



**Figure 6.10:** Wireless Telephony

The expected result was an XML representation of an equivalent Petri Net. The output from the application was checked and all the functional concepts in the XML representation of CSM were successfully translated into the resulting XML representation of Petri Nets.

### 6.2.2 List of Elements/Associations/Attributes Covered

In the wireless Telephony example following element/associations/attributes in CSM were successfully translated as XML representation of equivalent Petri Net

1) CSM
2) Scenario
3) Start
4) Step

5) Sequence
6) Refinement
7) Inbinding
8) Outbinding
9) Branch
10) Fork
11) End
12) Merge
13) Join
14) Id
15) Name
16) Source/Target
17) Successor/Predecessor
18) Perent
19) Sub
20) In/Out

## 6.4 Chapter Summary

This chapter describes a small case study of the transformation process from a CSM model into the Petri Net representation. We have drawn the figures for the CSM and the generated Petri Net manually based on XML files. UCMNav does not support the visual aspect of CSM and can only be used to generate a CSM form UCM. Our Petri Net tools do not support automatic layout.

# Chapter 7 Conclusion

## 7.1 Conclusion

Our work is one step in a larger research project aiming at deriving formal functional and performance models from UML and URN models and integrating the results of functional and performance analysis back to the these models. Although the CSM notation is still evolving, we still believe that the conceptual approach proposed in this thesis will be applicable to the future versions. The contributions of our work can be summarized as follows.

1) We proposed an approach to transform UML Activity Diagrams 2.0 to Petri Nets. The thesis reviewed the UML 2.0 Activity Diagrams and Petri Nets and proposed translation rules. It identified the metaclasses introduced in Activity diagrams and their corresponding representation in Petri Nets. It also discusses the concepts of Activity Diagrams that cannot be easily mapped on Petri Net concepts.
2) We proposed some simplification rules to Activity Diagrams prior to translation to Petri Nets in order to reduce the number of transitions, places and arcs in the resulting Petri Net.
3) The thesis describes an approach to translate functional parts of CSM into Petri Nets. It discusses our contributions in the definition of the CSM. It also identified the metaclasses introduced in CSM and their corresponding representation in Petri Nets.
4) In order to automate the transformation process the thesis proposed an algorithm for the transformation of an XML representation of a CSM into an XML representation Petri Nets. The thesis introduced the transformation rules firstly at the conceptual level and then at implementation level. This algorithm was implemented as Java application. The application uses the DOM API to process XML documents. The limitations of the implementation were also discussed.

The thesis made the first known attempt at Activity Diagram & CSM transformation to Petri Nets.

## 7.2 Future work

There are some issues which need to be addressed in future work. These issues are closely related to the challenge related to the automatic derivation of formal functional and performance models from software specification and the integration of the feedback in the these models:

- An immediate extension in the application developed would be to translate the performance information along with the functional information, contained in the Core Scenario Model and generate equivalent Stochastic Petri Net or other similar performance models. This will help in further verifying the suitability and benefits of CSM in different contexts.
- The Petri Net generated from CSM is in XML format and did not contain display layout information. It would be very useful to display such Petri Nets in Petri Net Tools.
- Performance information could also be added to Activity Diagrams and equivalent Petri Nets could be generated.

# Appendices

## A. Core Scenario Model Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Core Scenario Model Schema version 0.22 -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <!-- ========================================================================= -->
        <!-- CSM  (root element) -->
        <!-- ========================================================================= -->
        <!-- contained elements: CSMElement -->
        <!-- optional attributes: name

                description

                author

                created

                version

                traceability_link  -->
        <!-- ========================================================================= -->

        <xsd:element name="CSM" type="CSMType"/>
        <xsd:complexType name="CSMType">
                <xsd:sequence>
                        <xsd:element ref="CSMElement" minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string"/>
                <xsd:attribute name="description" type="xsd:string"/>
                <xsd:attribute name="author" type="xsd:string"/>
                <xsd:attribute name="created" type="xsd:dateTime"/>
                <xsd:attribute name="version" type="xsd:string"/>
                <xsd:attribute name="traceability_link" type="xsd:string"/>
        </xsd:complexType>

        <!-- ========================================================================= -->
        <!-- CSMElement -->
        <!-- ========================================================================= -->
        <!-- required attributes: id, name -->
        <!-- optional attributes: traceability_link -->
        <!-- ========================================================================= -->

        <xsd:element name="CSMElement" type="CSMElementType" abstract="true"/>
        <xsd:complexType name="CSMElementType">
                <xsd:attribute name="id" type="xsd:ID" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="required"/>
                <xsd:attribute name="traceability_link" type="xsd:string"/>
        </xsd:complexType>

        <!-- ========================================================================= -->
        <!-- Scenario -->
        <!-- ========================================================================= -->
        <!-- inherits from: CSMElement -->
        <!-- contained elements: ScenarioElement -->
        <!-- optional attributes: description
```

```
                probability

                transaction -->
        <!-- optional associations: refinement (Refinement IDs) -->
        <!-- ================================================================== -->

        <xsd:element name="Scenario" type="ScenarioType" substitutionGroup="CSMElement"/>
        <xsd:complexType name="ScenarioType">
                <xsd:complexContent>
                        <xsd:extension base="CSMElementType">
                                <xsd:sequence>
                                        <xsd:element ref="ScenarioElement" minOccurs="0"
maxOccurs="unbounded"/>
                                </xsd:sequence>
                                <xsd:attribute name="description" type="xsd:string"/>
                                <xsd:attribute name="probability" type="xsd:double"/>
                                <xsd:attribute name="transaction" type="xsd:boolean"/>
                                <xsd:attribute name="refinement" type="xsd:IDREFS"/>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>

        <!-- ================================================================== -->
        <!-- ScenarioElement -->
        <!-- ================================================================== -->
        <!-- required attributes: id, name -->
        <!-- optional attributes: traceability_link -->
        <!-- subtypes are: Step, PathConnection, Classifier -->
        <!-- ================================================================== -->

        <xsd:element name="ScenarioElement" type="ScenarioElementType" abstract="true"/>
        <xsd:complexType name="ScenarioElementType">
                <xsd:attribute name="id" type="xsd:ID" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="required"/>
                <xsd:attribute name="traceability_link" type="xsd:string"/>
        </xsd:complexType>

        <!-- ================================================================== -->
        <!-- Step -->
        <!-- ================================================================== -->
        <!-- inherits from: ScenarioElement -->
        <!-- contained elements: PreCondition

                PostCondition

                InputSet

                OutputSet

                ResourceAcquire

                ResourceRelease

                Refinement -->
        <!-- optional attributes: description

                host_demand

                probability

                rep_count -->
        <!-- required associations: predecessor (PathConnection IDs)
```

```
        successor (PathConnection IDs) -->
    <!-- optional associations: component (Component ID)

            parent (Scenario ID)

            extop (ExternalOperation IDs)

            perfMeasureTrigger (PerfMeasure IDs)

            perfMeasureEnd (PerfMeasure IDs) -->
    <!-- ======================================================================== -->

    <xsd:element name="Step" type="StepType" substitutionGroup="ScenarioElement"/>
    <xsd:complexType name="StepType">
        <xsd:complexContent>
            <xsd:extension base="ScenarioElementType">
                <xsd:sequence>
                    <xsd:element ref="PreCondition" minOccurs="0" maxOccurs="1"/>
                    <xsd:element ref="PostCondition" minOccurs="0" maxOccurs="1"/>
                    <xsd:element ref="InputSet" minOccurs="0"
maxOccurs="unbounded"/>
                    <xsd:element ref="OutputSet" minOccurs="0"
maxOccurs="unbounded"/>
                    <xsd:element ref="ResourceAcquire" minOccurs="0"
maxOccurs="unbounded"/>
                    <xsd:element ref="ResourceRelease" minOccurs="0"
maxOccurs="unbounded"/>
                    <xsd:element ref="Refinement" minOccurs="0"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="description" type="xsd:string"/>
                <xsd:attribute name="host_demand" type="xsd:double"/>
                <xsd:attribute name="probability" type="xsd:double"/>
                <xsd:attribute name="rep_count" type="xsd:double"/>
                <xsd:attribute name="predecessor" type="xsd:IDREFS" use="required"/>
                <xsd:attribute name="successor" type="xsd:IDREFS" use="required"/>
                <xsd:attribute name="component" type="xsd:IDREF"/>
                <xsd:attribute name="extop" type="xsd:IDREFS"/>
                <xsd:attribute name="perfMeasureTrigger" type="xsd:IDREFS"/>
                <xsd:attribute name="perfMeasureEnd" type="xsd:IDREFS"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- ======================================================================== -->
    <!-- ResourceAcquire -->
    <!-- ======================================================================== -->
    <!-- inherits from: Step -->
    <!-- optional attributes: r_units

            priority -->
    <!-- required associations: acquire (GeneralResource ID) -->
    <!-- ======================================================================== -->

    <xsd:element name="ResourceAcquire" type="ResourceAcquireType" substitutionGroup="Step"/>
    <xsd:complexType name="ResourceAcquireType">
        <xsd:complexContent>
            <xsd:extension base="StepType">
                <xsd:attribute name="r_units" type="xsd:double"/>
                <xsd:attribute name="priority" type="xsd:string"/>
                <xsd:attribute name="acquire" type="xsd:IDREF" use="required"/>
```

```
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>

            <!-- ======================================================================== -->
            <!-- ResourceRelease -->
            <!-- ======================================================================== -->
            <!-- inherits from: Step -->
            <!-- optional attributes: r_units -->
            <!-- required associations: release (GeneralResource ID) -->
            <!-- ======================================================================== -->

            <xsd:element name="ResourceRelease" type="ResourceReleaseType" substitutionGroup="Step"/>
            <xsd:complexType name="ResourceReleaseType">
                <xsd:complexContent>
                    <xsd:extension base="StepType">
                        <xsd:attribute name="r_units" type="xsd:double"/>
                        <xsd:attribute name="release" type="xsd:IDREF" use="required"/>
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>

