# Automatic Generation of Core Scenario Models from Execution Traces

By

**Ge Guo**

A thesis submitted to the Faculty of Graduate Studies

in partial fulfillment of the requirement

for the degree of

**Master of Science in Information and Systems Science**

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

January 18, 2008

*Your file Votre référence*
*ISBN: 978-0-494-40658-8*
*Our file Notre référence*
*ISBN: 978-0-494-40658-8*

# Abstract

Though performance models can reduce the possibility of performance-related failures, they are seldom used during the design of software because of the considerable effort needed to construct them. This thesis presents a model building technique that extracts CSM (Core Scenario Model) from execution traces. CSM is an intermediate model that captures the essence of performance specification and estimation, and can be transformed to a performance model. The transformation procedure from execution traces to CSM is to extract or deduce instances of CSM elements from execution traces or instances of other CSM elements. This technique is appropriate for a messages passing distributed system where tasks interact through point-to-point communication. It will allow performance analysts to focus on the principles of software performance analysis rather than model building.

# Acknowledgement

I would like to thank my supervisor Professor Gregory Franks for the opportunity to study at Carleton University, and thank for his patience, intelligence, comments and suggestions.

To my beloved wife, Lin, thank you for all you did for our family and for me. You always comfort me when I am frustrated, encourage me when I lose hope, and be happy when I am happy.

My son, Daniel, what a wonderful boy you are! I have to appreciate him the most. Nobody in the whole world can give me the joy, love, smiles, cries, peace, optimism and all others that he gives to me.

I never knew that what kind of love parents can give to their children until I have my own. Therefore, I have to thank my mom for all the supports she gave and will give to me. I will say thanks to my dad in heaven. Life was not easy after you left us many years ago, but it let me know how strong I am.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1: Introduction

## *1.1 Motivation*

Two different requirements, functional and non-functional requirements, are emphasized during the software development lifecycle. The former describes functions that the system executes, while the latter acts to constrain the solution, such as performance requirements, maintainability requirements, safety requirements, reliability requirements, and so on.

Software performance engineering can be performed late in the development cycle, when the system under development can be run and measured [Barber]. However, it can be costly to perform when applied to a large system. Also two obstacles, lack of theoretical justification and conflict between automation and adaptability, have been identified in particular to preclude the adoption of this method [Malony]. The modeling approach can explore larger range of input parameters with a lower cost. Moreover, it can explain results systematically and locate performance failures accurately.

Software performance engineering (SPE), firstly introduced in [Smith 90], is a systematic, quantitative approach to construct software systems that meet performance objectives. Early software performance analysis often involves the construction of predictive performance models from software requirements and specifications, the evaluation of the performance models using different solvers, and the generation of relevant performance results back to software developers. It evaluates the performance effects of different design and implementation alternatives throughout the software lifecycle, from requirements implementations to maintenance. The basic concept is to build a performance model, which captures the performance aspects of software behavior, and is to be solved later to obtain

performance results, such as response times, throughput, utilization of each component, and so on. The performance problems thus can be detected and alternative solutions for eliminating them can be assessed in a similar way.

Though performance models can reduce the possibility of performance-related failures, they are seldom used during the design of software, especially for distributed and concurrent software systems, which always require more attentions to performance concerns than other systems. The reason is SPE requires knowledgeable and trained people to implement it. Since software developers are not performance analysts, major efforts are still required to integrate software performance analysis into the ordinary software development process. Therefore, different techniques that target the construction of performance model automatically are developed to ease the burden. Among them are the trace-based model construction techniques that generate performance models from execution traces automatically, and model transformation techniques that transform design models to performance models.

Polymorphism, inheritance, late binding and other object-oriented features make it very difficult to capture performance parameters, so we use trace-based model construction techniques because they can provide more dynamic details related to performance issues in object-based developing environments. The target models of the two trace-based model construction techniques mentioned above, that is, TLC in [Hrischuk 99] and SAME in [Tauseef 05], are both Layered Queueing Network (LQN) models [Woodside 95], though they can be tailored to other performance models. The processes are very similar, that is, obtaining execution traces, extracting communication patterns, generating LQN sub – models, and then generating LQN model. The main difference is the format of input execution traces, and the process of extracting communication patterns. This means

application ranges are different for TLC and SAME, and they use different intermediate models to generate communication patterns.

The disadvantage of TLC is that the algorithm it adopts is based on graph grammar and graph rewriting, which depend on special tools and are costly in term of space and time. The disadvantage of SAME is that it is very tightly bound to LQN models because basically its algorithm is to detect three communication types crucial to LQN, thus it may need significant efforts to modify SAME for other target models.

The transformation from design to performance model derives a model of software execution patterns from the design, and then from this it constructs the performance model. This can provide earlier warnings about design problems, and since the design decisions can be easily revised, making adjustments more effective. Because there are various types of design and performance models, many different transformations have been developed individually. Therefore, intermediate models, such as Core Scenario Model (CSM), have been developed to reduce the number of transformations needed from N*M to only N+M. Another potential advantage is that they could possibly be simpler to devise than the direct transformations because the intermediate model is likely to be closer to both the source and target models.

## *1.2 Contributions*

The thesis makes the following contributions:

1. Define algorithms for transforming execution traces to CSM models, and then CSM models can be converted to LQN.
2. Extend conversion to find service time.
3. The implementation of transformation algorithms in Java programming language.

4. Validation of transformation algorithms.

## *1.3 Thesis Outline*

This thesis is organized as follows. Chapter 2 describes the background of performance model transformation techniques, the use of trace-based techniques and CSM model. Chapter 3 presents the transformation process from execution traces to CSM models. Chapter 4 discusses the validation of the transformation tool and chapter 5 is a case study. Chapter 6 gives the conclusion and future work.

# Chapter 2: Background

## *2.1 Obstacles to Developing Performance models*

There are several reasons that make the construction of performance models very difficult. First, for most performance models the effort needed to construct them is significant: A small set of key performance scenarios representative of the way in which the system will be used should be identified first; and then by following the execution path of each scenario, the quantitative demands for resources made by each component, and reasons for queueing delays should be discovered too; and finally they need to be mapped onto a performance model. Second, because some performance models are always tightly related to specific design tools, when new tools appear, new performance models and new solvers have to be developed, however, developing a solver for a certain type model is not easy either. This makes the use of performance models more unattractive.

For the first problem, automatic model building techniques will be a solution. At present, at least two such techniques have been developed, both of them extract performance models from design: one is from design itself, such as from a Unified Modeling Language (UML) design model annotated with performance information, this is known as a model transformation (from design model to performance model); the other one is from trace data, which are recorded during the execution of some executable forms of design, such as traces produced by ObjectTime, a design prototype environment, or a simplified version. Though the inputs for these two methods are different, both of them develop performance models in an automatic way, which combines software design and performance engineering, and may involve some intermediate models.

For the second problem, because developing a new solver or new tool for a new type performance model or design model takes a long time, we can also use model transformation techniques as a temporary solution (from a new design model to another old design model, or from a new design model to an old performance model). That is to say, first map the new design models to the old design models, or old performance models directly, and then use corresponding model constructors or/and solvers to solve. Since in most situations, performance models are much simpler than design models, the latter way is adopted commonly.

Therefore, for both problems model building and solver developing, model transformation techniques are very helpful. Figure 2-1 shows the relationship among different models that we mentioned above. The solid lines mean the relationships have been implemented for at lease one instance; the dashed lines mean the relationships may be implemented in the future; the solid lines with double arrow means the items are tightly related. For example, line (1) (with an arrow pointing to performance models) between design and performance domain indicates that we can transform design models to performance models though this transformation may not be necessary, since we can build performance models from other resources, such as software specifications as line (2) indicates. Line (3) (with arrow at each end) between a performance model and its solver means we need certain solvers for certain performance models. Line (4) (with an arrow pointing to trace data) between design models and trace data shows that the latter is generated from the former. Line (5) (with an arrow pointing to design models) between trace data and design models means we can create design models from trace data in the future, if we can obtain sufficient information from the latter. Transformations between design models are also possible, such as the lines between design model A and B. So far, different techniques have been developed for the transformations involved in figure 2-1, which will be shown in the following sections.

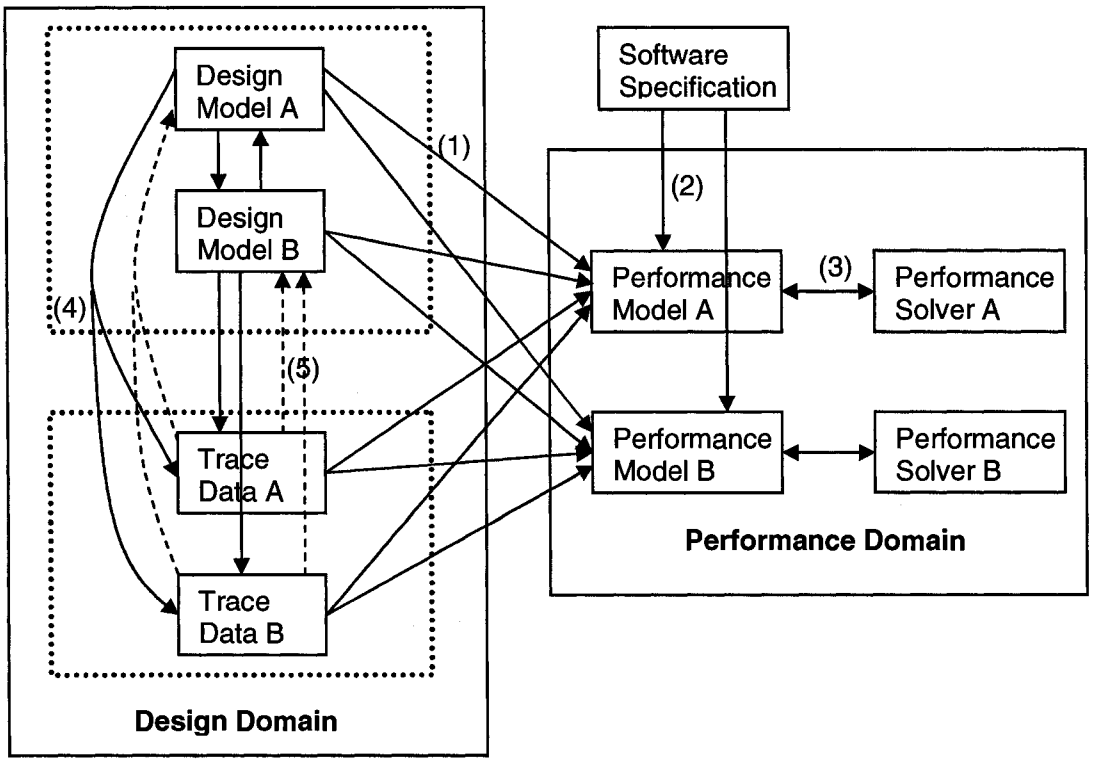



**Figure 2-1 Model Transformations Involved in Design and Performance Domains**

## *2.2 Transformation from Design to Performance Model*

The transformation from design to performance model is called "predictive approach", since it purely derives the performance model from the design. This approach, as described by Smith in [Smith 90] and [Smith 93], derives a model of software execution patterns (or execution graph) from the design, and then from this model (intermediate model) it constructs the performance model to solve for performance predictions which can be used to guide the modification of the design. This can give earlier warnings about design problems, and since the design decisions can be easily revised, this makes adjustments more cost effective.

In Figure 2-1, line (1) is such a transformation. Because there are various types of performance models, which include queueing networks (QN) [Lazowska] and their extensions called Extended Queueing Networks (EQN) [Williams 98] and Layered Queueing Networks (LQN) [Woodside 95], Stochastic Timed Petri nets (STPN) [King 99] or Generalized Stochastic Petri nets (GSPN) [Marsan], Stochastic Process Algebras (SPA) [Pooley 99] and simulation models [Balsamo 03], transformations from different design models, such as UML activity, sequence, and state machine diagrams and Use Case Maps (UCM) [Buhr 96] [Buhr 98], to these performance models have been developed individually. We will briefly describe some implemented transformations with the focus on the one from UML to LQN.

### 2.2.1 Transformation from UML Models to Queueing Networks

This transformation is based on the performance engineering methodology [Smith 90], and consists of two parts: the Software Model (SM) and the Machinery Model (MM), i.e., the hardware platform model. The former captures essential aspects of software behavior and is based on Execution Graphs (EG), a graph whose nodes represent software components that perform a specific task and whose edges represent transfer of control. And the latter is the model of the hardware platform and is based on Extended Queueing Network Models (EQNM). In order to specify an EQNM, it is necessary to define components and the connections among them, which can be obtained from system description, and parameters including job classes, scheduling discipline at service centers / components, service demands and so on, which, have to be derived from SM and knowledge of resource capabilities.

The transformation methodology proposed in [Cortellesa] makes use of the UML Use Case Diagram, the Sequence Diagram, and the Deployment Diagram, which are used to derive

the user/actor profile and execution scenarios, execution graphs, and an EQNM. The Deployment Diagram is also used to identify the deployment of software components on hardware so as to improve the accuracy of the performance model. There are four steps to performance the transformation:

1. Deduce from Use Case Diagram the user profile.
2. For each use case, process the corresponding set of Sequence Diagrams to obtain the meta-EG, which represents a labeled EG that is not tailored to any specific hardware platform.
3. Use its Deployment Diagram to obtain EQNM of the hardware platform and to appropriately tailor the meta-EG to the obtain EG-instance.
4. Assign numerical parameters to the EG-instance.
5. Combine EG-instance and EQNM to solve the obtained performance model by using the SPE approach.