            <!-- ======================================================================== -->
            <!-- PathConnection -->
            <!-- ======================================================================== -->
            <!-- required attributes: id -->
            <!-- optional attributes: traceability_link -->
            <!-- optional associations: source (Step IDs)

                target (Step IDs)

                classifier (Classifier IDs)

                subIn (InBinding IDs)

                subOut (OutBinding IDs) -->
            <!-- ======================================================================== -->

            <xsd:element name="PathConnection" type="PathConnectionType" abstract="true"
substitutionGroup="ScenarioElement"/>
            <xsd:complexType name="PathConnectionType">
                <xsd:complexContent>
                    <xsd:extension base="ScenarioElementType">
                        <xsd:attribute name="source" type="xsd:IDREFS"/>
                        <xsd:attribute name="target" type="xsd:IDREFS"/>
                        <xsd:attribute name="classifier" type="xsd:IDREFS"/>
                        <xsd:attribute name="subIn" type="xsd:IDREFS"/>
                        <xsd:attribute name="subOut" type="xsd:IDREFS"/>
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>

            <!-- ======================================================================== -->
            <!-- Sequence -->
            <!-- ======================================================================== -->
            <!-- inherits from: PathConnection -->
            <!-- ======================================================================== -->

            <xsd:element name="Sequence" type="SequenceType" substitutionGroup="PathConnection"/>
            <xsd:complexType name="SequenceType">
                <xsd:complexContent>
                    <xsd:extension base="PathConnectionType"/>
```

```
        </xsd:complexContent>
</xsd:complexType>

<!-- ======================================================================== -->
<!-- Branch -->
<!-- ======================================================================== -->
<!-- inherits from: PathConnection -->
<!-- ======================================================================== -->

<xsd:element name="Branch" type="BranchType" substitutionGroup="PathConnection"/>
<xsd:complexType name="BranchType">
        <xsd:complexContent>
                <xsd:extension base="PathConnectionType"/>
        </xsd:complexContent>
</xsd:complexType>

<!-- ======================================================================== -->
<!-- Merge -->
<!-- ======================================================================== -->
<!-- inherits from: PathConnection -->
<!-- ======================================================================== -->

<xsd:element name="Merge" type="MergeType" substitutionGroup="PathConnection"/>
<xsd:complexType name="MergeType">
        <xsd:complexContent>
                <xsd:extension base="PathConnectionType"/>
        </xsd:complexContent>
</xsd:complexType>

<!-- ======================================================================== -->
<!-- Fork -->
<!-- ======================================================================== -->
<!-- inherits from: PathConnection -->
<!-- ======================================================================== -->

<xsd:element name="Fork" type="ForkType" substitutionGroup="PathConnection"/>
<xsd:complexType name="ForkType">
        <xsd:complexContent>
                <xsd:extension base="PathConnectionType"/>
        </xsd:complexContent>
</xsd:complexType>

<!-- ======================================================================== -->
<!-- Join -->
<!-- ======================================================================== -->
<!-- inherits from: PathConnection -->
<!-- ======================================================================== -->

<xsd:element name="Join" type="JoinType" substitutionGroup="PathConnection"/>
<xsd:complexType name="JoinType">
        <xsd:complexContent>
                <xsd:extension base="PathConnectionType"/>
        </xsd:complexContent>
</xsd:complexType>

<!-- ======================================================================== -->
<!-- Start -->
<!-- ======================================================================== -->
<!-- inherits from: PathConnection -->
<!-- contained elements: Workload -->
<!-- optional associations: inBinding (InBinding IDs) -->
<!-- ======================================================================== -->
```

```
<xsd:element name="Start" type="StartType" substitutionGroup="PathConnection"/>
<xsd:complexType name="StartType">
        <xsd:complexContent>
                <xsd:extension base="PathConnectionType">
                        <xsd:sequence>
                                <!-- contained elements -->
                                <xsd:element ref="Workload" minOccurs="0"/>
                        </xsd:sequence>
                        <xsd:attribute name="inBinding" type="xsd:IDREFS"/>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<!-- ========================================================================= -->
<!-- End -->
<!-- ========================================================================= -->
<!-- inherits from: PathConnection -->
<!-- optional associations: outBinding (OutBinding IDs) -->
<!-- ========================================================================= -->

<xsd:element name="End" type="EndType" substitutionGroup="PathConnection"/>
<xsd:complexType name="EndType">
        <xsd:complexContent>
                <xsd:extension base="PathConnectionType">
                        <xsd:attribute name="outBinding" type="xsd:IDREFS"/>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<!-- ========================================================================= -->
<!-- InputSet -->
<!-- ========================================================================= -->
<!-- contained elements: PreCondition -->
<!-- required associations: predecessorSubset (PathConnection IDs) -->
<!-- constraint: the predecessors are a subset of the parent step's -->
<!--          predecessors -->
<!-- ========================================================================= -->

<xsd:element name="InputSet" type="InputSetType"/>
<xsd:complexType name="InputSetType">
        <xsd:sequence>
                <xsd:element ref="PreCondition" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="predecessorSubset" type="xsd:IDREFS" use="required"/>
</xsd:complexType>

<!-- ========================================================================= -->
<!-- OutputSet -->
<!-- ========================================================================= -->
<!-- contained elements: PostCondition -->
<!-- required associations: successorSubset (PathConnection IDs) -->
<!-- constraint: the successors are a subset of the parent step's -->
<!--          successors -->
<!-- ========================================================================= -->

<xsd:element name="OutputSet" type="OutputSetType"/>
<xsd:complexType name="OutputSetType">
        <xsd:sequence>
                <xsd:element ref="PostCondition" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="successorSubset" type="xsd:IDREFS" use="required"/>
```

```
</xsd:complexType>

<!-- ========================================================================= -->
<!-- Constraint -->
<!-- ========================================================================= -->
<!-- required attributes: expression -->
<!-- ========================================================================= -->

<xsd:element name="Constraint" type="ConstraintType" abstract="true"/>
<xsd:complexType name="ConstraintType">
        <xsd:attribute name="expression" type="xsd:string" use="required"/>
</xsd:complexType>

<!-- ========================================================================= -->
<!-- PreCondition -->
<!-- ========================================================================= -->
<!-- inherits from: Constraint -->
<!-- ========================================================================= -->

<xsd:element name="PreCondition" type="PreConditionType" substitutionGroup="Constraint"/>
<xsd:complexType name="PreConditionType">
        <xsd:complexContent>
                <xsd:extension base="ConstraintType"/>
        </xsd:complexContent>
</xsd:complexType>

<!-- ========================================================================= -->
<!-- PostCondition -->
<!-- ========================================================================= -->
<!-- inherits from: Constraint -->
<!-- ========================================================================= -->

<xsd:element name="PostCondition" type="PostConditionType" substitutionGroup="Constraint"/>
<xsd:complexType name="PostConditionType">
        <xsd:complexContent>
                <xsd:extension base="ConstraintType"/>
        </xsd:complexContent>
</xsd:complexType>

<!-- ========================================================================= -->
<!-- Classifier -->
<!-- ========================================================================= -->
<!-- inherits from: ScenarioElement -->
<!-- ========================================================================= -->

<xsd:element name="Classifier" type="ClassifierType" substitutionGroup="ScenarioElement"/>
<xsd:complexType name="ClassifierType">
        <xsd:complexContent>
                <xsd:extension base="ScenarioElementType"/>
        </xsd:complexContent>
</xsd:complexType>

<!-- ========================================================================= -->
<!-- Message -->
<!-- ========================================================================= -->
<!-- inherits from: Classifier -->
<!-- required attributes: kind -->
<!-- optional attributes: size
        multi -->
<!-- ========================================================================= -->

<xsd:element name="Message" type="MessageType" substitutionGroup="Classifier"/>
```

```
<xsd:complexType name="MessageType">
        <xsd:complexContent>
                <xsd:extension base="ClassifierType">
                        <xsd:attribute name="kind" type="MsgKind" default="async"/>
                        <xsd:attribute name="size" type="xsd:double"/>
                        <xsd:attribute name="multi" type="xsd:double"/>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<!-- ========================================================================== -->
<!-- <<enumeration>> MsgKind -->
<!-- ========================================================================== -->
<!-- values: async | sync | reply -->
<!-- ========================================================================== -->

<xsd:simpleType name="MsgKind">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="async"/>
                <xsd:enumeration value="sync"/>
                <xsd:enumeration value="reply"/>
        </xsd:restriction>
</xsd:simpleType>

<!-- ========================================================================== -->
<!-- Refinement -->
<!-- ========================================================================== -->
<!-- contained elements: InBinding

     OutBinding -->
<!-- optional attributes: selection_cond -->
<!-- required associations: parent (Step ID)

        sub (Scenario ID) -->
<!-- ========================================================================== -->

<xsd:element name="Refinement" type="RefinementType"/>
<xsd:complexType name="RefinementType">
        <xsd:sequence>
                <xsd:element ref="InBinding" maxOccurs="unbounded"/>
                <xsd:element ref="OutBinding" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="selection_cond" type="xsd:string"/>
        <xsd:attribute name="parent" type="xsd:IDREF" use="required"/>
        <xsd:attribute name="sub" type="xsd:IDREF" use="required"/>
</xsd:complexType>

<!-- ========================================================================== -->
<!-- InBinding -->
<!-- ========================================================================== -->
<!-- required attributes: id -->
<!-- required associations: in (PathConnection ID)

        start (Start ID) -->
<!-- ========================================================================== -->

<xsd:element name="InBinding" type="InBindingType"/>
<xsd:complexType name="InBindingType">
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
        <xsd:attribute name="in" type="xsd:IDREF" use="required"/>
        <xsd:attribute name="start" type="xsd:IDREF" use="required"/>
</xsd:complexType>
```