A similar method with the emphasis on multiclass application is proposed in [Balsamo 05], which differentiates the algorithm for open workloads and closed workloads, and needs the support of UML Use Case, Sequence, and Deployment Diagrams annotated with SPT [OMG SPT] information.

### 2.2.2 Transformation from UML Diagram to Stochastic Petri Nets

Petri nets (PN) [Murata 89] are a formal and graphical language, appropriate for modeling systems with concurrency. It graphically depicts the structure of a distributed system as a directed bipartite graph with annotations. Generally, a basic Petri net consist of place nodes, transition nodes, and directed arcs connecting them but not connecting place nodes and place nodes, or transition nodes and transition nodes. The place nodes from which an arc

runs to a transition not are called the input places of the transition; the place nodes to which arcs run from a transition are called the output places of the transition.

Places may contain any number of tokens. A distribution of tokens over the places of a net is called a marking. Transitions act on input tokens by a process known as firing. A transition is enabled if it can fire, i.e., there are tokens in every input place. When a transition fires, it consumes the tokens from its input places, performs some processing task, and places a specified number of tokens into each of its output places. It does this atomically, i.e., in one non-interruptible step.

### *Transformation from UML 1.4 Statechart and Collaboration Diagrams to Colored Petri Nets*

Colored Petri nets (CPNs) [Jensen 92], final target models in [Saldhana] when transforming UML Diagram to PN, are a generalization of ordinary PNs, allowing convenient definition and manipulation of data values. The transformation needs an intermediate model, Object Petri Net Models (OPM) [Lakos 94] that extensively adopts object oriented structuring into Petri nets. This relies on objects' event-based behaviors in relation to other objects in the environment and this behavior can be modeled as a CPN.

The transformation methodology makes use of the UML 1.4 Statechart Diagrams and Collaboration Diagrams. As shown in Figure 2-2, Statechart diagrams are first converted to flat state machines. These state machines are then converted to a form of OPM. Then the UML collaboration diagrams are used to connect these OPM models to derive a single CPN. The transformation rules are:

1. The actions that are part of the transitions in the Statechart Diagrams are mapped onto event generators, which are associated with states and are functions that generate event tokens.
2. Guard conditions are mapped, by the environment or the object in question, onto an event token that is created when the condition becomes true.
3. The entry and exit actions of a state can be mapped onto internal events that are generated before the object enters the state and after it leaves the state, respectively
4. Composite states may involve sequential or concurrent substates.

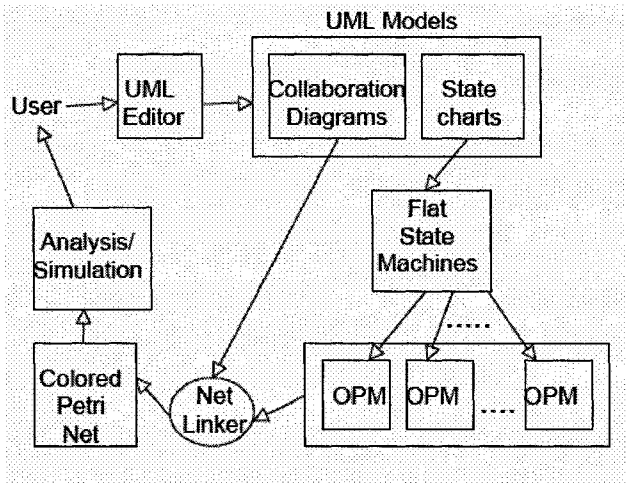



**Figure 2-2 Transformation from UML 1.4 Statechart and Collaboration Diagrams to Petri Net**

### *Transformation from UML Sequence Diagrams to Petri Nets*

[Ouardani] deals with the transformation of UML Sequence Diagrams into Petri Nets. The technique primarily depends on the type of the messages that link up the objects to each other. For example, a message can be used to express a pure data transmission, or a method call either with or without data transmission. On the other hand, a message can wait or not wait an acknowledgement. This leads to translate synchronous (highly synchronous communication, equivalent to the remote procedure call RPC, and loosely synchronous communication, equivalent to RPC with a second phase at receiver) or asynchronous

communications into Petri nets,. The graphic transformation rules on meta-model level are defined in [Ouardani] too.

### *Transformation from UML Activity Diagrams to Generalized Stochastic Petri nets*

Generalized Stochastic Petri nets (GSPN) [Marsan] categorize transitions in Petri nets into two different classes: immediate transitions and timed transitions. Immediate transitions fire in zero time once they are enabled. Timed transitions fire after a random, exponentially distributed time enable once. Firing rates are associated only with timed transitions. GSPN are successfully applied to the performance analysis of a variety of systems whose main characteristics include concurrency and synchronization. [Lo'pez-Grao] extends GSPN to a Labeled Generalized Stochastic Petri net (LGSPN), where both places and transitions can be labeled, and the same label can be assigned to place(s) and to transition(s).

A compost LGSPN that represents a performance model is the target model of the transformation [Lo'pez-Grao] from UML Activity Diagrams. The translation of each element in an Activity Diagram can be summarized as a three-phased process:

1. Translation of each outgoing and self-loop transition. Applicable to action, subactivity and call states, and to fork pseudostates. Depending on the kind of transition, a different rule must be applied.
2. Composition of the LGSPNs corresponding to the whole set of each kind of transitions considered in step 1. Applicable to action, subactivity and call states, and to fork pseudostates.
3. Working out the LGSPN for the element by superposition of the LGSPNs obtained in the last step (if any) and, occasionally, an additional LGSPN corresponding to the

entry to its associated state (the so-called 'basic' subnets for subactivity states and fork pseudostates.

### *Transformation from UML Sequence and Statechart Diagrams to LGSPN*

The same research group as in [Marsan] developed another methodology [Bernardi] that transfers UML Sequence and Statechart diagrams into LGSPNs. A central contribution of this methodology is to establish relationships between Sequence Diagrams and Statechart Diagrams according to the UML metamodel, that allows defining a rule for Statechart and Sequence Diagrams composition that has its root in the UML metamodel and that produces a single executable model. The approach taken for the translation from Sequence Diagrams to a LGSPN model consists of the following steps:

1. Model each message of a Sequence Diagram with a LGSPN subsystem.
2. Compose the LGSPN subsystems obtained in the previous step to take into account the partial order relation that exists between messages belonging to the same interaction. In a certain sense this operation will consist in "connecting" those subsystems modeling messages which are in precedence relation.
3. Define the initial marking on the resulting LGSPN representing the Sequence Diagram.

The LGSPN obtained in this way will be used in conjunction with a previously defined translation of Statecharts into a GSPN to provide a complete model of the system behavior.

## 2.2.3 Transformation from UML Diagram to Process Algebras

Process algebras are mathematical theories which model concurrent systems by their algebra and provide apparatus for reasoning about the structure and behavior of the model. While the variety of existing process algebras is very large, there are several features that all process algebras have in common: a system is characterized by its active components and the interactions, or communications, between them. Unlike queueing networks or Petri nets

there is no notion of entity or flow within a model, instead, it describes processes and systems using a small collection of primitives, and operators for combining those primitives. Compositional reasoning is an integral part of the language, defining algebraic laws for the process operators, which allow process expressions to be manipulated using equational reasoning.

Stochastic Process Algebras (SPA) [Hillston 98] [Herzog] is an extension of process algebras which add quantification on its model, making it suitable for performance modeling. The extension consists of associating a random variable, representing time duration, with every action. The Performance Evaluation Process Algebra (PEPA) [Hillston 96] is an algebraic description technique based on SPA and enhanced with timing information. This extension results in models which may be used to calculate performance measures as well as deduce functional properties of the system. The basic elements of PEPA are components and activities. Each activity is represented by two pieces of information: the label, or action type, which identifies it, and the activity rate which is the parameter of the negative exponential distribution determining its duration.

### *Transformation from UML1.4 Collaboration and Statechart Diagrams to PEPA*

Generation of a process algebra model in [Pooley 99] from the UML model concentrates on the combined collaboration and statechart diagrams. The paper explains the methodology by transforming a small producer-consumer UML model to a PEPA model. The process begins by constructing a PEPA model of each object. The object Producer has two states and each of them is modeled as an agent, called Producer and Blocked respectively. The transition from Producer to Blocked requires the completion of the associated action and time, which can be used to generate a related rate for PEPA. In the same way, all the

transitions with rate of all the objects are identified. That is to say, all the activities, basic elements, for PEPA and their parameters are identified. In order to complete the model, separate processes must be combined using the PEPA cooperation primitive.

The method for the same transformation in [Canevet 03] needs the support by a tool set which comprises some existing modeling tools (ArgoUML and the PEPA Workbench), and other translators which are used to connect the Extractor and the Reflector. The extractor is used to extract PEPA models from UML statechart diagrams and collaboration diagrams; the algorithm for generating definitions of sequential components from state machines is relatively simple and involves traversing the transitions of the state diagrams and accumulating behaviours which are presented as choices in the definition of the component. The reflector is basically used to read data from files defined by formal languages, and generate objects, which can be modified and rewrite to the same files.

### *Transformation from UML Activity Diagrams to PEPA*

The UML activity diagrams considered in [Cavenet 04] as the input model for a transformation to PEPA includes choice, looping, control, object flows, and other elements defined in UML specification documents, but no synchronization. The first step is to identify the components of the PEPA, distinguishing tokens and static components, i.e., the context object of the activity diagram is a token of the PEPA net, as is each object token involved in an object flow. Each activity of the diagram becomes an activity in the PEPA definition of the component. When a choice in the activity diagram is labeled by guards, the guards are elevated to the status of activities offered in competition in the PEPA token component. For each object token, a PEPA token component should be defined, and the activity on which it cooperates with the context token, it is given activities to bring it into the place of interaction

and remove it from the place of interaction. An exponential delay must be associated with each activity, whether it is a transition.

### 2.2.4 Transformation from UML Diagram to Simulation Model

#### *Transformation from UML Use Case and Activity Diagrams*

[Balsamo 03] [Marzolla] derive a simulation model from annotated UML software architectures. It first proposes the annotation, a subset of annotations defined in [OMG SPT], for some UML diagrams to describe performance parameters. Then the simulation model is derived by automatically extracting information from Use Case and Activity Diagrams in terms of the XMI descriptions of UML diagrams. This information is used to build a discrete-event simulation model, which is finally executed. Simulation results are inserted back into the original UML diagrams as tagged values to provide feedback at the software architectural design level.

The main steps are defined as follows:

1. Consider an UML representation of a software system in terms of Use Case and Activity diagrams. Actors representing workloads applied to the system and Use Cases representing scenarios are identified. Actors can be stereotyped as <<PAopenLoad>> or <<PAclosedLoad>> [OMG SPT] to represent respectively open and closed population of users accessing the system. Use Cases are tagged with <<PAprob>> whose value indicates the probability of executing that scenario. Each Activity should be tagged with (PArep), the number of times the step has to be repeated, (PAinterval), the delay between repetitions of the same step, (PAdelay), an additional delay for each step representing user "think time", and (PAdemand), the service demand of the step.

2. The simulation model is automatically derived from the XMI description of the UML diagrams with the support of some tools.
3. The simulation model is executed, optionally asking the user to specify some parameters for the simulation, such as the desired confidence level for the estimation of the performance indices, the confidence interval width and the simulation length.
4. Simulation results, such as average delay, are inserted back into the UML model as tagged values associated with Activities and Use Cases.

### *Transformation from UML Class and Sequence Diagrams*

The same two authors, Arief and Speirs, developed an automatic tool in a several papers [Arief 99 A], [Arief 99 B] and [Arief 00] for deriving process-oriented simulation models from UML Class and Sequence diagrams. Their approach consists in transforming the UML diagrams into a simulation model described as an XML document, which is used to store the information that is relevant to both UML and simulation model. This model can then be translated into different kinds of simulation programs, even written in different languages. The authors introduce a simulation framework and a Java package which is constructed to support the framework and the simulation environment.

## 2.2.5 Transformation from UML Diagram to LQN

The basic elements In LQN models are task, entry, device and messages. The software resources are called tasks, representing a software process with its own thread of execution. The hardware resources are called devices, such as CPUs and disks. Tasks can have priority on devices. There are three types of message interaction, asynchronous messages with no reply, synchronous messages which block the sender until there is a reply, and forwarding messages, combinations of synchronous and asynchronous behavior. The workload of a LQN is driven by arrival streams of external requests, and by tasks which cycle and make requests, called reference tasks. Tasks receive either kind of request message at

designated interface points called entries. A task has a different entry for every kind of service it provides; an entry also represents a class of service. Internally an entry has service demands defined by sequences of smaller computational blocks called activities, which are related in sequence, loop, parallel (AND fork/joins) and alternative (OR fork/joins) configurations. Activities have processor service demands and generate calls to entries in other tasks.

### *Transformation from UML 1.4 Collaboration, Deployment, and Activity Diagrams*

Two methods are used to implement this transformation: it can be a purely graph grammar-base transformation with the support of certain graph rewriting tool, or the combination of graph transformation and XML tree manipulation.

Because UML and LQN models are described by graphs, the graph-grammar formalism is appropriate for the transformation from UML to LQN. A graph grammar is a set of production rules that generate a language of terminal graphs and produce non-terminal graphs as intermediate results. A graph rewriting system, such as PROGRESS, is a set of rules that transform one instance of a given class of graphs into another instance of the same class of graphs without distinguishing terminal and non-terminal results.