```
        <!-- ========================================================================== -->
        <!-- OutBinding -->
        <!-- ========================================================================== -->
        <!-- required attributes: id -->
        <!-- required associations: end (End ID)

                out (PathConnection ID) -->
        <!-- ========================================================================== -->

        <xsd:element name="OutBinding" type="OutBindingType"/>
        <xsd:complexType name="OutBindingType">
                <xsd:attribute name="id" type="xsd:ID" use="required"/>
                <xsd:attribute name="end" type="xsd:IDREF" use="required"/>
                <xsd:attribute name="out" type="xsd:IDREF" use="required"/>
        </xsd:complexType>

        <!-- ========================================================================== -->
        <!-- GeneralResource -->
        <!-- ========================================================================== -->
        <!-- inherits from: CSMElement -->
        <!-- optional attributes: multiplicity

                sched_policy -->
        <!-- optional associations: perfMeasure (PerfMeasure IDs) -->
        <!-- ========================================================================== -->

        <xsd:element name="GeneralResource" type="GeneralResourceType" abstract="true"
substitutionGroup="CSMElement"/>
        <xsd:complexType name="GeneralResourceType">
                <xsd:complexContent>
                        <xsd:extension base="CSMElementType">
                                <xsd:attribute name="multiplicity" type="xsd:int"/>
                                <xsd:attribute name="sched_policy" type="xsd:string"/>
                                <xsd:attribute name="perfMeasure" type="xsd:IDREFS"/>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>

        <!-- ========================================================================== -->
        <!-- PassiveResource -->
        <!-- ========================================================================== -->
        <!-- inherits from: GeneralResource -->
        <!-- ========================================================================== -->

        <xsd:element name="PassiveResource" type="PassiveResourceType"
substitutionGroup="GeneralResource"/>
        <xsd:complexType name="PassiveResourceType">
                <xsd:complexContent>
                        <xsd:extension base="GeneralResourceType"/>
                </xsd:complexContent>
        </xsd:complexType>

        <!-- ========================================================================== -->
        <!-- Component -->
        <!-- ========================================================================== -->
        <!-- inherits from: PassiveResource -->
        <!-- optional attributes: is_active -->
        <!-- optional associations: host (ProcessingResource ID)

                parent (Component ID)
```

```
        sub (Component IDs) -->
    <!-- ========================================================================= -->

    <xsd:element name="Component" type="ComponentType" substitutionGroup="PassiveResource"/>
    <xsd:complexType name="ComponentType">
        <xsd:complexContent>
            <xsd:extension base="PassiveResourceType">
                <xsd:attribute name="is_active_process" type="xsd:boolean"/>
                <xsd:attribute name="host" type="xsd:IDREF" use="required"/>
                <xsd:attribute name="parent" type="xsd:IDREF"/>
                <xsd:attribute name="sub" type="xsd:IDREFS"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- ========================================================================= -->
    <!-- ActiveResource -->
    <!-- ========================================================================= -->
    <!-- inherits from: GeneralResource -->
    <!-- optional attributes: operation_time -->
    <!-- ========================================================================= -->

    <xsd:element name="ActiveResource" type="ActiveResourceType" substitutionGroup="GeneralResource"/>
    <xsd:complexType name="ActiveResourceType">
        <xsd:complexContent>
            <xsd:extension base="GeneralResourceType">
                <xsd:attribute name="op_time" type="xsd:double"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- ========================================================================= -->
    <!-- ProcessingResource -->
    <!-- ========================================================================= -->
    <!-- inherits from: ActiveResource -->
    <!-- ========================================================================= -->

    <xsd:element name="ProcessingResource" type="ProcessingResourceType"
substitutionGroup="ActiveResource"/>
    <xsd:complexType name="ProcessingResourceType">
        <xsd:complexContent>
            <xsd:extension base="ActiveResourceType"/>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- ========================================================================= -->
    <!-- ExternalOperation -->
    <!-- ========================================================================= -->
    <!-- inherits from: ActiveResource -->
    <!-- optional attributes: description -->
    <!-- ========================================================================= -->

    <xsd:element name="ExternalOperation" type="ExternalOperationType"
substitutionGroup="ActiveResource"/>
    <xsd:complexType name="ExternalOperationType">
        <xsd:complexContent>
            <xsd:extension base="ActiveResourceType">
                <xsd:attribute name="description" type="xsd:string"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

```
<!-- ========================================================================== -->
<!-- Workload -->
<!-- ========================================================================== -->
<!-- optional attributes: arrival_pattern

        arrival_param1

        arrival_param2

        external_delay

        value

        coeff_var_sq

        description

        traceability_link -->
<!-- optional associations: response_time (PerfMeasure IDs) -->
<!-- ========================================================================== -->

<xsd:element name="Workload" type="WorkloadType" abstract="true"/>
<xsd:complexType name="WorkloadType">
        <xsd:attribute name="id" type="xsd:ID" use="required"/>
        <xsd:attribute name="arrival_pattern" type="ArrivalProcess" default="poissonPDF"/>
        <xsd:attribute name="arrival_param1" type="xsd:double"/>
        <xsd:attribute name="arrival_param2" type="xsd:double"/>
        <xsd:attribute name="external_delay" type="xsd:double"/>
        <xsd:attribute name="value" type="xsd:double"/>
        <xsd:attribute name="coeff_var_sq" type="xsd:double"/>
        <xsd:attribute name="description" type="xsd:string"/>
        <xsd:attribute name="traceability_link" type="xsd:string"/>
        <xsd:attribute name="responseTime" type="xsd:IDREFS"/>
</xsd:complexType>

<!-- ========================================================================== -->
<!-- ClosedWorkload -->
<!-- ========================================================================== -->
<!-- inherits from: Workload -->
<!-- required attributes: population -->
<!-- ========================================================================== -->

<xsd:element name="ClosedWorkload" type="ClosedWorkloadType" substitutionGroup="Workload"/>
<xsd:complexType name="ClosedWorkloadType">
        <xsd:complexContent>
                <xsd:extension base="WorkloadType">
                        <xsd:attribute name="population" type="xsd:int" use="required"/>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>

<!-- ========================================================================== -->
<!-- OpenWorkload -->
<!-- ========================================================================== -->
<!-- inherits from: Workload -->
<!-- ========================================================================== -->

<xsd:element name="OpenWorkload" type="OpenWorkloadType" substitutionGroup="Workload"/>
<xsd:complexType name="OpenWorkloadType">
        <xsd:complexContent>
                <xsd:extension base="WorkloadType"/>
        </xsd:complexContent>
```

```
        </xsd:complexType>

        <!-- ========================================================================= -->
        <!-- <<enumeration>> ArrivalProcess -->
        <!-- ========================================================================= -->
        <!-- values: poissonPDF | periodic | uniform | phase_type -->
        <!--       (not included: bounded, bursty, bernoulli, binomial) -->
        <!-- ========================================================================= -->

        <xsd:simpleType name="ArrivalProcess">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="poissonPDF"/>
                        <xsd:enumeration value="periodic"/>
                        <xsd:enumeration value="uniform"/>
                        <xsd:enumeration value="phase_type"/>
                </xsd:restriction>
        </xsd:simpleType>

        <!-- ========================================================================= -->
        <!-- PerfMeasure -->
        <!-- ========================================================================= -->
        <!-- contained elements: PerfValue -->
        <!-- required attributes: measure_type -->
        <!-- optional association: trigger (Step ID) -->
        <!-- optional association: end (Step ID) -->
        <!-- optional association: duration (Workload ID) -->
        <!-- optional association: resource (GeneralResource ID) -->
        <!-- constraint: one of trigger, duration, or resource is required. -->
        <!-- constraint: end can only be appear if trigger is present. -->
        <!-- ========================================================================= -->

        <xsd:element name="PerfMeasure" type="PerfMeasureType" substitutionGroup="CSMElement"/>
        <xsd:complexType name="PerfMeasureType">
                <xsd:complexContent>
                        <xsd:extension base="CSMElementType">
                                <xsd:sequence>
                                        <xsd:element ref="PerfValue" minOccurs="0"
maxOccurs="unbounded"/>
                                </xsd:sequence>
                                <xsd:attribute name="measure" type="PerfAttribute" default="delay"/>
                                <xsd:attribute name="trigger" type="xsd:IDREF"/>
                                <xsd:attribute name="end" type="xsd:IDREF"/>
                                <xsd:attribute name="duration" type="xsd:IDREF"/>
                                <xsd:attribute name="resource" type="xsd:IDREF"/>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>

        <!-- ========================================================================= -->
        <!-- PerfValue -->
        <!-- ========================================================================= -->
        <!-- required attributes: value -->
        <!-- optional attributes: source

                kind

                percentile

                kth_moment -->
        <!-- ========================================================================= -->

        <xsd:element name="PerfValue" type="PerfValueType"/>
```

```
	<xsd:complexType name="PerfValueType">
		<xsd:attribute name="value" type="xsd:string" use="required"/>
		<xsd:attribute name="kind" type="PerfValueKind" default="mean"/>
		<xsd:attribute name="source" type="PerfValueSource" default="required"/>
		<xsd:attribute name="percentile" type="xsd:string"/>
		<xsd:attribute name="kth_moment" type="xsd:string"/>
	</xsd:complexType>

	<!-- ========================================================================== -->
	<!-- <<enumeration>> PerfValueKind -->
	<!-- ========================================================================== -->
	<!-- values: mean | variance | percentile | moment | min | max | distribution -->
	<!-- ========================================================================== -->

	<xsd:simpleType name="PerfValueKind">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="mean"/>
			<xsd:enumeration value="variance"/>
			<xsd:enumeration value="percentile"/>
			<xsd:enumeration value="moment"/>
			<xsd:enumeration value="min"/>
			<xsd:enumeration value="max"/>
			<xsd:enumeration value="distribution"/>
		</xsd:restriction>
	</xsd:simpleType>

	<!-- ========================================================================== -->
	<!-- <<enumeration>> PerfValueSource -->
	<!-- ========================================================================== -->
	<!-- values: required | assumed | predicted | measured -->
	<!-- ========================================================================== -->

	<xsd:simpleType name="PerfValueSource">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="required"/>
			<xsd:enumeration value="assumed"/>
			<xsd:enumeration value="predicted"/>
			<xsd:enumeration value="measured"/>
		</xsd:restriction>
	</xsd:simpleType>

	<!-- ========================================================================== -->
	<!-- <<enumeration>> PerfAttribute -->
	<!-- ========================================================================== -->
	<!-- values: delay | throughput | utilization | interval | wait -->
	<!-- ========================================================================== -->

	<xsd:simpleType name="PerfAttribute">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="delay"/>
			<xsd:enumeration value="throughput"/>
			<xsd:enumeration value="utilization"/>
			<xsd:enumeration value="interval"/>
			<xsd:enumeration value="wait"/>
		</xsd:restriction>
	</xsd:simpleType>
</xsd:schema>
```