### *Graph Grammar-based Transformation*

In [D.C.Petriu 00] [Amer] [Amer thesis] [D.C.Petriu 02], a graph grammar-based transformation from UML design models into LQN performance models is proposed. The LQN structure is generated from the high-level software architecture, such as a collaboration diagram, which shows the architectural patterns used in the system, and from deployment diagram, which indicates the allocation of software components to hardware devices; The

LQN parameters are obtained from detailed models of key performance scenarios, such as UML sequence and activity diagrams, which include performance annotations based on [OMG SPT]. The transformation is implemented with PROGRES, a well-known visual language and environment for programming with graph rewriting systems.

The input graph schema for architectural descriptions and the output graph schema for LQN models are defined according to the PROGRESS. Graph transformation rules have been defined for each architectural pattern. A PROGRES transaction is executed for every architectural pattern found in the input graph. The translation process ends when all the patterns have been processed. The final result is an LQN model.

A graph schema that describes the static properties of the graph has to be defined first so as to use PROGRESS for the transformation. The schema shows the types of nodes and edges that can appear in both the input and the output graph. In the intermediary transformation stages, the graphs contain mixed nodes and edges. Applying a set of production rules in a controlled way performs the desired graph transformations. A production rule has a left-hand side defining a graph pattern that will be replaced by the right-hand side, another graph pattern. A rule also shows how to compute the attributes of the new nodes from the attributes of the old nodes [Schürr]

As mentioned above, the input graph, a UML model, contains collaboration diagrams representing the high-level software architecture, deployment diagram illustrating the allocation of high-level software components to hardware devices, and sequence or activity diagrams showing a set of key performance scenarios annotated with performance information. The output graph is an LQN model that can be read and solved by the existing LQN solvers.

Three main steps complete the transformation:

1. Generate the LQN model structure, i.e., determine LQN software tasks from the high-level architecture, LQN hardware devices from deployment diagram.
2. Generate LQN details on entries, phases, activities from scenarios. First transform scenarios represented as sequence diagrams into activity diagrams. Then for each scenario process the corresponding activity diagram(s) and match the communication pattern from the architectural pattern with the messages between components given in the activity diagram. Finally identify the activity diagram elements corresponding to different LQN entries, phases, and activities, and create the LQN elements.
3. Traverse the LQN elements, compute their parameters and output the model file.

### *XSLT transformation*

The Extensible Stylesheet Language for Transformations (XSLT) is a flexible language for transforming XML documents into various formats including HTML, XML, text, PDF and even Java code. XSLT is already supported by tools, and is used to define transformations that convert a source document in a tree format into a result document expressed as a tree format too. The input to the transformation in [Gu 02] is an XML file that contains an UML model in XML format. The output is another XML tree that corresponds to the generated LQN model, and the final output is a text file containing the LQN model description in a text format. The transformation process is very similar to above one:

1. Generate the LQN model structure
   1.1. determine LQN software tasks from the high-level architecture
   1.2. determine LQN hardware devices from the deployment diagram
2. Generate LQN details on entries, phases, activities from scenarios

2.1. transform scenarios represented as sequence diagrams into activity diagrams

2.2. for each scenario process the corresponding activity diagram(s)

2.2.1. match the inter-component communication from the architectural pattern with the messages between components given in the activity diagram

2.2.2. divide the activity diagram into subgraphs, map them to different LQN entries, phases, and activities, and create the respective LQN elements

3. Traverse the LQN tree and write the model's textual description.

### *XML Algebra - based Transformation*

The mapping of XML algebra - based method [Gu 05] between the input model, an annotated UML model and the output model, LQN model, is defined at a higher level of abstraction based on graph transformation concepts, but the implementation of the transformation rules and algorithm uses lower-level XML tree manipulation techniques, XML algebra. In another word, XMLgebra is used to express higher-level transformation rules from an input model to an output model.

The theoretical foundation is that XML becomes a first-class data type in any modern programming language, such as Java. First-class data type basically means all operations allowed on any object of any type in the language must be allowed on all objects of this first-class data type. In a Java program XACT framework allows programmers to manipulate XML templates as first-class data types. An XML template is a well-formed XML fragment containing named gaps that may appear in place of elements and attributes. The XMLgebra operations are based on the XPath and DTD standards.

This technique accepts an XML file representing the input model compliant with the input DTD/schema, and generates an output XML tree compliant with the output DTD/schema.

The input and output DTD (schemas) describe formalisms from different domains. The detailed definition and implementation of the transformation rules and algorithms are at the XML level using XML tree manipulations based on eXMLgebra. The implementation of this technique actually uses XSLT, a XML tool.

Transformation rules map a concept from the input model to an output concept and define how to compute the output node attributes based on the input nodes. A transformation algorithm decides in what order to invoke the transformation rules over a given input tree for generating output sub-trees, and how to "glue" these sub-trees together to construct the complete output tree. The transformation has two steps. First it transforms the input UML model to a common intermediate model, IM (Intermediate Model), which is similar to Core Scenario Model (CSM) [D.B.Petriu 04], then from IM to any preferred performance model, such as LQN.

### *Transformation from UML 2 Deployment and Activity Diagrams*

The transformation technique in [D'Ambrogio] uses metamodeling for defining the abstract syntax of models, the interrelationships between model elements and the model transformation rules. It exploits model-driven development and MDA (Model Driven Architecture) that support the development of software systems through the transformation of abstract models to executable components and applications. MDA emphasizes the role of models as the primary artifacts of development by providing a set of guidelines for structuring specifications expressed as models and the transformations between such models. The transformation maps the elements of a source model that conforms to a specific metamodel to elements of another model, the target model that conforms to the same or to a different metamodel. The proposed approach is guided by a model transformation framework that applies the model-driven principles in the context of performance engineering. The

framework is applied to the transformation from source models, UML 2.0 Deployment and Activity Diagrams, to target models, LQN.

### 2.2.6 Transformation from UCM to LQN

The UCM notation [Buhr 96] [Buhr 98] was invented to capture designer intentions while reasoning about concurrency and partitioning of a system in the earliest stages of design, and is intended to be intuitive, and high-level. Details can be represented, but are not the purpose. A Use Case Map is a collection of elements that describe one or more scenarios unfolding throughout a system. The core notation is very simple. The notation has only three fundamental elements: scenario paths, runtime components and responsibility-points along paths touching components to indicate that components have responsibilities along paths. The start and end symbols for paths indicate places—in the environment or internal to the system—where stimuli occur and the effects of stimuli stop actively rippling through the system. Paths, components, and responsibilities all have labels.

A scenario is represented by a path traversed by a token from start to end points A path can be traversed by many tokens, and several tokens may occupy a single path at once. Paths can be overlaid on components which represent functional or logical entities, which may represent hardware or software resources. The workload of a path is indicated by annotations to its start point. A path can be refined hierarchically by adding stubs, which represent separately specified maps called plug-ins. There may also be several alternative plug-ins for any stub. Paths have the usual behavior constructs of OR fork/joins, AND fork/joins and loops.

The UCM is transformed into an LQN [D.B.Petriu 02] on a path by path basis. Each start point is assumed to begin an independent path, and as such is assigned to its own reference

task. Reference tasks act as the work generators for the LQN model. Each reference tasks is assigned arrival rates and distributions as specified by the start points in the UCM. Similarly, LQN activities are assigned workload demands as specified in the corresponding UCM responsibility and OR branches are assigned probabilities set in the UCM. If any performance data is missing from the UCM, default values are assigned. The Following table shows correspondences between UCM and LQN elements.

**Table 2-1 Correspondences between UCM and LQN Elements**

| UCM Construct | LQN Construct |
|---|---|
| start point | reference task |
| responsibility | activity |
| AND/OR forks and joins | LQN AND/OR forks and joins |
| component | task |
| device | device |
| service | entry in a task (with a dedicated processor) |

### 2.2.7 Summary

We investigated a few implemented transformations from design to performance model in this section, some of them are surveyed in [Balsamo 01 A], [Balsamo 01 B] and [Balsamo 04]. Therefore, the corresponding transformations (from design to performance model) in Figure 2-1 can be instanced as following:

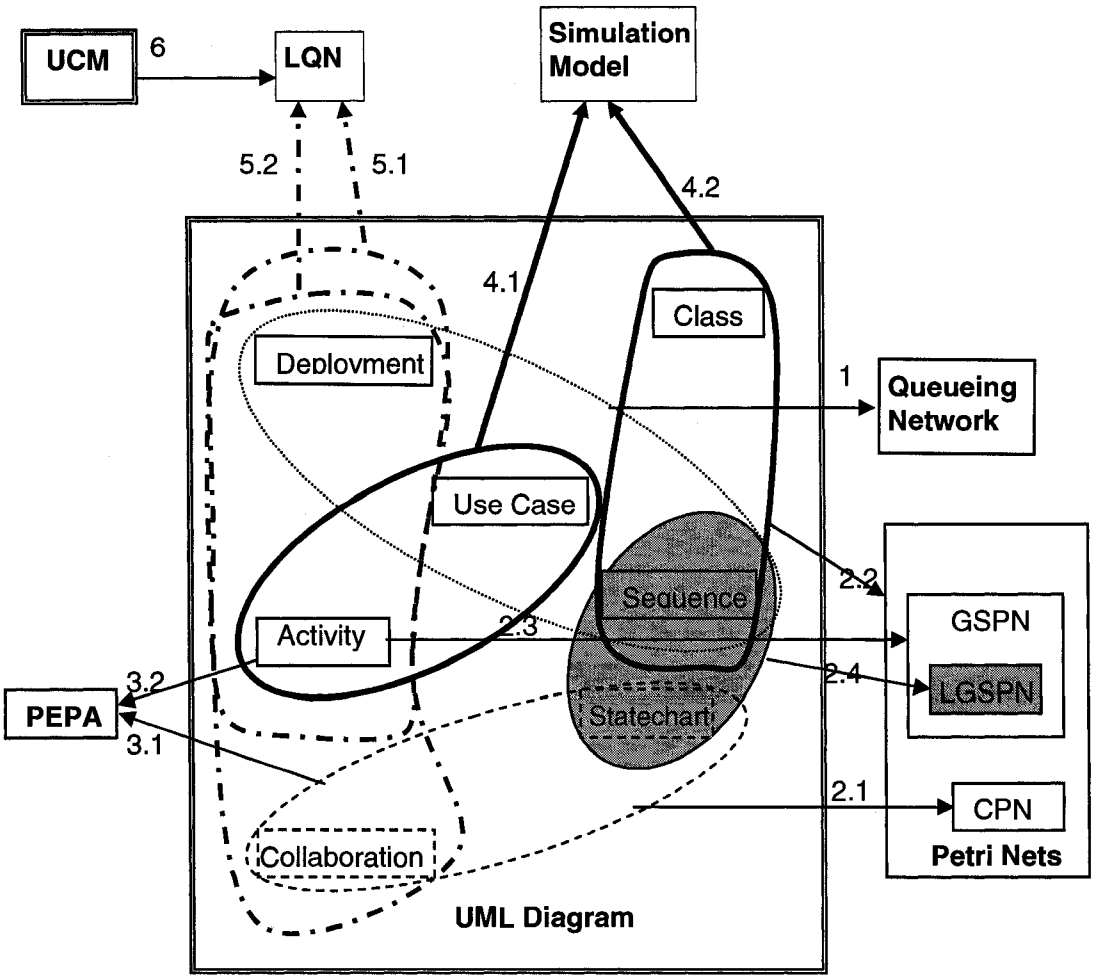



**Figure 2-3 Transformations from Design to Performance Model**

## *2.3 Trace – Based Approaches to Generate LQN model*

Hrischuk described an automated generation technique for various software performance models based on object-oriented prototyping environments [Hrischuk 95]. Later he proposed a more formal technique [Hrischuk 99], called Trace-Based Load Characterization (TLC), which greatly enlarged the range of patterns that can be detected and modeled by the former one. To be exact, TLC can handle both synchronous and asynchronous interactions, and no longer depends on a global monitor to order the events. Tauseef A. Israr developed another technique, Software Architecture and Model Extraction (SAME), to extract the

communication patterns through interaction tree transformations [Tauseef 05], and then created LQN model from communication patterns.

Both TLC and SAME are based on execution traces, which can be obtained from many sources at various steps of development, such as a very abstract executable CASE tool model for a code prototype, and this is why they are named "Trace-based" method. Also they have similar performance model generation processes, i.e., obtain execution traces, extract communication patterns, generate LQN sub–models, and then generate a LQN model. As the design evolves, both allow the increasing details of design be easily incorporated into the performance model to make more complex performance concerns to be examined.

The difference is mainly the way to extract communication patterns. This means they use different intermediate models as bridges between execution traces and LQN models: TLC forms a graph called a TOEG (Task and Operation Event Graph) produced from a trace event and reduced to a LQN sub-model, while SAME uses interaction tree created as events unfold in a trace and transformed to communication patterns, from which LQN models are extracted.

Another difference is they use different traces. SAME uses a simpler trace format that includes only task id, timestamp, and message id and event type for each trace event, but TLC uses a more complicated one, third generation angio trace, which contains scenario name, thread id, thread's node counter and so on. The different formats of trace data mean they have the different ranges of application, different assumptions about the system and different algorithm complexity. For example, in SAME a global monitor is necessary, because timestamp in trace data depends on the synchronization of clocks on all distributed nodes, however, in TLC it is unnecessary because the introduction of the causal logical clock.

Moreover, TLC is more platform dependent, because an angio trace has to be extracted from certain environment, such as ObjecTime, Parasol, and so on; but SAME does not rely on certain application environment. However, In SAME, components must not have internal parallelism with forking and joining of the follow execution.