# B. Colored Petri Nets Document Type Definition (DTD)

```
<!--
  COPYRIGHT (C) 2002 by the CPN group, University of Aarhus, Denmark.

  Contact: cpntools-support@daimi.au.dk
  WWW URL: http://www.daimi.au.dk/CPNtools/

  File: cpn.dtd
  DTD for XML format for CPN Tools ver. 1.0.0
   20031121: Updated with hidden attribute for marking.
         (Comming in version 1.0.1)
   20030120: Updated with token and marking elements for places.
         (Comming in version 0.1.48)

  Note: For CPN Tools ver. 0.1.47 the format attribute of the
      generator tag must be "2".
-->

<!-- The possible orientations of an arc:
     bothdir = Bidirectional arc: O<->[]
     nodir = Arc without arrows: O-[]
     ptot = Arc from Place to Transition: 0->[]
     ttop = Arc from Transition to Place: []->O -->
<!ENTITY % arcors "CDATA">

<!-- Boolean values -->
<!ENTITY % boolean "(true | false)">

<!-- Colours. These colours corresponds to the standard colours of
    HTML:
    Name and RGB value:
       black  = #000000   green  = #008000
       silver = #c0c0c0   lime   = #00ff00
       gray   = #808080   olive  = #808000
       white  = #ffffff   yellow = #ffff00
       maroon = #800000   navy   = #000080
       red    = #ff0000   blue   = #0000ff
       purple = #800080   teal   = #008080
       fuchsia= #ff00ff   aqua   = #00ffff -->
<!ENTITY % cols "CDATA">

<!-- Types of declarations -->
<!ENTITY % decls "block | color | var | ml | globref">

<!-- Line types: -->
<!ENTITY % lintyps "CDATA">

<!-- Numbers -->
<!ENTITY % number "CDATA">

<!-- Possible attributes of objects:
     posattr = Position attributes
     fillattr = Fill Attributes
     lineattr = Line Attributes
     textattr = Text Attributes -->
<!ENTITY % objatts "posattr, fillattr, lineattr, textattr">

<!-- Possible fill patterns for objects -->
<!ENTITY % pats "CDATA">
```

```
<!-- Possible porttypes:
     in = Input Port.
     out = Output Port.
     inout = Input/Output Port
     general = General Port -->
<!ENTITY % prttyps "CDATA">

<!ELEMENT alias    (id)?>

<!ELEMENT and      (ml)+>

<!ELEMENT annot                (%objatts;, text)>
<!ATTLIST annot    id          ID          #IMPLIED>

<!ELEMENT arc      (%objatts;, arrowattr, transend, placeend, ((annot?, bendpoint*)|
                                    (bendpoint*, annot?)))>
<!ATTLIST arc      id          ID          #IMPLIED
                   orientation %arcors;    #IMPLIED>

<!ELEMENT arrowattr EMPTY>
<!ATTLIST arrowattr headsize   %number;    #IMPLIED
                   currentcyckle %number;  #IMPLIED>

<!ELEMENT Aux      (%objatts;, label, text)>
<!ATTLIST Aux      id          ID          #IMPLIED>

<!ELEMENT bendpoint (%objatts;, text)>
<!ATTLIST bendpoint id         ID          #IMPLIED
                   serial      %number;    #IMPLIED>

<!ELEMENT block    (id, (%decls;)*)>
<!ATTLIST block    id          ID          #IMPLIED>

<!ELEMENT bool     (with)?>

<!ELEMENT box      EMPTY>
<!ATTLIST box      w           %number;    #IMPLIED
                   h           %number;    #IMPLIED>

<!ELEMENT by       (ml)>

<!ELEMENT color    (id, declare?, timed?, (unit | bool | int | real | string |
                        enum | index | product | record | list |
                        union | alias | subset)*)>
<!ATTLIST color    id          ID          #IMPLIED>

<!ELEMENT code     (%objatts;, text)>
<!ATTLIST code     id          ID          #IMPLIED>

<!ELEMENT code-key  (%objatts;, text)>
<!ATTLIST code-key  id         ID          #IMPLIED>

<!ELEMENT cond     (%objatts;, text)>
<!ATTLIST cond     id          ID          #IMPLIED>

<!ELEMENT cpnet    (globbox, page*, fusion*)>

<!ELEMENT declare  (id)+>

<!ELEMENT ellipse  EMPTY>
<!ATTLIST ellipse  w           %number;    #IMPLIED
```

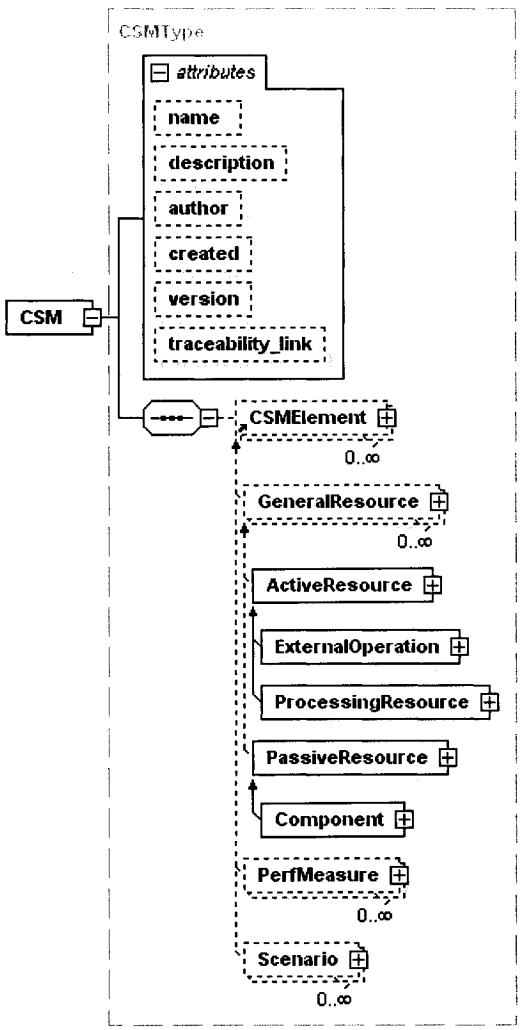
```
            h       %number;   #IMPLIED>

<!ELEMENT enum     (id)+>

<!ELEMENT fillattr  EMPTY>
<!ATTLIST fillattr  colour     %cols;     #IMPLIED
            pattern    %pats;     #IMPLIED
            filled     %boolean;  #IMPLIED>

<!ELEMENT fusion    (fusion_elm*)>
<!ATTLIST fusion    id         ID         #IMPLIED
            name       CDATA      #IMPLIED>

<!ELEMENT fusion_elm EMPTY>
<!ATTLIST fusion_elm idref     IDREF      #IMPLIED>

<!ELEMENT fusioninfo (%objatts;)>
<!ATTLIST fusioninfo id        ID         #IMPLIED
             name      CDATA      #IMPLIED>

<!ELEMENT generator EMPTY>
<!ATTLIST generator tool       CDATA      #IMPLIED
            version    CDATA      #IMPLIED
            format     CDATA      #IMPLIED>

<!-- Note: format must be "2" for CPN Tools ver. 0.1.47 -->

<!ELEMENT globbox   (%decls;)*>

<!ELEMENT globref   ((id)?, (ml)?)>
<!ATTLIST globref   id     ID         #IMPLIED>

<!ELEMENT group_elm EMPTY>
<!ATTLIST group_elm idref  IDREF      #IMPLIED>

<!ELEMENT group (group_elm)*>
<!ATTLIST group id         ID         #IMPLIED
          name     CDATA      #IMPLIED>

<!ELEMENT guideline_elm EMPTY>
<!ATTLIST guideline_elm idref  IDREF      #IMPLIED>

<!ELEMENT hguideline (guideline_elm)*>
<!ATTLIST hguideline id        ID         #IMPLIED
             y         %number;   #IMPLIED>

<!ELEMENT id       (#PCDATA)>

<!ELEMENT index    (ml, ml, id)>

<!ELEMENT initmark  (%objatts;, text)>
<!ATTLIST initmark  id         ID         #IMPLIED>

<!ELEMENT int      (with)?>

<!ELEMENT label    EMPTY>
<!ATTLIST label    w          %number;   #IMPLIED
            h          %number;   #IMPLIED>

<!ELEMENT lineattr  EMPTY>
<!ATTLIST lineattr  colour     %cols;     #IMPLIED
            thick      %number;   #IMPLIED
```

```
            type      %lintyps;  #IMPLIED>

<!ELEMENT list     ((with)?, id)>

<!ELEMENT marking  EMPTY>
<!ATTLIST marking  x        %number;   #IMPLIED
            y        %number;   #IMPLIED
            hidden %boolean;  #IMPLIED>

<!ELEMENT ml       (#PCDATA)>
<!ATTLIST ml       id       ID        #IMPLIED>

<!ELEMENT page     (pageattr, (trans | place | arc | Aux | vguideline | hguideline | group)*)>
<!ATTLIST page     id       ID        #IMPLIED>

<!ELEMENT pageattr EMPTY>
<!ATTLIST pageattr name     CDATA     #IMPLIED>

<!ELEMENT place    (%objatts;, text, ellipse, (token | marking | fusioninfo | port | type | initmark)*)>
<!ATTLIST place    id       ID        #IMPLIED>

<!ELEMENT placeend EMPTY>
<!ATTLIST placeend idref    IDREF     #IMPLIED>

<!ELEMENT port     (%objatts;)>
<!ATTLIST port     id       ID        #IMPLIED
            type     %prttyps;  #IMPLIED>

<!ELEMENT posattr  EMPTY>
<!ATTLIST posattr  x        %number;   #IMPLIED
            y        %number;   #IMPLIED>

<!ELEMENT product  (id)+>

<!ELEMENT real     (with)?>

<!ELEMENT record   (recordfield)+>

<!ELEMENT recordfield (id,id)>

<!ELEMENT string   (with)?>

<!ELEMENT subset   (id?, (with | by))>

<!ELEMENT subst    EMPTY>
<!ATTLIST subst    subpage  IDREF     #IMPLIED
            portsock CDATA     #IMPLIED>

<!ELEMENT text     (#PCDATA)>

<!ELEMENT textattr EMPTY>
<!ATTLIST textattr colour   %cols;    #IMPLIED
            bold     %boolean;  #IMPLIED>

<!ELEMENT time     (%objatts;, text)>
<!ATTLIST time     id       ID        #IMPLIED>

<!ELEMENT timed    EMPTY>

<!ELEMENT token    EMPTY>
<!ATTLIST token    x        %number;   #IMPLIED
            y        %number;   #IMPLIED>
```

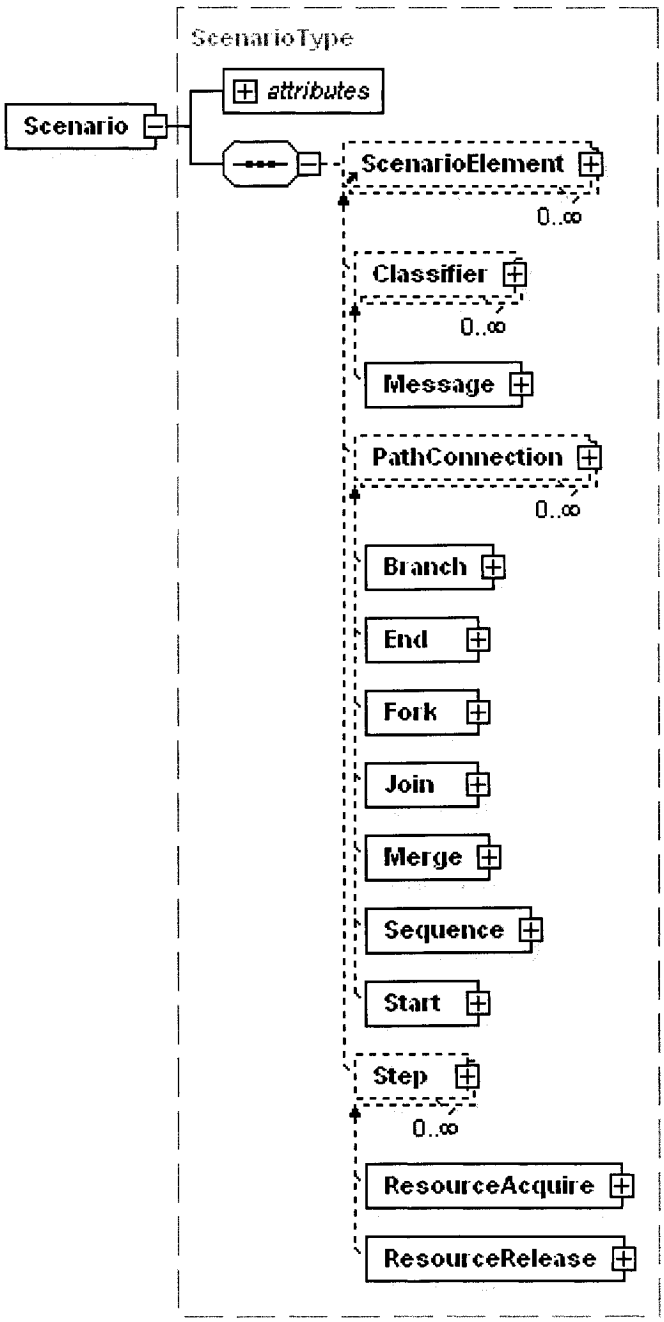
```
<!ELEMENT trans     (%objatts;, text, box, subst?, (time | cond | code-key | code)*)>
<!ATTLIST trans     id          ID          #IMPLIED>

<!ELEMENT transend  EMPTY>
<!ATTLIST transend  idref       IDREF       #IMPLIED>

<!ELEMENT type      (id | (%objatts;, text))>
<!ATTLIST type      id          ID          #IMPLIED>

<!ELEMENT union     (unionfield)+>

<!ELEMENT unionfield (id, (type)?)>

<!ELEMENT unit      (with)?>

<!ELEMENT var       (type, id+)>
<!ATTLIST var       id          ID          #IMPLIED>

<!ELEMENT vguideline (guideline_elm)*>
<!ATTLIST vguideline id         ID          #IMPLIED
                x           %number;    #IMPLIED>

<!ELEMENT with      ((ml, (ml?|and))|(id,(id)?))>

<!ELEMENT workspaceElements (generator, cpnet)>
```

# C. Graphical Views of CSM Schema and Petri Nets DTD

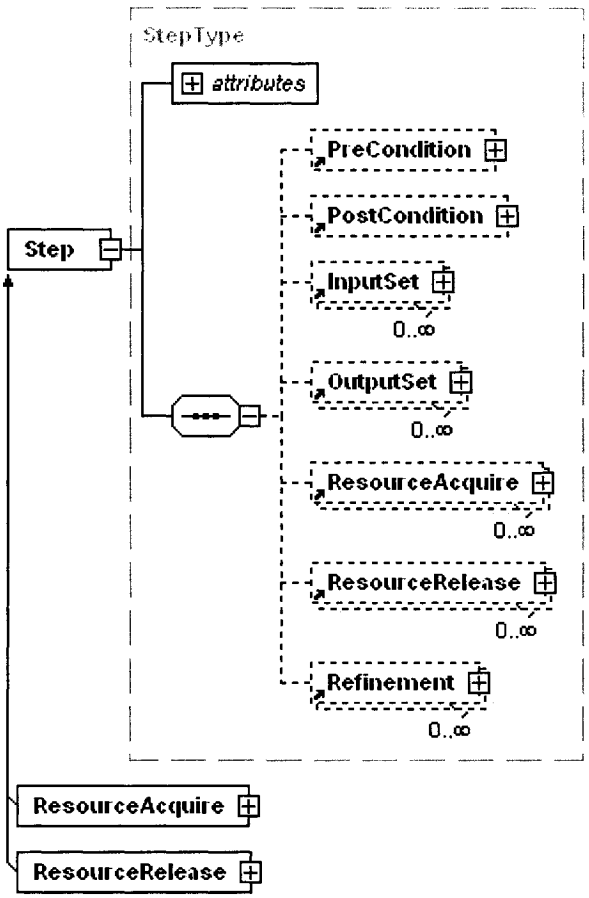
The Figures included in this appendix have been prepared to view the tree structure of XML Schema for CSM. To graphically view the Structure of DTD it was first converted to XML Schema using the tool XMLSpy. These diagrams have also been prepared with XMLSpy tool and show attributes and the elements contained with in the CSM Schema and Petri Net DTD at different levels.