### 2.3.1 Trace-based Load Characterization (TLC)

TLC can be used for distributed systems composed of geographically dispersed, heterogeneous hardware with scheduled, concurrent software components (referred as tasks) and can simultaneously execute several distributed operation, i.e., a set of coordinated interactions between server and user tasks. Tasks communicate solely by messages. Communication is reliable, point-to-point, dynamically established at execution, having finite but unpredictable delay. The communication protocol may be RPC (Remote Procedure Call which are analogous to synchronous or blocking), asynchrony (blocking) or forwarding. A scenario is an execution of a distribute operation and can be recorded as a trace. To apply TLC It must be assumed that a finite set of scenarios can capture all the behavioral characteristics of the distributed operation needed for constructing performance model. And the executable design must at least represent some task architecture and include the coarse behavior of each task.

There are three steps to construct a LQN model in TLC:

1. **TLC needs constructing an early prototype which is used to describe an object-based system, and then it is executed to produce angio traces.**

In the first step, the significant scenarios, such as heavily used requests or requests of special importance, are selected. Besides a scenario's name, scenario options are also defined, which reflect variations in the request data or prototype data state. Scenario name

and options are associated with trace name, which identifies an execution of a scenario and later identify the scenario's resulting workload description, i.e., LQN sub-model. Instrumentation is added to the prototype so when the prototype is executed the trace name is recorded in the angio trace. Execution of prototype also generates angio events, which include all the activities that a certain task generates during the execution of the scenario and thus provides causal information. The angio events are then gathered and filtered into angio traces.

Each angio trace event records an event timestamp representing cause and effect relationships between two angio events. Each event timestamp has at least six parameters: two parameters (task time stamp) are related to the task's execution and four parameters (operation time stamp) describe the execution of the distributed operation. A task time stamp consists of a unique task identifier and a task node counter that orders the events recorded by a task. An operation time stamp consists of a unique operation (scenario) name that associates an event with a scenario, a unique scenario thread identifier that is assigned when an operation thread begins, a thread node counter that orders the events of an operation thread, and an event type label.

2. **These angio traces are used to produce an intermediate model, i.e., TOEG, which is transformed to LQN submodel that characterizes the activities of involved tasks and their interactions with each other.**

In the second step, the angio traces are ordered by timestamps and used to construct the TOEG. An angio trace has six event ordering relations to identify a given event's succeeding or preceding event in the task event graph or the operation event graph. The transformation from an angio trace into a TOEG is largely composed of converting events to node and adding edges according to a set of graph grammars and graph rewriting rules.

TOEG is a node-labeled, directed, binary, acyclic graph whose structure is based on the causal relationships between the events in the same distributed operation providing empirical information about traversal frequencies for data-dependent choices and loops. Each trace event produces a labeled node in the TOEG. The node's label contains six parameters corresponding to the parameters of event timestamp (fields in the record of trace data) as Table 2-2 shows.

**Table 2-2 The Parameters of Node Labels in TOEG and in Trace Events**

$$e = \left| \frac{\text{distributed operation information}}{\text{task information}} \right| = \left| \frac{j,\ k,\ m,\ l}{i,\ v} \right| , \text{(eq. 1)}$$

where

$j$ = the scenario name;
$k$ = the scenario thread identifier;
$m$ = the scenario thread's node counter;
$l$ = {*Be*, *Ac*, (*Fk*, *j*, *k'*), *Ed*} is the type label. The fork label also includes the scenario name ($j$) and thread identifier of the forked thread ($k'$);
$i$ = the task identifier;
$v$ = the task node counter.

The arcs in the TOEG are deduced from the event timestamp. Combined with a set of resource functions, which give the resource demands of each statement in the prototype, TOEG is used to estimate service times. From the distributed operation and task's view, TOEG is actually a graph overlays task event graph and operation event graph. Therefore each event in a TOEG has two points of reference and this allows the same event to be interpreted differently for each frame of reference. A task event graph characterizes the execution of a single task as an attributed, edge labeled, directed, linear graph, which has two types of nodes and two types of edges. An operation event graph characterizes an operation as a set of execution threads that share a set of interacting tasks, that is to say, it characterizes the concurrency and sequence of events that occur in a scenario. An operation

event graph is a node labeled, edge labeled, binary, finite, directed, acyclic graph which has more node types and edge types than task event graph.

The core of the TLC is the reducing of a TOEG to an LQN submodel; this is basically a graph rewriting transformation, i.e., transform one graph to another according to some graph rewriting rules. The rules have an antecedent part and a consequent part. The former is a graph fragment that completely matches an interaction pattern i.e., an interaction template. The latter is a sequence of modeling operations to develop parts of the LQN submodel. The order in which the rules are applied is important and governed by a control algorithm.

The main focus of the rules is to identify when an interaction template occurs and the type of the interaction. There are at least nine interaction templates: basic RPC, asynchronous, RPC with phase two, forwarding RPC, single RPC using asynchronous, nested RPC using asynchronous, activity node, end node and begin node. When a template match occurs, modeling operations are executed to create a corresponding fragment of the LQN submodel. The control algorithm manages the matching of interaction templates. It has an initialization component and an analysis component.

3. **Finally, by aggregating several LQN submodels and adding configuration information, a performance model is developed.**

In the third step, an LQN is built by including the workload parameters, providing the target configuration and assigning resource demands to the activities. The workload parameters for each scenario include the customer population and thinking time for closed model, or the arrived rate for open model. The target configuration includes the number of processors, processor speeds, scheduling disciplines, task priorities and the assignment of tasks to processing elements. Resource demands for activity nodes are assigned by using resource

functions. Once the LQN model is produced it may be simplified. By adding new types of requests, varying environment information or modifying the design, different scenarios can be generated so different measures can be obtained for system tuning or refining.

### 2.3.2 Software Architecture and Model Extraction (SAME)

In SAME, communication patterns are detected through interaction tree. An interaction tree represents a part of a scenario, i.e., a sub-scenario, which is started with a single message making the root always have one child. It is created and transformed as events unfold in a trace. Vertices in a tree represent activations of processes, and arcs represent the messages passed between these processes. Since any cycles are removed as soon as they are created, ITT always forms a set of trees, which is maintained by an algorithm. The algorithm has four parts:

- **Part A**: initialize the set of tasks, set their live node counters to zero and their next node counters to one.
- **Part B**: Process message records until the trace terminates. First add a message arc with time label according to two different cases. Then deal with any cycle induced in the last step. Finally do tree cleanup, a sequence of operations when the root of a trace is removed and its last child added becomes a root of a new tree. When the trace terminates, for each child node of its parent node create an asynchronous interaction record. At this point we have a set of interaction records, each of which represents a single interaction.
- **Part C**: Assemble the interaction records into a LQN model. The tasks in this LQN model may have many entries. Many of these entries may perform the same operations so they should be merged into a single entry.

- **Part D**: Simplify the entries to merge those that perform the same operations. The result of this part is the final LQN model structure. To create an LQN model, the performance parameters must be estimated and inserted.

## *2.4 Trace – Based Approaches for Performance Visualization, Evaluation and Tuning*

Trace information can not only be used to construct performance model, such as execution traces in TLC and SAME , but to depict behavior and provide graphical performance summaries [Heath 91] [Heath 95], to visualize and evaluate performance [Dauphin], to visualize and tune performance [Yan], and to determine resource demands in [Mehra] and [Sarukkai].

### 2.4.1 Performance Visualization with ParaGraph

PICL [Geist], a subroutine library that implements a generic message-passing interface on a variety of multiprocessors, runs on parallel architectures from many different machines on which the library has been implemented (including Cogent, Intel, N-Cube, and Symult). PICL provides a trace file that records important execution events like sending and receiving messages. Similar to angio trace, the trace file contains one event record per line, and each event record comprises an integer set that specifies the event type, time stamp, processor number, message length, and other similar information. In PICL, the clock calls necessary to determine the time stamps for the event records. The user defines tasks within a program by using special PICL routines to mark the beginning and end of each task and assign it a task number.

In [Heath 91], [Heath 95], a graphical visualization software tool, ParaGraph, was developed to gain insight into the behavior of parallel programs so as to improve their performance.

ParaGraph provides a detailed, dynamic, graphical animation of the behavior of message-passing parallel programs and graphical summaries of their performance. The trace file is treated as a script to be played out, visually reenacting the original live action to provide insight into a program's dynamic behavior, so ParaGraph is basically a trace file viewer. It provides several distinct visual perspectives from which to view processor utilization (including processor count, concurrency profile, and utilization meter and so on), communication traffic (including message queues, communication matrix, communication meter and so on), tasks, and other performance data.

### 2.4.2 Performance Evaluation with SIMPLE

A comprehensive methodology for monitoring and modeling programs in parallel and distributed systems is presented and implemented in [Dauphin]. The monitor system ZM4 used is an event-driven hardware monitor, which reveals the dynamic flow of program activities represented by sequences of events. The dynamic behavior is abstracted to event traces, the basis of SIMPLE, a comprehensive set of tools for performance evaluation and visualization. The trace format is different with angio trace as following graph (Figure 2-4) shows. Basically it allows arbitrary structure, format, and representation described by the versatile event Trace Description Language TDL. An event trace is a trace file generated during one program execution. A section in the event trace which has been continuously recorded is called a trace segment, or a scenario. A trace segment describes the dynamic behavior of the monitored system. Each trace segment begins with a special data record, the segment header, which contains some useful information about the following segment, or is simply used to mark the beginning of a new trace segment. Whenever the monitor device recognizes an event, it stores a data record, an E-record. E-record fields can be classified in four basic field types: Token, Flags, Time and Data.

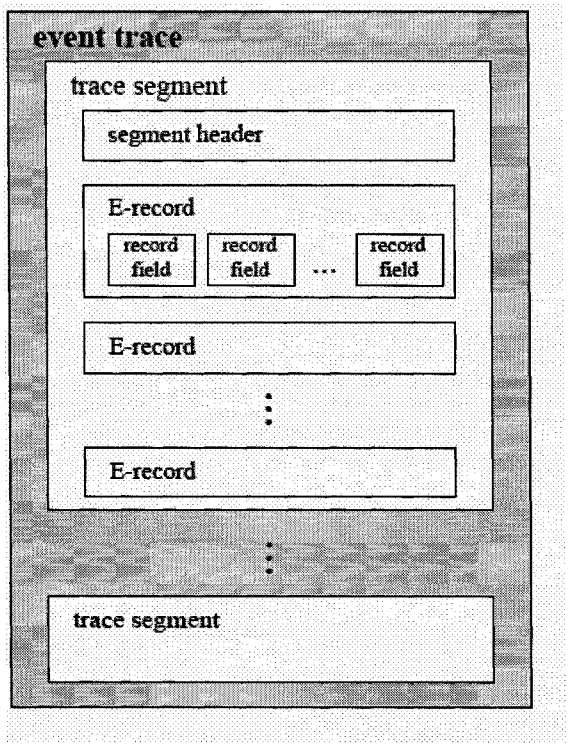



**Figure 2-4 Trace Format in [Dauphin]**

The methodology in [Dauphin] is the integration of monitoring and modeling. A functional model that disregards all implementation aspects is constructed first. In this model, properties of an algorithm which determine the functional behavior of a program are described. Mapping the functional model onto a given computer configuration leads to a functional implementation model. The monitoring model is a subset of the functional implementation model. The instrumented program results from an already implemented program and the respective monitoring model. Running the instrumented program, i.e. execution and measurement, produces an event trace by ZM4. The subsequent evaluation of the event traces by SIMPLE provides overall results for validation and detailed performance parameters for assigning realistic time attributes to the program's activities.

### 2.4.3 Performance Tuning with AIMS

A software toolkit that facilitates performance evaluation of parallel applications on multiprocessors, the Automated Instrumentation and Monitoring System (AIMS), is described in [Yan]. AIMS has four major software components: source-code instrumentor, run-time performance-monitoring library, trace file animation and analysis toolkit, and trace post-processor. The source-code instrumentor, Xinstrument, inserts instrumentation into the source-code to record performance data (execution traces) from the program's execution. The monitor (run-time performance-monitoring library) comprises routines called by the instrumented code in order to trace performance during program execution. Trace data is collected into a trace file subsequently used by VK and tally (trace file animation and analysis toolkit), the former provides animated views for displaying program behavior, while the latter tabulates cumulative statistics gleaned from the trace file. The trace files can be processed by tpp (trace post-processor) before being fed to VK/tally.

Similar to ParaGraph in subsection 2.4.1, visualization of execution traces can help application developers uncover system behavior not modeled and to take advantage of specific application characteristics. For example, VK displays a program's execution (execution traces) via five animated views. These views present information indicating when certain constructs were executing, when messages were sent, how long messages were queued-up before being processed, and when nodes were running/idle. Tally generates a list of resource-utilization statistics on node-by-node and routine-by-routine bases. These statistics can help point out inefficient sections of code, which can then be more closely examined with VK.

## 2.5 Core Scenario Model (CSM)

From section 2.2 and 2.3 we know that the number of transformations from design model (including execution traces) to performance model is huge. In Figure 2-3, there are twelve such transformations. If we use M and N denotes types of design and performance model, then for this N-by-M problem, we need N*M different types transformation, which are best addressed by a common intermediate format, such as CSM in [D.B.Petriu 04] [Woodside 05], and a kernel language KLAPER (Kernel LAnguage for PErformance and Reliability analysis) in [Grassi]. The advantage of using intermediate language / metamodel is that only N+M transformations need to be developed instead of N*M. A potential advantage is that they could be possibly simpler to devise than the direct transformations because the intermediate model is likely to be closer to both the source and target models, this will be approved later in this thesis.