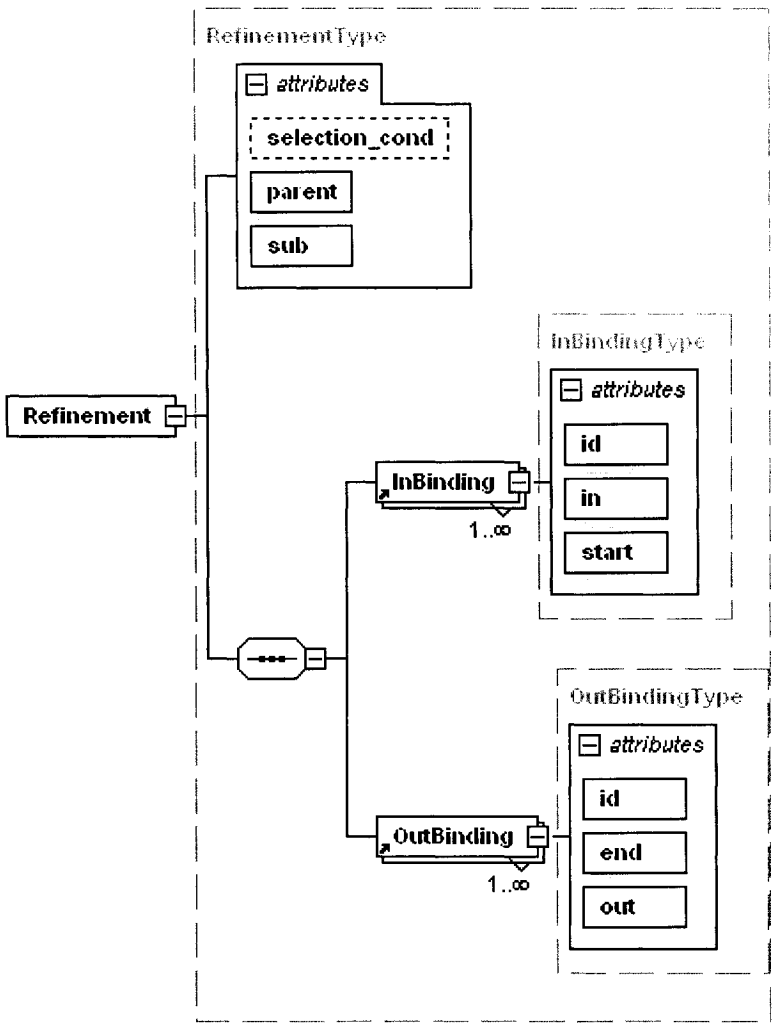

Graphical view of CSM tree at level 1 and 2



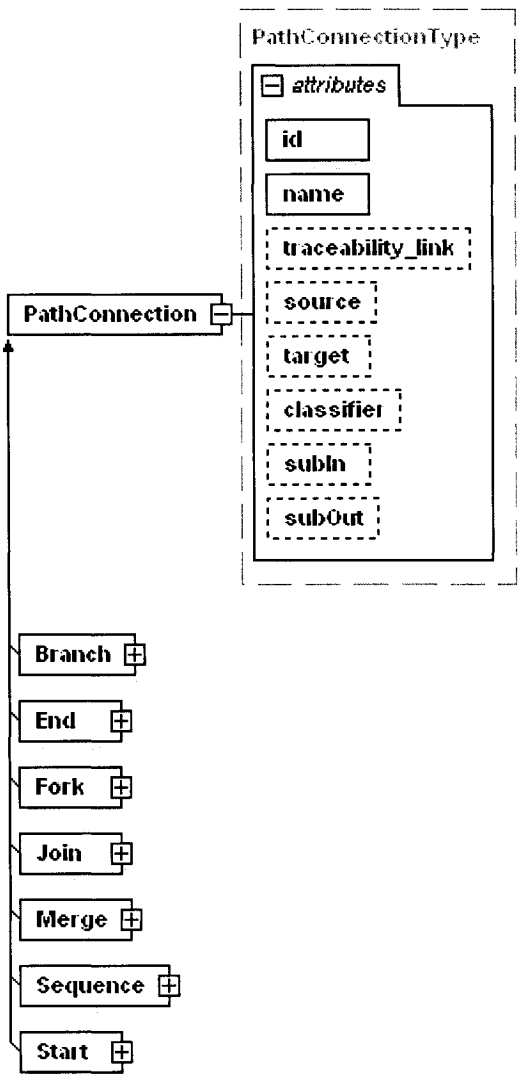

Graphical view of CSM tree at level 2 and 3



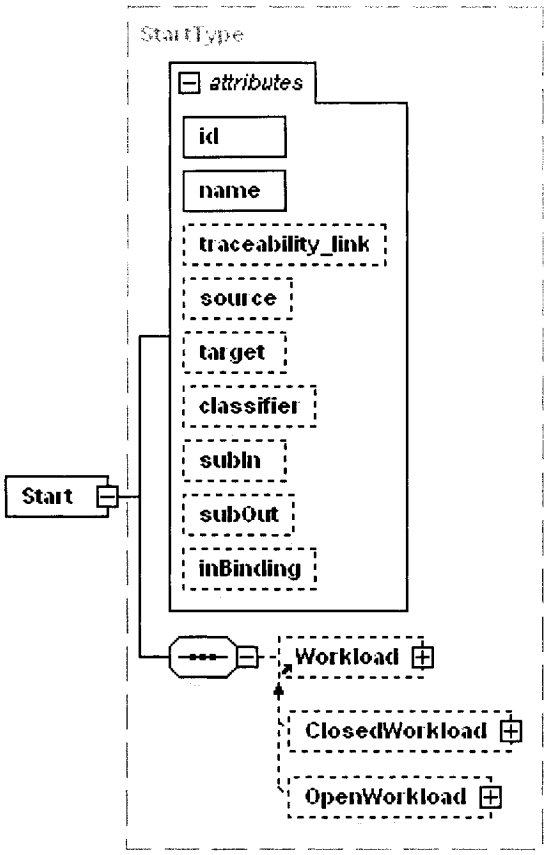

Graphical view of CSM tree at level 3 and 4



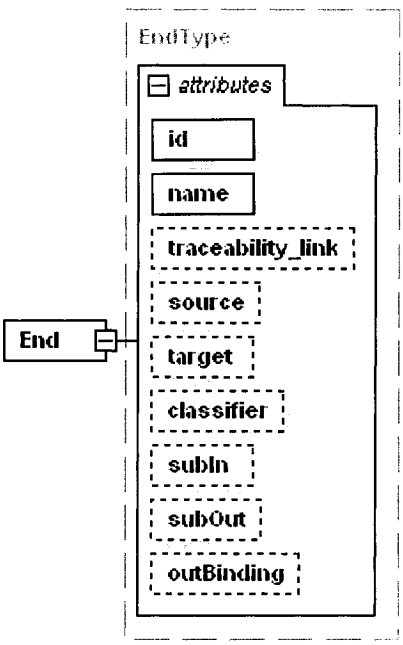

Graphical view of CSM tree at level 4 and 5



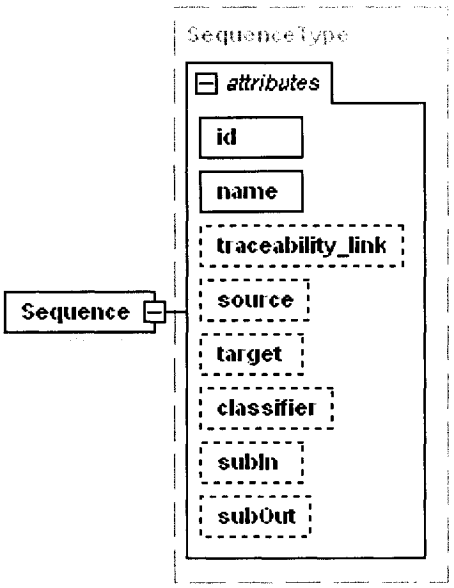

Path connections graphical view with attributes



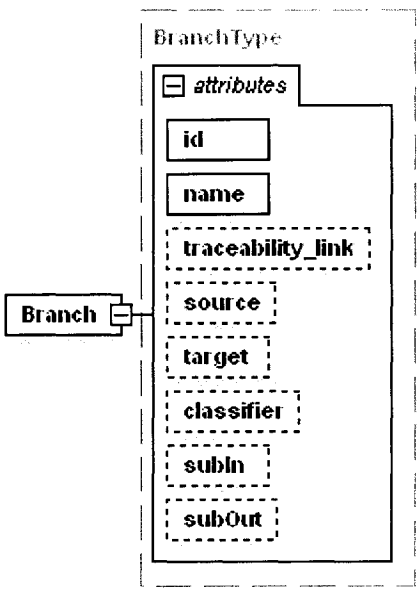

Starts graphical view with attributes



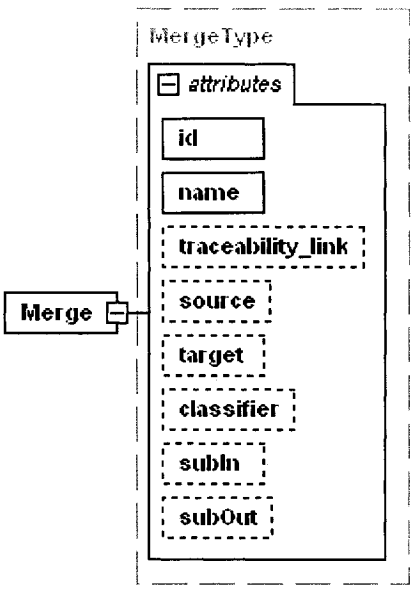

Ends graphical view with attributes



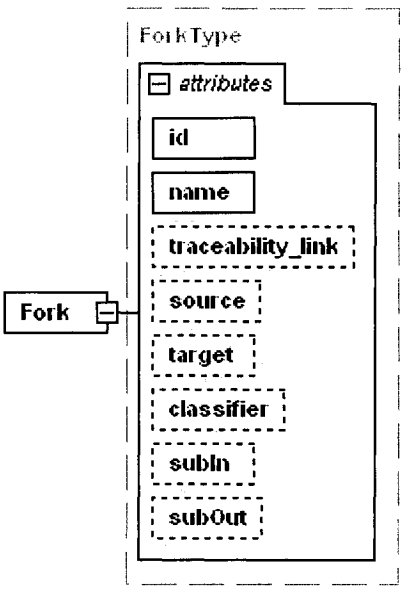

Sequence graphical view with attributes



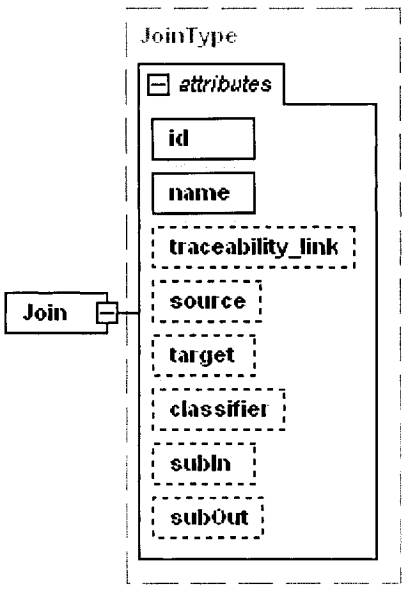

Branch graphical view with attributes




Merges graphical view with attributes

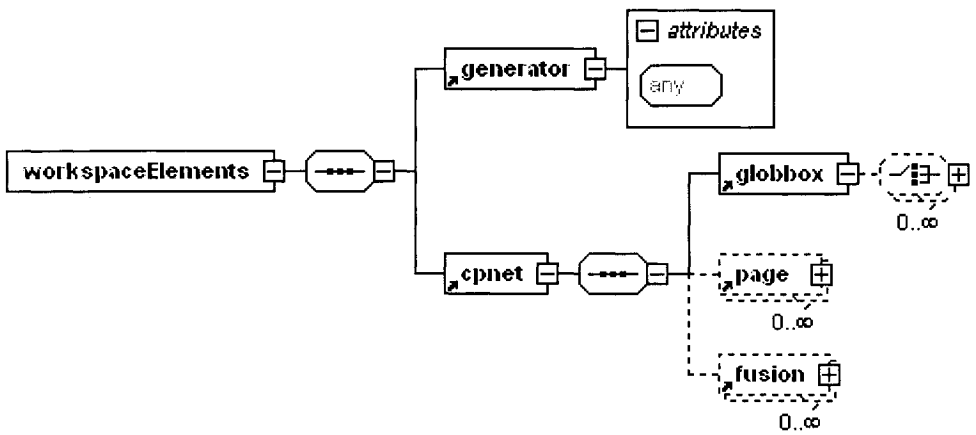



Forks graphical view with attributes

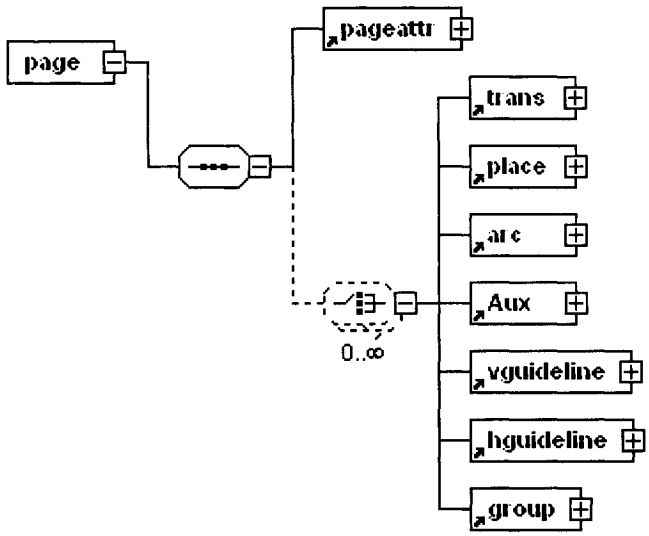



Joins graphical view with attributes

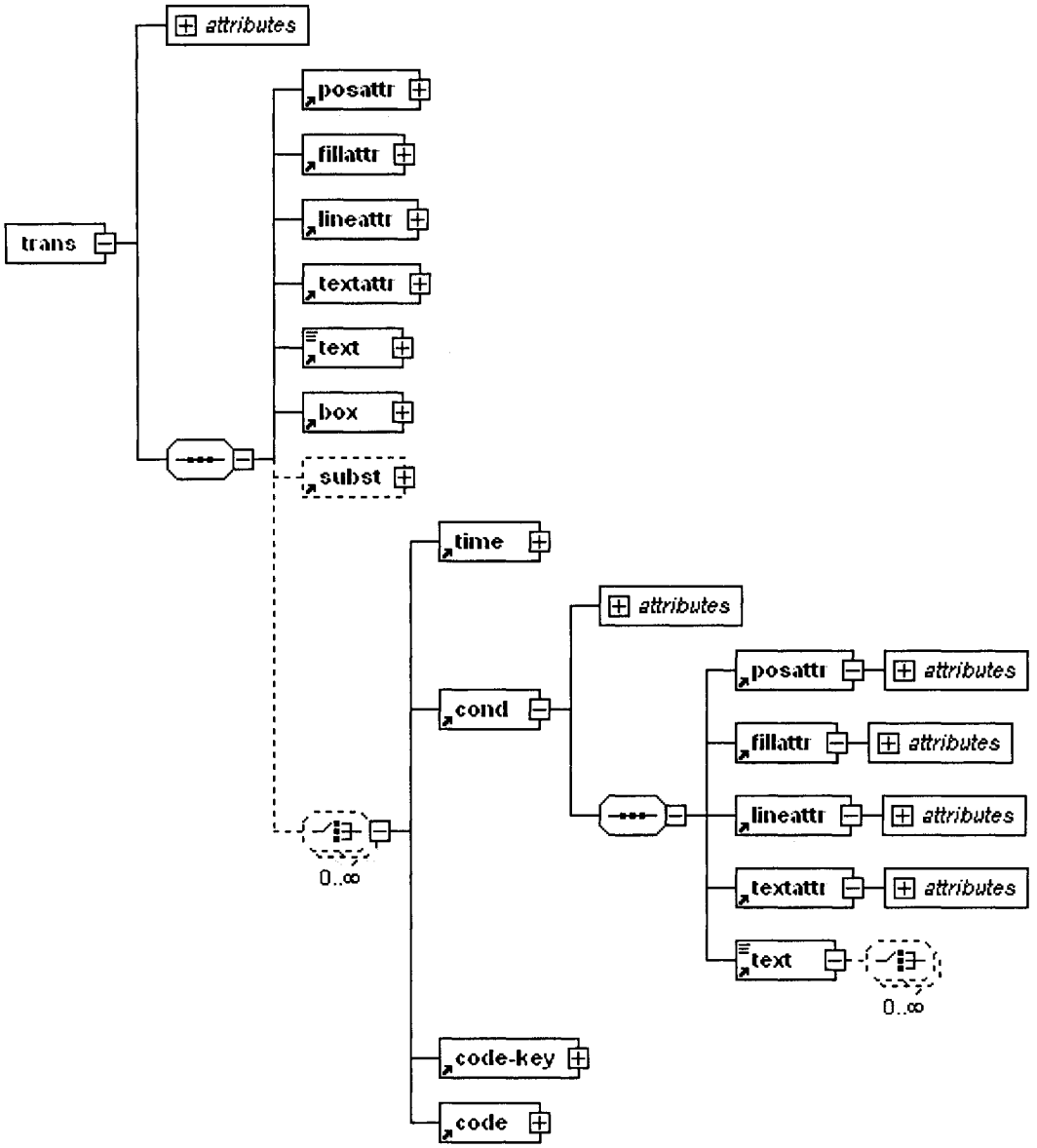
## Graphical Views of Petri Net DTD



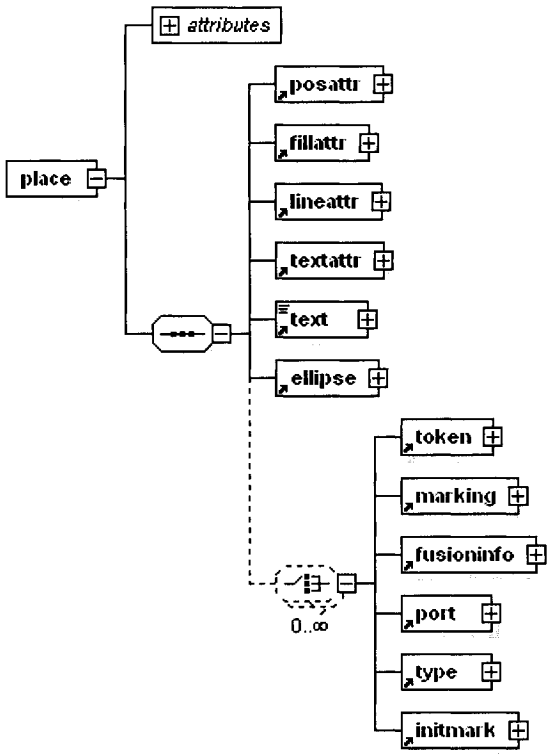

Graphical view of Petri Net tree at level 1, 2 and 3



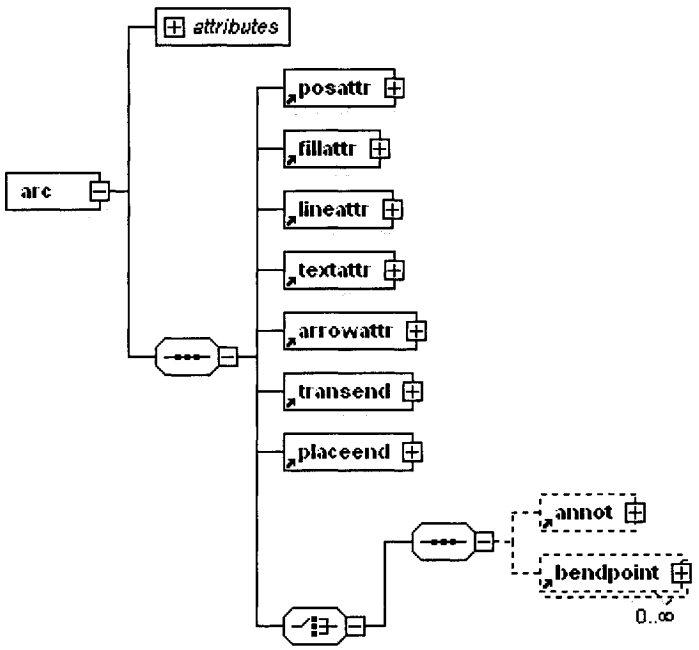

Graphical view of Petri Net tree at level 3 and 4




Partial graphical view of Petri Net tree at level 4, 5 and 6




Partial graphical view of Petri Net tree at level 4 and 5




Partial graphical view of Petri Net tree at level 4 and 5

## D. Petri Net representation of Transformed CSM discussed as Case Study 1


```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE workspaceElements PUBLIC "-//CPN/DTD CPNXML 1.0 //EN"
"http://www.daimi.au.dk/~cpntools/bin/DTD/2/cpn.dtd">

<workspaceElements>
<generator tool="CPN Tools" version="1.0.4" format="2"/>
<cpnet>
<globbox>
<block id="IDP1">
<id>Standard declarations</id>
<color id="IDP2">
<id>E</id>
<enum>
<id>e</id>
</enum>
</color>
<color id="IDP3">
<id>INT</id>
<int/>
</color>
<color id="IDP4">
<id>Bool</id>
<bool/>
</color>
<color id="IDP5">
<id>String</id>
<string/>
```

```
</color>
</block>
</globbox>

<page id="m0">
<pageattr name="New Page"/>

<place id="h0">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Request</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h1" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h0"/>
<placeend idref="h2"/>
</arc>

<trans id="h2">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
```

<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Sorig</text>
<box w="20.00" h="15.00"/>
<subst subpage="m1" portsock="(h1)(h3 h4)"/>
</trans>

<arc id="h4" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h2"/>
<placeend idref="h5"/>
</arc>

<place id="h5">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Notify</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h3" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>

<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h2"/>
<placeend idref="ID1"/>
</arc>

<place id="ID1">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>dummy</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="ID2" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="ID1"/>
<placeend idref="h6"/>
</arc>

<trans id="h6">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>

<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Sterm</text>
<box w="20.00" h="15.00"/>
<subst subpage="m2" portsock="(h3)(h7 h9 h13)"/>
</trans>

<arc id="h7" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h6"/>
<placeend idref="h8"/>
</arc>

<place id="h8">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Ring</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h9" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>

<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h6"/>
<placeend idref="ID3"/>
</arc>

<place id="ID3">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>dummy</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="ID4" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="ID3"/>
<placeend idref="h10"/>
</arc>

<trans id="h10">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>

```
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Fwd_sig</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="h11" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h10"/>
<placeend idref="h12"/>
</arc>

<place id="h12">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Busy</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h13" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
```

<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h6"/>
<placeend idref="ID5"/>
</arc>

<place id="ID5">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>dummy</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="ID6" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="ID5"/>
<placeend idref="h14"/>
</arc>

<trans id="h14">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>

```
<textattr colour="Black" bold="false"/>
<text>Fwd_sig</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="h15" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h14"/>
<placeend idref="h16"/>
</arc>

<place id="h16">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Ringing</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

</page>

<page id="m1">
<pageattr name="New Page"/>
```

```
<place id="h17">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Start</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h18" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h17"/>
<placeend idref="h19"/>
</arc>

<trans id="h19">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Check OCS</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="h20" orientation="TtoP">
```

<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h19"/>
<placeend idref="h21"/>
</arc>

<place id="h21">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text> </text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h22" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h21"/>
<placeend idref="h23"/>
</arc>

<trans id="h23">

<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Deny</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="h24" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h23"/>
<placeend idref="h25"/>
</arc>

<place id="h25">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Fail</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h26" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>

```
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h21"/>
<placeend idref="ID7"/>
</arc>

<trans id="ID7">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>dummy</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="ID8" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="ID7"/>
<placeend idref="h27"/>
</arc>

<place id="h27">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
```

```
<textattr colour="Black" bold="false"/>
<text>Success</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

</page>

<page id="m2">
<pageattr name="New Page"/>

<place id="h28">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Start</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h29" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h28"/>
<placeend idref="ID9"/>
```

```
</arc>

<trans id="ID9">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>dummy</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="ID10" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="ID9"/>
<placeend idref="h30"/>
</arc>

<place id="h30">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text> </text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>
```

```
<arc id="h32" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h30"/>
<placeend idref="h33"/>
</arc>

<trans id="h33">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Busy Treatment</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="h34" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h33"/>
<placeend idref="h35"/>
</arc>

<place id="h35">
```

```
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Fail</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h31" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h30"/>
<placeend idref="h36"/>
</arc>

<trans id="h36">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Step</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="h38" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
```

<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h36"/>
<placeend idref="ID11"/>
</arc>

<place id="ID11">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>dummy</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="ID12" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="ID11"/>
<placeend idref="h39"/>
</arc>

<trans id="h39">
<posattr x="0.00" y="0.00"/>

<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Ringing Treatment</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="h40" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h39"/>
<placeend idref="h41"/>
</arc>

<place id="h41">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Report Success</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="h37" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>

<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h36"/>
<placeend idref="ID13"/>
</arc>

<place id="ID13">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>dummy</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

<arc id="ID14" orientation="PtoT">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="ID13"/>
<placeend idref="h42"/>
</arc>

<trans id="h42">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>

<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Ring Treatment</text>
<box w="20.00" h="15.00"/>
</trans>

<arc id="h43" orientation="TtoP">
<posattr x="0.000000" y="0.000000"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<arrowattr headsize="1.200000" currentcyckle="2"/>
<transend idref="h42"/>
<placeend idref="h44"/>
</arc>

<place id="h44">
<posattr x="0.00" y="0.00"/>
<fillattr colour="White" pattern="" filled="false"/>
<lineattr colour="Black" thick="1" type="Solid"/>
<textattr colour="Black" bold="false"/>
<text>Success</text>
<ellipse w="20.00" h="15.00"/>
<token x="0.00000" y="0.00000"/>
<marking x="0.00000" y="0.00000" hidden="false"/>
</place>

</page>
</cpnet>
</workspaceElements>

# References

[1] Amyot D, Specification and validation of telecommunication systems with use case maps and lotos; http://lotos.site.uottawa.ca/ftp/pub/Lotos/Theses/da_phd.pdf

[2] Amyot D, Use Case Maps Quick Tutorial, See http://www.usecasemaps.org/pub/UCMtutorial/UCMtutorial.pdf

[3] Bock, C., UML 2 Activity and Action Models, in Journal of Object Technology, vol. 2, no. 4, July-August 2003, pp. 43-53.

[4] Bock, C., UML Without Pictures, to appear in IEEE Software Special Issue on Model-driven Development, September/October 2003.

[5] Bock, C., UML 2 Activity and Action Models, Part 2: Actions, in Journal of Object Technology, vol. 2, no. 5, September-October 2003, pp. 41-56. http://www.jot.fm/issues/issue_2003_09/column4

[6] Bock, C., UML 2 Activity and Action Models, Part 3: Control nodes, in Journal of Object Technology, vol. 2, no. 6, November-December 2003, http://www.jot.fm/issues/issue_2003_11/column1

[7] Bock, C., UML 2 Activity and Action Models, Part 4: Object nodes, in Journal of Object Technology, vol. 3, no. 1, January-February 2004, http://www.jot.fm/issues/issue_2004_01/column3

[8] Bock, C., UML 2 Activity and Action Models, Part 5: Object nodes, in Journal of Object Technology, vol. 3, no. 7, July-August 2004, http://www.jot.fm/issues/issue_2004_07/column4

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

[10] Booch, G., Rumbaugh, J. & Jacobson, I., The Unified modeling language user guide, Reading, Mass., Addison-Wesley. 1999.

[11] Buhr, R.J.A. (1998) "Use Case Maps as Architectural Entities for Complex Systems". In: IEEE Transactions on Software Engineering, Special Issue on Scenario Management. Vol. 24, No. 12, December 1998, 1131-1155. http://www.UseCaseMaps.org/pub/tse98final.pdf

[12] CPN tools at University of Aarhus, http://wiki.daimi.au.dk/cpntools/cpntools.wiki

[13] C. U. Smith and L. G. Williams, Performance Solutions. Addison-Wesley, 2002.

[14] Dorin B. Petriu, Murray Woodside, A Metamodel for Generating Performance models from UML Designs; 7th International Conference on Unified Modeling Language, Lisbon, Portugal, 2004, pp. 41-53

[15] D.C. Petriu and H. Shen, Applying the UML Performance Profile: Graph Grammar-based derivation of LQN models from UML specifications, in Proc. 12th Int. Conf. on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation, London, England, 2002.

[16] Garrido Jose Luis, Gea Miguel. 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.

[17] Gregor v. Bochmann, Activity Nets; A UML profile for modeling workflow and business processes: http://beethoven.site.uottawa.ca/dsrg/PublicDocuments/Publications/Boch00d.pdf.

[18] HyperModel: http://update.xmlmodeling.com/updates/

[19] http://www.daimi.au.dk/CPnets/intro/verybrief.html

[20] Juan Pablo Lopez-Grao, Jose Merseguer, Javier Campos, From UML Activity Diagrams To Stochastic Petri Nets:Application To Software Performance Engineering; In: Proceedings of the Fourth International Workshop on Software and Performance (WOSP'04), ACM, Redwood City, California, USA, pages 25-36. January 2004.

[21] Kurt Jensen, A Brief Introduction to Coloured Petri Nets , Computer Science Department, University of Aarhus , http://www.daimi.aau.dk/~kjensen/

[22] MDA Guide Version 1.0.1 http://www.omg.org/docs/omg/03-06-01.pdf

[23] Miga, A. (1998) Application of Use Case Maps to System Design with Tool Support. M.Eng. thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada. http://www.UseCaseMaps.org/pub/am_thesis.pdf

[24] Murray Woodside A Metamodel for Generating Performance Models from UML Designs

[25] Nicholas Chase, Understanding DOM, July 2003, http://www-106.ibm.com/developerworks/edu/x-dw-xudom-i.html

[26] Object Management Group, UML 2.0 Superstructure Specification, http://www.omg.org/cgi-bin/doc?ptc/03-08-02, August 2003.

[27] Object Management Group, UML Profile for Schedulability, Performance, and Time Specification, OMG Adopted Specification ptc/02-03-02, July 1, 2002

[28] Object Management Group, XML Metadata Interchange Specification, January 2003; http://www.omg.org/technology/documents/formal/xmi.htm.

[29] Quatrani, T., Visual modeling with rational rose and UML, Reading Mass., Addison-Wesley. 1998.

[30] Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise, Introduction to Model Driven Architecture: http://www.informit.com/articles/article.asp?p=170798&rl=1

[31] Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise: MDA Distilled, Principles of model drive architecture, 2004.

[32] Toqueer Israr: Master's Thesis under preparation at Carleton University Ottawa.

[33] W3 Consortium (1998) Extensible Markup Language (XML) 1.0. W3C Recommendation, 10 February 1998. http://www.w3.org/TR/REC-xml

[34] W. Reisig, 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.

[35] W.M.P. van der Aalst, The Application of Petri Nets to Workflow Management, in the Journal of Circuits, Systems and Computers, 8(1):21-66, 1998.

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

[37] Yong Xiang Zeng, Transforming Use Case Maps to the Core Scenario Model Representation; http://lotos.site.uottawa.ca/ftp/pub/Lotos/Theses/