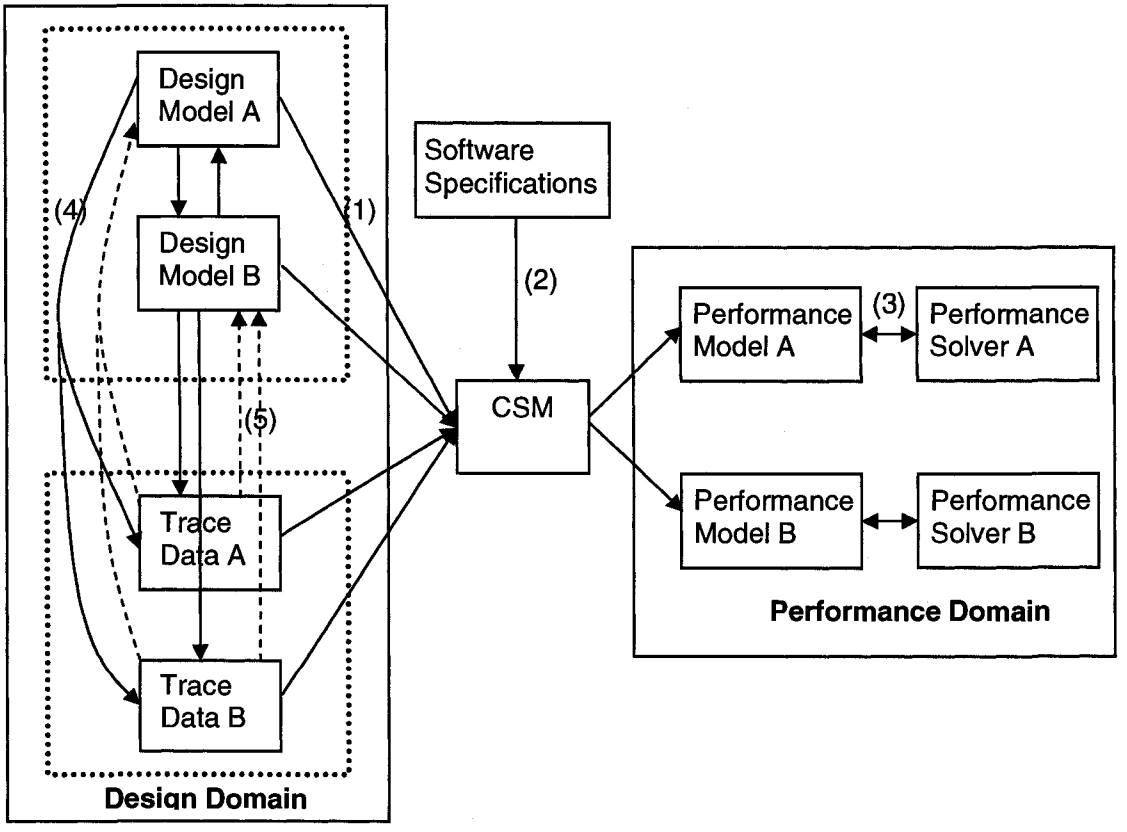



**Figure 2-5 Model transformations with CSM**

Figure 2-5 shows changes of Figure 2-1 after the introduction of CSM, which is basically a bridge between design model and performance model thus it reduces the burden of defining a variety of direct transformations from the former to the latter as we discussed above. It captures the essence of performance specification and estimation as expressed in the UML SPT Profile, or extracted from trace data, and abandons the design detail irrelevant to analysis. As with KLAPER, CSM is defined as a MOF (Meta-Object Facility) metamodel, which is the metamodeling framework proposed by the OMG for the management of models and their transformations within the MDA (Model Driven Architecture) approach to software design. A metamodel defines the syntactic rules to build legal models.

### 2.5.1 CSM Metamodel

The CSM metamodel is defined in[D.B.Petriu 04], which includes its class structure shown in Figure 2-6. A CSM consists of a collection of two classes, GeneralResource and Scenario. GeneralResource defines the class of resource; Scenario is an instance of a use case describing software behavior composed of objects of three classes, Step, PathConnection and Classifier.

**Step**

- A fundamental unit of executable functionality
- Describes the processing tasks (e.g. procedures, functions actions, etc.) that a system perform.
- Supports checking of the model and performance model generation.
- Subtypes:
    - **ResourceAcquire**: acquire step of a resource
    - **ResourceRelease**: release step of a resource

**PathConnection**

- Explicit representation of the scenario flow
- PathConnection objects are between a pair of Steps
- Its subtypes correspond to the sequential relationship types that are common in path models for real-time software. They have a different number of source Step and successor Step:
    - **Sequence** for simple sequence, one step after another, exactly one source (predecessor) and one target Step (successor).
    - **Branch** for an OR-fork, describes the branch of alternative paths, one source Step and multiple target Steps
    - **Merge** for an OR-join, describes the join of two or more independent alternative paths, multiple source Step and one target Steps. It is not used to synchronize concurrent paths but to accept one among several alternative paths.
    - **Fork** for AND-fork splits a path into two or more concurrent paths, one source Step and multiple target Steps
    - **Join** for AND-join, describes the join of parallel paths, multiple source Step and one target Steps
    - **Start** is where scenario is invoked. A scenario may have more than one start.
    - **End** is where scenario is stopped. A scenario may have more than one end.

**Classifier**

- Describes a set of instances that have common features.
- Only one subtype, Message, is defined to describe the communications between steps.

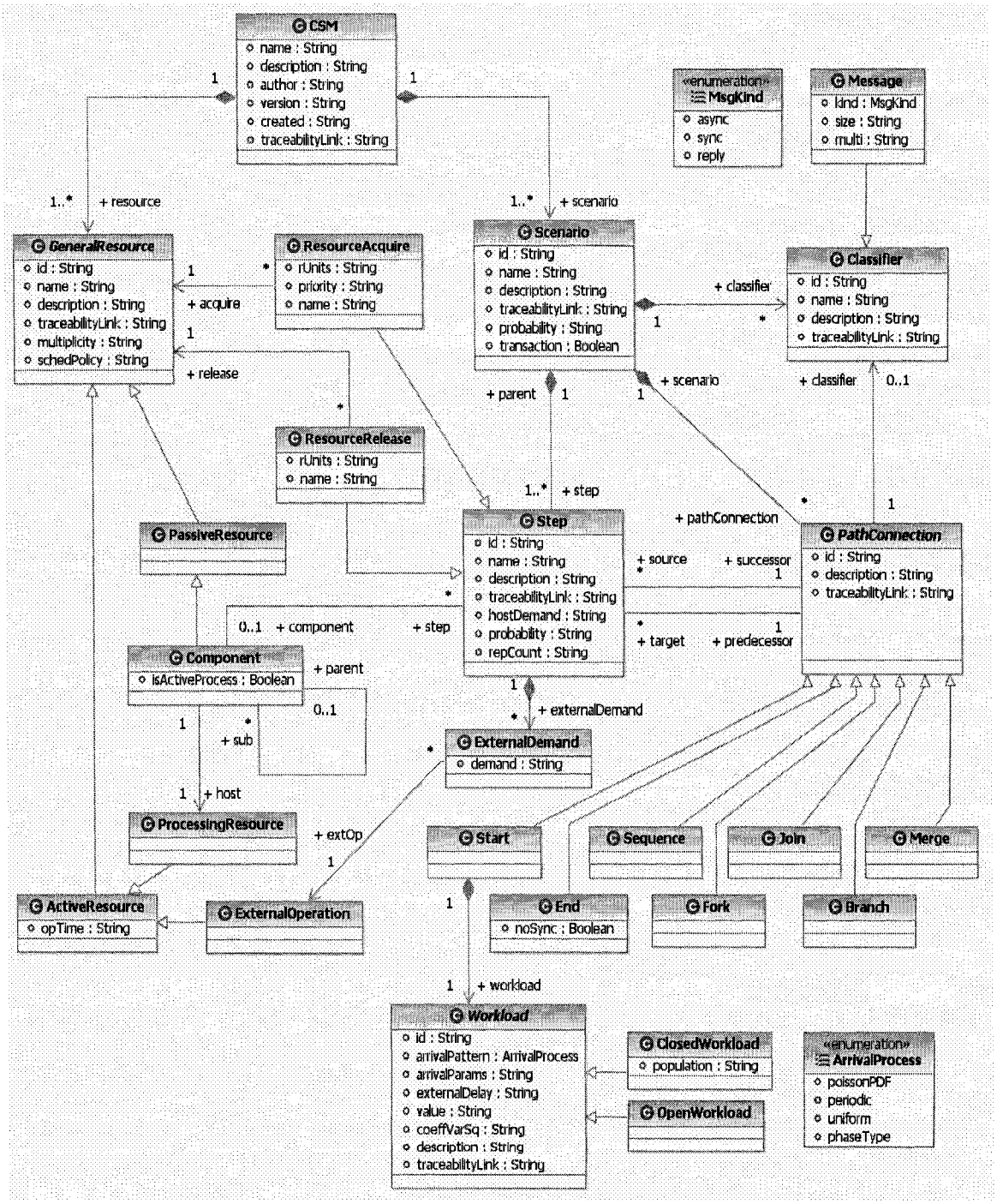



**Figure 2-6 Classes in CSM meta-model**

### 2.5.2 Notations of CSM

Only objects of class Step and PathConnection and their subclasses can be shown on CSM diagram. Figure 2.7 shows all the CSM notations.

**Step and its subclasses:**

Step: **Step** : event

ResourceAcquire: **R_Acquire** : resource

ResourceRelease: **R_Release** : resource

**PathConnection and its subclasses:**

Start: **Start** : scenario

Fork: **Fork** ...

Join: **Join**

End: **End**

Sequence:

**Figure 2-7 CSM Notations**

### 2.5.3 Transformation from Design Model to CSM

#### *Transformation from UML Deployment and Activity Diagrams to CSM*

A general transformation approach is described in [D.B.Petriu 04], which reads XMI files representing the UML design model, builds a data structure to represent the UML objects, and uses it to create a DOM (Domain Object Model) tree for the CSM, which can be output in XML. Basically it depends on the one-to-one mapping relationships of elements in UML activity diagrams with SPT Profile and CSM, and examines each activity diagrams so as to obtain all the information needed in CSM except for components, which are constructed from UML Deployment diagram. For example, from the Initial PseudoState in UML activity diagram, a Start Step is created for CSM. If a State following this has a <<PAworkload>> stereotype defined in SPT Profile, the workload parameters become attributes of the Start Step.

#### *Transformation from UCM to CSM*

Similar to the transformation from UML to CSM, the transformation from UCM to CSM [Zeng] also refers to the mappings from the UCM design models to the CSM representation. However, the mappings are divided into two groups: explicit mappings, such as UCM design Class to CSM class CSM, Model Class to scenario, component-ref to component and so on, and implicit mappings, such as CSM ResourceAcquire and ResourceRelease class deduced from UCM paths. Corresponding to different mappings, several algorithms are proposed to implement these mappings.

### 2.5.4 Transformation from CSM to Performance Model

#### *Transformation from CSM to LQN*

An algorithm to generate a LQN model from a CSM is proposed in[Woodside 05], which is based on the algorithm successfully used to generate models from UCM in [D.B.Petriu 02]. The algorithm first generates the LQN resources by examining the CSM resources, an LQN Processor for each CSM ProcessingResource, and an LQN Task for each CSM Component. Then it traverses the scenario in order to determine the sequencing of the CSM Steps and to discover the calling interactions between Components. The traversal generates an LQN Activity for each CSM Step it encounters. Whenever a calling interaction between two Components is detected, an Activity is created in the Task corresponding to the caller Component and an LQN Entry is created in the Task corresponding to the called Component. The type of calls is detected by their context, that is, a message which returns to a Component that previously sent a request is considered to be a reply to a synchronous call. Any calls that have not generated replies by the time the end of the scenario is reached are considered to be asynchronous. In a similar way, the algorithm generates all the other elements needed in LQN.

#### *Transformation from CSM to Petri Nets*

The automatic transformation in [Maqbool] is achieved by making use of the XML Schema for CSM and Petri Net DTD. The algorithm reads an XML file containing the input CSM and converts it to a DOM tree, then finds the root node and from there finds the Scenario node. The Scenario node is translated as a Page (a network of places, transitions and arcs) in the Petri Net representation, and then the algorithm traverses through the children of Scenario node and applies transformation rules.

# Chapter 3: CSM Elements Generation

The two trace-based transformation techniques mentioned earlier use LQN as their target model. However, we know that an intermediate model, such as CSM, can reduce the number of transformation types between a design model and a performance model, or between trace data and a performance model, so it is a natural way to transform execution traces. Also TLC and SAME use different traces: the former uses the third generation angio trace, while the latter uses a much simpler trace format. Though the third generation angio trace informal includes more information, which allows TLC to have a more extensive range of application than that of SAME, it also needs the supporting of development tools, and may need more complicated algorithms. On the other hand, the traces that SAME uses are very simple and much easier to obtain, therefore, we will use the same trace for the transformation to the CSM. Figure 3-1 shows the scope of thesis, i.e., the transformation from the execution traces to CSM. Methods used to generate instances of CSM classes will be discussed in this chapter.


**Figure 3-1 Thesis Scope**

This chapter will first introduce the trace format in section 3.1. Next, section 3.2 will explain why instances of Component and Step can be extracted from execution traces directly while other objects have to be deduced. Further, this section will show how to generate associations between instances. In section 3.3, the overall algorithm used to construct CSM models will be described. Finally detailed algorithms will be introduced in the following sections.

## 3.1 Execution Traces

The trace used in thesis is much easier to obtain without the support of certain development environment because it does not require inserting a dye into log events. For example, a logger class that saves method invocations in Java can be used to record execution traces. When a message is sent to a receiving component from a sending component, both sender and receiver will generate a record in a trace file with plain text format. These trace records are separated by line terminators. The sending time and receiving time are always different due to the latency of transfer, time used for recording, and so on.

Each execution record contains four fields:

*Timestamp* *Event Type* *Component* *Message*

**Timestamp**: An integer; indicates the time that an event occurs.

**Event Type**: A string, indicates what kind of event a component can generate. There are two event types: sending event and receiving event. A sending event means that a component is sending a message to another component; a receiving event means that a component is receiving a message from another component.

We should notice that sending and receiving events generate two records in the trace data. These two records do not contain the component name at the other end. This means that a sending event record does not contain a receiving component name, and that a receiving event does not contain a sending component name. For example, the first record in Table 3-1, "9926267 send Server Book_START", indicates a sending event named Book_START is generated by Server at time 9926267, but we do not know who the receiving component is from this record. Similarly from the second record, we do not know who the sending component is.

**Component**: indicates the name of a component which can be a task, a thread, an object, a sub-module, and so on that generates an event, which is always a message passing event.

**Message**: indicates the message name that a component sends or receives. It is also the event name for receiving component.

**Table 3-1 Example of Trace Records**

| |
|---|
| 9926267 send Server Book_START |
| 9926465 receive Book136228 Book_START |
| 9926617 send Book136228 Book_END |
| 9926812 receive Server Book_END |
| 9927042 send Server setName_START |
| 9927266 receive Book136228 setName_START |
| 9927467 send Book136228 setName_END |
| 9927663 receive Server setName_END |
| 9927865 send Server setAuthor_START |
| 9928084 receive Book136228 setAuthor_START |
| 9928283 send Book136228 setAuthor_END |
| 9928479 receive Server setAuthor_END |

## *3.2 Extractable and Deducible CSM Class Instances*

Execution traces are input data for our transformation algorithm, while a CSM model is the output. From the view of implementation language, such as Java, a model is a collection of objects organized in a way defined by the CSM meta-model. Therefore, our job is to convert execution traces to CSM objects. The last section shows what information execution traces can offer, and subsection 2.5.1 shows how CSM models are constructed. Therefore, comparing execution traces, the input, with CSM models, the output, will show the kind of

CSM objects, that is, the instances of CSM classes defined in CSM meta-model that can be generated from execution traces.

For some CSM objects, such as objects of class Component and Step, the process is very straight forward. That is, from execution traces, component and step names can be extracted directly because execution records have fields called component and message name. The former is exactly the component name for a component object in CSM, and the latter is the step name for a general step object in the CSM. We call these CSM objects extractable CSM objects. For other CSM objects, there is insufficient information to directly extract them from execution traces. Some objects can be deduced from CSM Component and Step objects, or from objects deduced from these two objects. These objects are called deducible CSM objects. All other CSM objects can not be deduced at all. Some of them, such as ClosedWorkload, OpenWorkload and ProcessingResource objects must be added to the CSM manually in order to produce a performance model.

For object attributes, some use default values, thus can be ignored in the transformation from execution traces to CSM, for example, description, traceabilityLink and multiplicity can be none, so they can be ignored for instances of all the classes in CSM meta-model. Some attributes only need a global unique value, for example, all the classes have an attribute *id*, a string starting with "id" followed by a unique number, which can be created by a global integer variable. Others can be extracted or deduced in a similar way that objects are generated. Table 3-2 gives a brief view of what objects and attributes can be extracted or deduced from execution traces, and a brief explanation. This table does not contain the abstract classes in CSM class meta-model, that is, GeneralResource and PathConnection. Figure 3-2 shows all the classes whose instances can be extracted or deduced from execution traces in bold.

**Table 3-2 CSM Extractable and Deducible Class Instances and Attributes**

| *CSM Class* | *Description* | *Attribute* | *Description* |
|---|---|---|---|
| **Component** | Extractable | **name** | Extract from component field of execution traces. See Section 3.5 for details. |
| | | isActiveProcess | Ignorable |
| | | schedPolicy | No information |
| **Step** | Extractable | **name** | Extract from message name field of execution traces. Need to couple messages first. See Section 3.5 for details. |
| | | **hostDemand** | Calculate from message timestamp. Need to know communication patterns. See Section 3.8 for details. |
| | | probability | No information |
| | | repCount | No information |
| **Start, End, Sequence, Join and Fork** | Deduced from related Step instances. See Section 3.6 for details. | | |
| **ResourceAcquire** | Deduced from communication patterns. See Section 3.9 for details. | rUnits | No information |
| | | priority | No information |
| | | name | Ignorable |
| | | Other attributes are inherited from Step | |
| **ResourceRelease** | | rUnits | No information |
| | | name | Ignorable |
| | | Other attributes are inherited from Step | |
| **Message** | Deduced from communication patterns. See Section 3.7 for details. | kind | Deduced from communication patterns. |
| | | size | No information |
| | | multi | No information |
| | | name | Ignorable |
| **Scenario** | Composed of instances of Step, PathConnection, and Classifier | | |
| **CSM** | Composed of instances of Scenario and GeneralResource | | |
| Classifier | Ignorable: Message is the only subclass of Classifier | | |
| PassiveResource | Ignorable: Component is the only subclass of PassiveResource | | |
| ProcessingResource | Can be annotated to CSM | | |
| ExternalOperation | | | |
| ActiveResource | | | |
| ExternalDemand | | | |
| ClosedWorkload | | | |
| OpenWorkload | | | |
| Workload | | | |
| Merge | | | |
| Branch | | | |

In the CSM meta-model, object containers are used to construct associations between two objects. We need to set or add one object reference to containers of its associated objects. For example, in a Scenario object, there is a container that holds all the instances of Step, Classifier and PathConnection that construct this scenario. When a Step object is created, its reference needs to be added to this container thus their association is created. This is also the way that CSM instances are connected.

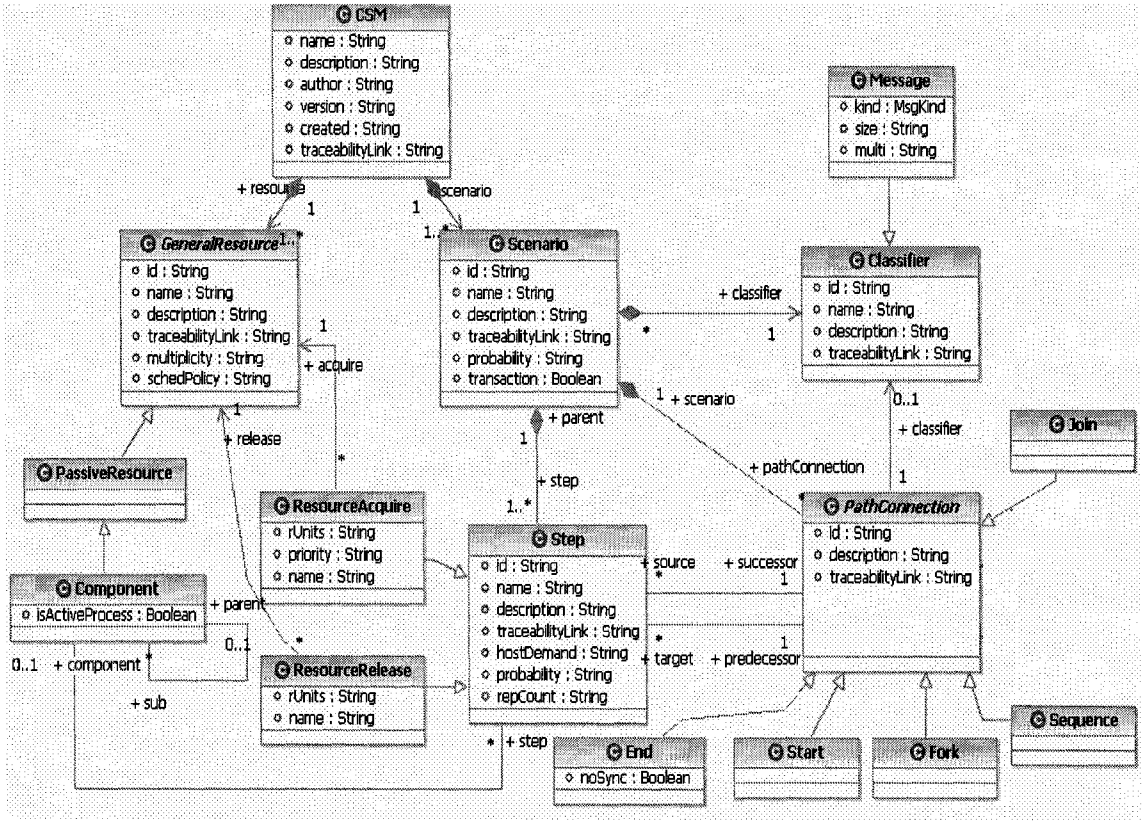



**Figure 3-2 Extractable and Deducible CSM Classes**

## *3.3 Overall Algorithm*

The transformation from trace information into a CSM described in the following is divided into several phases.

**Algorithm 3-1 Transformation of Execution Traces to CSM**

**Input:** Execution Traces

**Output:** An instance of CSM model

1. Preprocess
    a. Import execution traces from data file and generate instances of class RawMessage, and validate execution records (subsection 3.4.1)
    b. Generate instance of class ETComponent (subsection 3.4.2) (we use ETComponent for the execution traces component so as to distinguish CSM component)
    c. Generate instances of class CoupledMessage (subsection 3.4.3, **Algorithm 3-2**), which contains both message sending and message receiving components.
2. Generate instances of CSM Component and Step (Only Step itself, no its subclasses ResourceAcquire and ResourceRelease), and associations with related classes (Section 3.5, **Algorithm 3-3**).
3. Generate instances of PathConnection (Start, End, Sequence, Join and Fork) and associations with related classes (Section 3.6, **Algorithm 3-4**).
4. Detect synchronous communications and generate instances of Message with type *sync, async* or *reply*, and their associations with PathConnection instances (Section 3.7, **Algorithm 3-5**).
5. Calculate hostDemand value for each Step instance (Section 3.8, **Algorithm 3-6**).
6. Generate ResourceAcquire and ResourceRelease instances, and create their associations with Component, PathConnection and Scenario instances (Section 3.9, **Algorithm 3-7**).

The transformation algorithm is implemented in Java and the class organization is shown in Figure 3-3. The principle classes are:

- **RawMessage** hold execution records without changing message content

- **ETComponent** hold component names, and components' sorted sending and receiving message lists
- **CoupledMessage** hold messages with both sending and receiving components
- **CoupledMessageGenerater** read execution trace file and generate RawMessage instances
- **CoupledMessageGenerater** generate CoupledMessage instances
- **CSMGenerater** generate CSM instances (implement Algroithm 3-1)

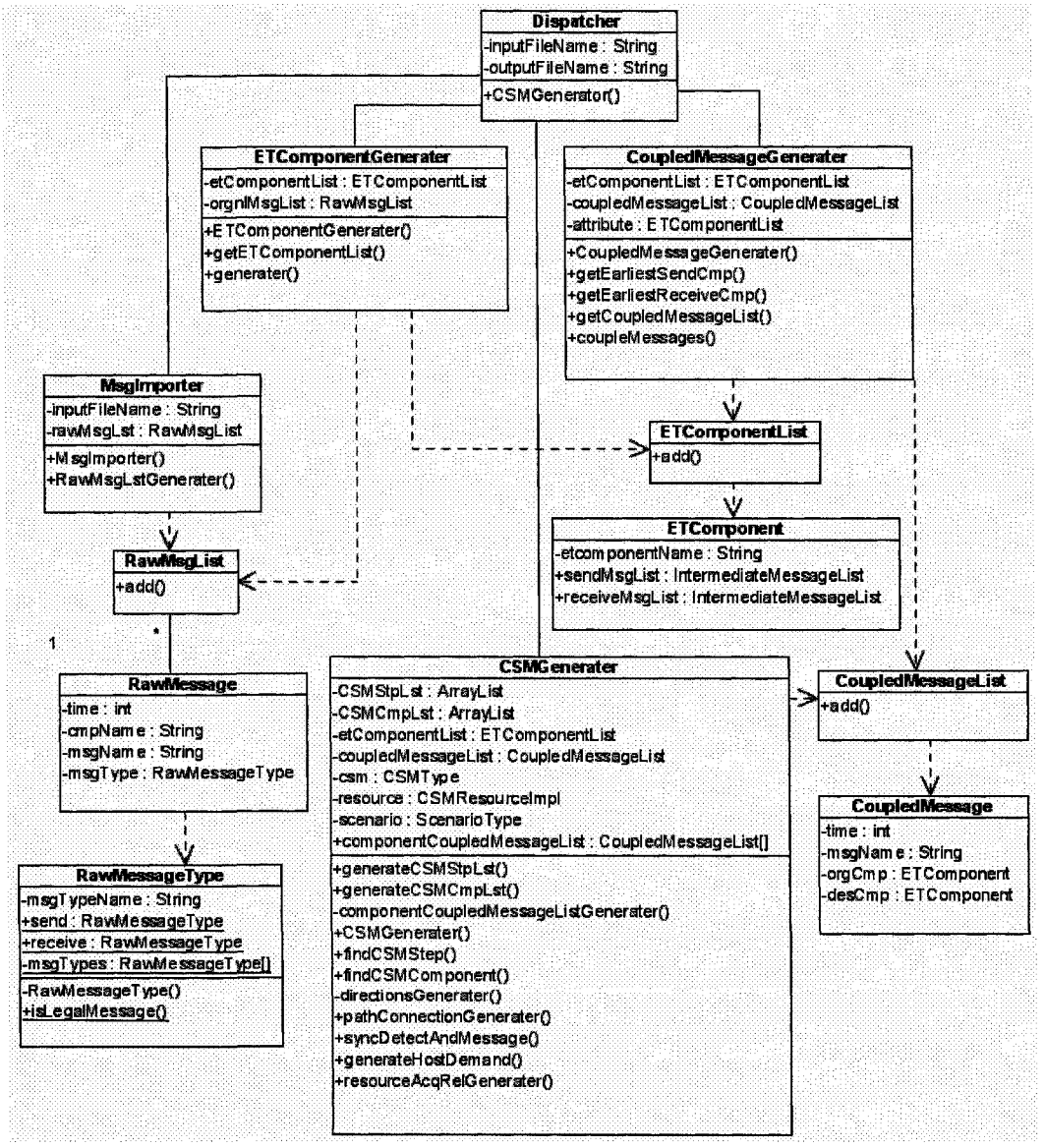



**Figure 3-3 Class Diagram of Transformation Program**

## *3.4 Preprocess*

### 3.4.1 Import Execution Traces

A Java class, RawMessage, is used to hold execution records without changing message content. This class has four attributes corresponding to the four fields in execution traces:

- time stamp: integer
- message type: only send and receive
- component name: string
- message name: string

Also, this phase will check if given records follow the above defined data types.

### 3.4.2 Generate ETComponent Instances

A Java class, ETComponent, is used to hold component names, and components' sorted sending and receiving message lists. It has three attributes:

- component name: string
- sending message list : IntermediateMessageList
- receiving message list: IntermediateMessageList

Instances of IntermediateMessageList are used to hold intermediate messages, which only contain time stamp and message names obtained from the first phase, i.e., subsection 3.4.1. Sending and receiving message lists only hold sending and receiving messages respectively, and messages are sorted according to time. This class will be used to couple messages.

### 3.4.3 Coupling Trace Records

A Java class, CoupledMessage, is used to hold messages with both sending component and receiving component. These coupled messages are the bases to generate instances of CSM elements. It has five attributes:

- time: integer
- message name: string
- originated component: ETComponent
- destination component: ETComponent
- message type: string with a value of "sync", "async", "reply" and "unknown" (this attribute is used to indicate message types, see Section 3.7 for details)

In order to know how components interact we need messages containing both sending and receiving component names. However, the execution record does not contain interactive component name, therefore we have to couple two trace records to a single message that satisfy this requirement. Class CoupledMessage is used for this purpose.

Figure 3-3 shows how two trace records are coupled into one single message represented by the named arrow Book_START from component Server at time 9926267 to Book136228 at time 9926465. The trace records are the first two records in Table 3-1. The diagram is similar to a UML sequence diagram. In such a diagram, a named rectangle represents a component, like component Server and Book136228; messages are shown as arrows between two components; doted lines mean time flows from top to bottom, that is, lifelines in UML sequence diagram. Similar diagrams will be used for later analysis involving components and steps between two components.

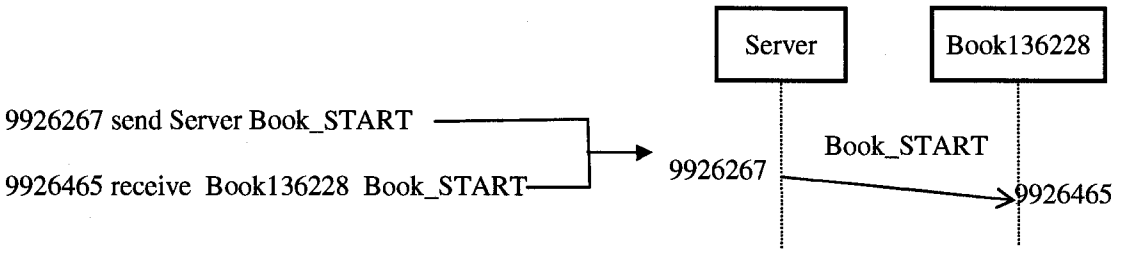



**Figure 3-4 Coupling Two Trace Records to a Step**

### *Message Coupling Algorithm*

Two possible methods can be used to couple two execution records, namely, using message names or timestamps to identify messages. However, neither method is practical. Message names occur repeatedly throughout a trace, so unique methods can not be found directly. Sending and receiving timestamp often differ because of latency. Therefore, an assumption has to be made, that is, a message sent first will be received first for all the components. Based on this assumption, following algorithm will couple the earliest sending message and the earliest receiving message into one instance of CoupledMessage.

**Algorithm 3-3 Message Coupling**

**Assumption:** A message sent first will be received first for all the components. This assumption needs a global clock to generate time so as to make message timestamp comparable.

**Input:** A list of ETComponent instances *etComponentList* (output from Subsection 3.4.2), and an empty CoupledMessage list, *coupledMessageList*.

**Output**: A list of CoupledMessage instances, *coupledMessageList*

1. while one of the sending message list of all the components is not empty
    1.1. Find the earliest sending message *earliestSendingM* from sending message lists of all the components, and its component (instance of ETComponent) *SendingETC*.
    1.2. Find the earliest receiving message *earliestReceivingM* from receiving message lists of all the components, and its component (instance of ETComponent) *ReceivingETC*.
    1.3. If message names of *earliestSendingM* and *earliestReceivingM* are same:
        1.3.1. Create an instance of CoupledMessage *coupledMessage*, and add it to *coupledMessageList*.

1.3.2. Set message name of *coupledMessage* to be message name of *earliestSendingM.*

1.3.3. Set originated component of *coupledMessage* to be *SendingETC.*

1.3.4. Set destination component: of *coupledMessage* to be *ReceivingETC.*

1.3.5. Set time of *coupledMessage* to be the timestamp of *ReceivingETC.*

1.3.6. set message type of *coupledMessage* to "unknown"

1.4. else

1.4.1. print out warning information

1.5. Remove *earliestSendingM* and *earliestReceivingM* from message lists

2. Return *coupledMessageList*

### *Algorithm Assumption*

The assumption for the above algorithm is "a message sent first will be received first for all the components". The following example shows that if this assumption is violated, i.e. messages are received out-of-order from messages sent coupled messages will contain incorrect sending and receiving components.

In the true scenario shown in Figure 3-4 (1), component A sends a message *getName* to B at time 1, while B receives the message at time 4. Component C sends a message with the same name to D at time 2, while D receives the message at time 3. However, algorithm 3.2 generates two coupled messages in (2), which is different from the actual scenario. The reason is that A is the first sending component, but D, not B, is the first receiving component. Since the message names are same (*getName*), the execution records of A and D are coupled into one instance of CoupledMessage. The reason is the same for component for B and C.

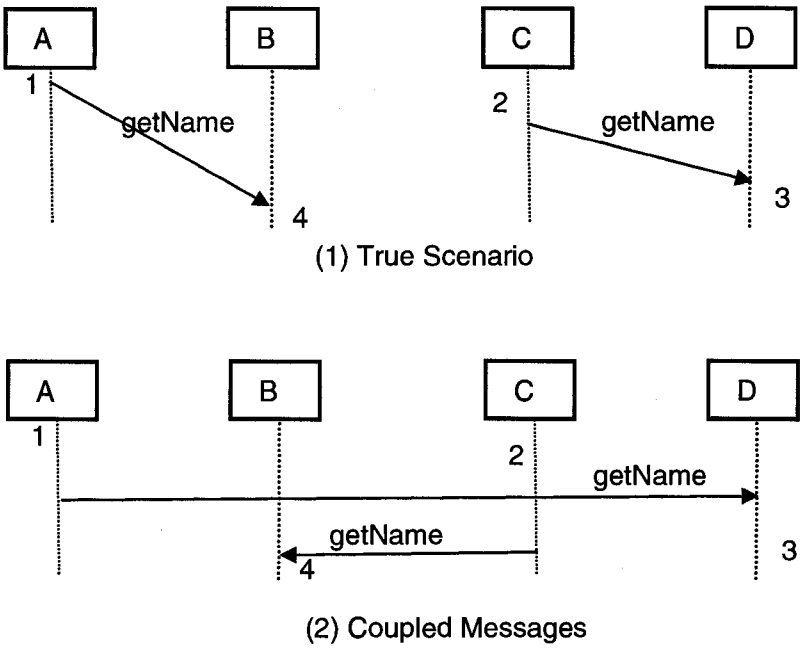



**Figure 3-5 Problem of Eliminating the Assumption for Coupling Traces**

If we include the name of the receiving components in the execution traces of sending message events, this problem can be eliminated and coupling messages will be unnecessary.

## *3.5 Generate Instances of CSM Component and Step*

Because there is no information about devices such as CPUs and disks in execution traces, instances of ProcessingResource, a subclass of ActiveResource (which in turn is a subclass of GeneralResource), can not be created. Similarly, instances of ExternalOperation, another subclass of ActiveResource, can not be created either. This can be an area for future research. Therefore, only instances of component, a subclass of PassiveResource, can be derived from the trace information. Components represent abstract entities such as: actors, objects, resources, processes, containers, and so on.

An instance of Component has associations with instances of CSM, ResourceAcquire, ResourceRelease, Step, ProcessingResource and Component itself. A Component instance needs one ProcessingResource as its host, and one Component may have multiple sub-components or may have one parent-component. However, because we can not generate instances of ProcessingResource from execution traces, and there is no information about relationships among components, these two associations will be excluded.

A Step describes the processing tasks (e.g. procedures, functions actions, etc.) that a system must perform. Instances of Step, PathConnection and Classifier compose an instance of Scenario. Scenario and Component instances will compose an instance of CSM. Because both Step and Component are extractable classes, the following algorithm will create their instances at the same time. Associations between Component and Step, Step and Scenario, Scenario and CSM, Component and CSM will be implemented in this section.

**Algorithm 3-4 Component and Step Extraction Algorithm**

**Input:** A list of CoupledMessage instances *coupledMessageList* (output from Algorithm 3-2)

**Output**: An instance of CSM, *csm*, composed of instances of Component and an instance of Scenario, which contains instances of Step

1. Create an instance *csm* of CSM
2. Create an instance *scenario* of CSM class Scenario
3. Add *scenario* to *csm* (implement associations between Scenario and CSM)
4. For each *coupledMessage* in *coupledMessageList*
    4.1. Create an instance of CSM Step *step*
    4.2. Set the name of *step* to the message name of *coupledMessage*
    4.3. Get the name *name* of destination component of *coupledMessage*
    4.4. If a CSM Component *component* with name *name* does not exist
        4.4.1. Create an instance of CSM Component *component*

4.4.2. Set the name of *component* to be *name*

4.4.3. Add *component* to *csm* (implement associations between Component and CSM)

4.5. Set the value of Component of *step* to *component*. (implement association between Step and Component)

4.6. Add *step* to *scenario* (implement associations between Step and Scenario)

## 3.6 Generate Instances of PathConnection

A PathConnection element is an abstract class for the connections between two steps of a scenario. Its subclasses, Start, End, Sequence, Fork, Join, Branch and Merge, describe different connection types between two steps as shown in subsection 2.5.1. A Branch (Or Fork) can be explicitly depicted with conditions in a design model, such as an UML activity diagram, but it cannot be recorded in a trace file because only one condition can be true during an execution of the design. Therefore, for execution traces we can not deduce a Branch, but only a Fork (And Fork). Similarly, we can not deduce a Merge (Or Join) from the execution traces, but only a Join (And Join).

In Algorithm 3-3 a Step instance *step* is created when we encounter one CoupledMessage instance *CoupledMessage*, this means a Step instance always has a corresponding CoupledMessage instance. Logically related messages are a group of CoupledMessage instances whose corresponding Step instances can be connected by a PathConnection object.

### *Definitions of Logically Related Messages*

For a group of messages, if they satisfy the following conditions we call them logically related messages.

- ***Share Same Components:*** If two messages do not share the same component the corresponding Step instances can not be connected by a single PathConnection object though there may exist a sequence of PathConnectoins and Steps that connect them. In Figure 3-5, because message 1 and 2 do not share any components no PathConnection instance can be deduced.

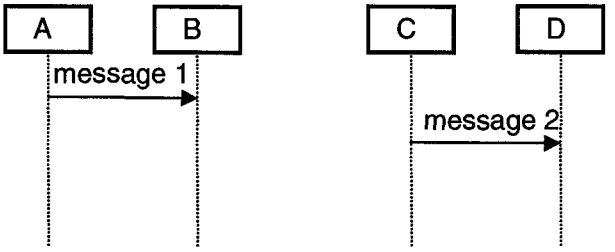



**Figure 3-6 No Shared Component between Two Messages**

- ***Consecutive***: If two or more messages are not consecutive from the point of the shared component, no PathConnection instances will be generated. In Figure 3-6, though message 1 and 3 share component A, their corresponding Step instances can not be connected by a single PathConnection because they are not consecutive from the point of view of A. However, there is an exception: for several consecutive sending messages a Fork object needs to be generated though the first message is not consecutive with the last one. For consecutive receiving messages, a Join object needs to be generated.

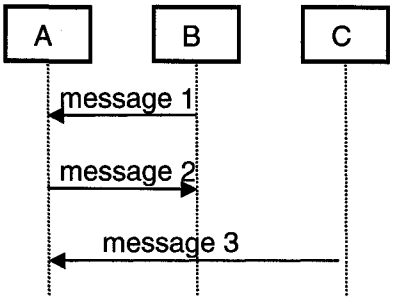



**Figure 3-7 Consecutive Messages**

- ***First Message is A Receiving Message:*** A receiving message of a Component can originate the occurrences of consecutive sending messages of this Component, but a

sending message can not. This means, if two messages share a component and are consecutive in time, there may not exist a PathConnection object, if from the view of the shared component it receives a message after it sends another message. In the following example, message 1 and 2 share Component B, however, message 1, the sending messages occurs first, it can not lead to the occurrence of message 2, so there is no PathConnection between corresponding Step instances.

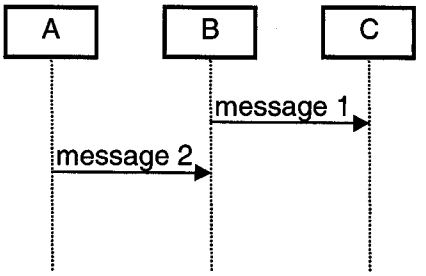



**Figure 3-8 Sending before Receiving: No PathConnection**

### *Deducible Pattern*

Logically related messages mean we can deduce PathConnection instances for one specific pattern, that is, a Component instance receives one or more message and then sends one or more messages. Figure 3-8 is such an example. Message 1, a receiving message for B and message 2, a sending message for B are time consecutive, therefore we can say there is a Sequence instance between messages' corresponding step objects, Step 1 and 2. However, how to deal with other patterns? The answer is we do not need to handle other patterns. The reason will be explained in the following subsections.

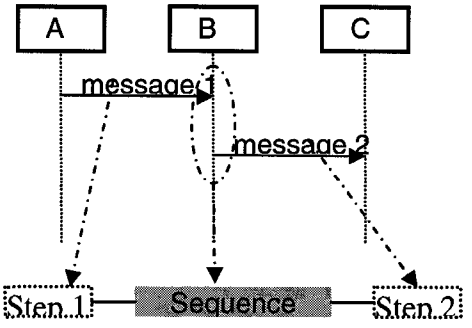



**Figure 3-9 Deducible Pattern**

### 3.6.1 Completeness of Logically Related Messages

According to the CSM meta-model in Figure 2-6, a Step instance must have exactly one predecessor and one successor, while a Pathconnection instance can be associated to multiple sources and target Steps. In Figure 3-9 a predecessor connects a Step with its consecutively preceding Step, while a successor connects it with its consecutively successive Step. Steps including ResourceAcquire and ResourceRelease can be associated to any PathConnection instance as their successor or predecessor.

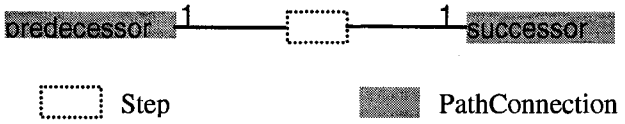



**Figure 3-10 Step's Predecessor and Successor PathConnection**

The way to differentiate instances of different subclasses of PathConnection is to check the number of source and target Steps of a PathConnection as shown in Figure 3-10. If a PathConnection has no source Step, it is a Start; if it has no target Step, it is an End; if it has one source and one target, then it is a sequence. If two or more Steps share a predecessor that has only one source, then this PathConnection instance is a Fork; if two or more Steps share a successor that has only one target, then the PathConnection is a Join.

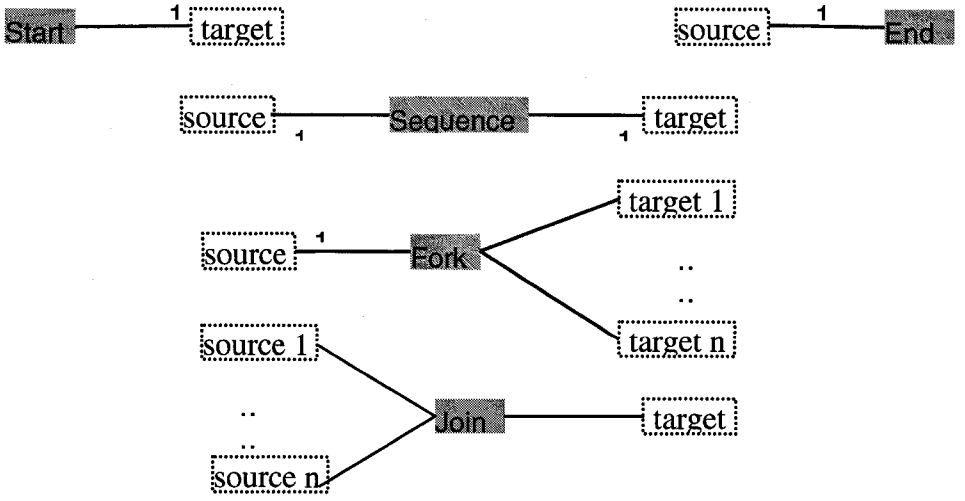



**Figure 3-11 PathConnection's Source and Target Step**

### *Mutual Successor and Predecessor PathConnection*

Combing Figure 3-9 with Figure 3-10 the successor of one Step instance can be the predecessor of its successive step. For example, in Figure 3-11 PathConnection 2 is the successor of Step A and the predecessor of Step B.

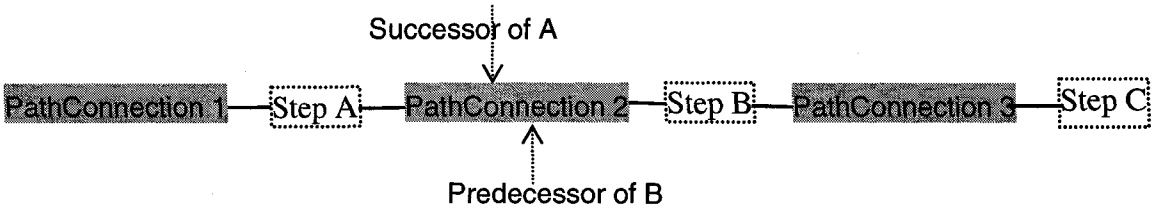



**Figure 3-12 Mutual Successor and Predecessor PathConnection**

In Figure 3-11, PathConnection 2 is deduced from logically related messages corresponding to Step A and B, but PathConnection 3 as Step B's Successor can not be deduced from the same message group. However, it can be deduced from logically related messages corresponding to Step B and C, here it is the Predecessor of C. Therefore, we say logically related messages have completeness, which means if all the message groups (logically related messages) with respect to the pattern in Figure 3-8 can be found then all the PathConnection instances can be deduced from these message groups.

## 3.6.2 PathConnection Instances Deduction Algorithm

This algorithm finds all the logically related messages (subsection 3.6.2), and, according to the number of receiving and sending messages (Figure 3-10), generates corresponding PathConnection instances.

- If the first CoupledMessage of an ETcomponent is a sending message a Start instance needs to be created.

- If the last CoupledMessage of an ETcomponent is a receiving message an End instance needs to be created.
- For instances of CoupledMessage that correspond to logically related messages, such as Step 1 and 2 in Figure 3-8:
    - If a sending message or no messages is ahead of them, and they are followed by one receiving message or no messages, a Sequence instance needs to be created.
    - If they are followed by one or more sending messages, a Fork instance need to be created.
    - If one or more receiving messages are ahead of them, a Join instance needs to be created.

**Algorithm 3-5 PathConnection Instances Deduction Algorithm**

**Input:** An instance of CSM, *csm*, (output from Algorithm 3-3), and a list of CoupledMessage instances *coupledMessageList* (output from Algorithm 3-2)

**Output**: An instance of CSM, *csm*, composed of instances of Component and an instance of Scenario, which contains instances of Step connected by PathConnection instances

1. Preprocess
    1.1. Generate an array *coupledMessageListArray,* each of its elements is an ArrayList object *coupledMessageList* containing CoupledMessage instances for one ETComponent object.
    1.2. For each *coupledMessageList* in *coupledMessageListArray*, generate a string *direction* containing only "0" and "1". When a receiving message is found, a "0" is appended to *direction*; when a sending message is found, a "1" is appended to *direction*. (0 and 1 indicates the message type, sending or receiving, of CoupledMessage instances for each ETComponent)
2. Generate Start instances

2.1. For each *direction* starts with "1" (a sending message)

2.1.1. Find its corresponding *coupledMessageList* object and the first CoupledMessage object, *firstCoupledMessage*, in *coupledMessageList*

2.1.2. Find corresponding Step object *step* of *firstCoupledMessage*

2.1.3. Create a Start instance *start*

2.1.4. Add *start* to *scenario* (implement associations between Start and Scenario)

2.1.5. Set predecessor of *step* to be *start* (implement associations between Start and Step)

2.1.6. Add *step* to *start*'s target list (implement associations between Start and Step)

3. Generate End instances

3.1. For each *direction* ends with "0" (a receiving message)

3.1.1. Find its corresponding *coupledMessageList* object and the last CoupledMessage object, *lastCoupledMessage*, in *coupledMessageList*

3.1.2. Find corresponding Step object *step* of *lastCoupledMessage*

3.1.3. Create an End instance *end*

3.1.4. Add *end* to *scenario* (implement associations between End and Scenario)

3.1.5. Set successor of *step* to be *end* (implement associations between End and Step)

3.1.6. Add *step* to *end*'s source list (implement associations between End and Step)

4. Generate Fork instances

4.1. For each substring *forkPattern* in *direction* with pattern 011+ (a receiving message followed by two or more sending messages.)

4.1.1. Find its corresponding *coupledMessageList* object and corresponding CoupledMessage group, *coupledMessageGroup*, in *coupledMessageList* according to *forkPattern's* position in *direction*

4.1.2. Find corresponding Step object *source* of the first CoupledMessage object in *coupledMessageGroup*

4.1.3. Create a Fork instance *fork*

4.1.4. Add *fork* to *scenario* (implement associations between Fork and Scenario)

4.1.5. Set successor of *source* to be *fork* (implement associations between Fork and Step)

4.1.6. Add *source* to *fork*'s source list (implement associations between Fork and Step)

4.1.7. For each "1" in *forkPattern*

4.1.7.1. Find corresponding Step object *branch* of CoupledMessage object in *coupledMessageGroup* according to 1's position in *coupledMessageGroup*

4.1.7.2. Set predecessor of *branch* to be *fork* (implement associations between Fork and Step)

4.1.7.3. Add *branch* to *fork*'s target list (implement associations between Fork and Step)

5. Generate Join instances

5.1. For each substring *joinPattern* in *direction* with pattern 00+1+ (two or more receiving messages followed by one sending message)

5.1.1. Find its corresponding *coupledMessageList* object and corresponding CoupledMessage group, *coupledMessageGroup,* in *coupledMessageList* according to *joinPattern's* position in *direction*

5.1.2. Create a Join instance *join*

5.1.3. Add *join* to *scenario* (implement associations between Join and Scenario)

5.1.4. For each "0" in *joinPattern*

5.1.4.1. Find corresponding Step object *branch* of CoupledMessage object in *coupledMessageGroup* according to 0's position in *coupledMessageGroup*

5.1.4.2. Set successor of *branch* to be *join* (implement associations between Join and Step)

5.1.4.3. Add *branch* to *join*'s source list (implement associations between Join and Step)

5.1.5. Find corresponding Step object *target* of the last CoupledMessage object in *coupledMessageGroup*

5.1.6. Set predecessor of *target* to be *join* (implement associations between Join and Step)

5.1.7. Add *target* to *join*'s target list (implement associations between Join and Step)

6. Generate Sequence instances

6.1. For each substring *sequencePattern* in *direction* with pattern 1?010? (one receiving message followed by one sending message)

6.1.1. Find its corresponding *coupledMessageList* object and corresponding CoupledMessage group, *coupledMessageGroup,* in *coupledMessageList* according to *sequencePattern's* position in *direction*

6.1.2. Create a Sequence instance *sequence*

6.1.3. Add *sequence* to *scenario* (implement associations between Sequence and Scenario)

6.1.4. Find corresponding Step object *source* of the first CoupledMessage object in *coupledMessageGroup*

6.1.5. Set successor of *source* to be *sequence* (implement associations between Sequence and Step)

6.1.6. Add *source* to *sequence*'s source list (implement associations between Sequence and Step)

6.1.7. Find corresponding Step object *target* of the last CoupledMessage object object in *coupledMessageGroup*

6.1.8. Set predecessor of *target* to be *sequence* (implement associations between Sequence and Step)

6.1.9. Add *target* to *sequence*'s target list (implement associations between Sequence and Step)

### 3.6.3 Proof of "One Step Instance has Exactly One Predecessor and One Successor"

According to the CSM meta-model in Figure 2-6, a Step instance must have exactly one predecessor and one successor. When algorithm 3-4 generates PathConnection instances five different patterns of message direction are checked for each ETComponent:

| | | | |
|---|---|---|---|
| 1. | 1(0\|1)* | Start | algorithm 3-4 2 |
| 2. | (0\|1)*0 | End | algorithm 3-4 3 |
| 3. | 011+ | Fork | algorithm 3-4 4 |
| 4. | 001+ | Join | algorithm 3-4 5 |
| 5. | 1?010? | Sequence | algorithm 3-4 6 |

It is possible that the same 0 or 1 can be found in two different matches of different patterns; this is called an "Overlapping Match". For example, in string "0110", matches for case 2, (0|1)*0, and case 3, 011+, can be found. For case 2, the matched string is "0110"; and for case 3, the matched string is 011 (the first three characters of string "0110"). However, when this situation occurs, the PathConnection objects generated are different: for case 2, an End instance will be created as the last 0's corresponding Step object's successor; and for case 3, a Fork instance will be created as the 0's correspond to the Step object's successor and the two 1s' correspond to the Step object's predecessor. Similarly this also applies to all the

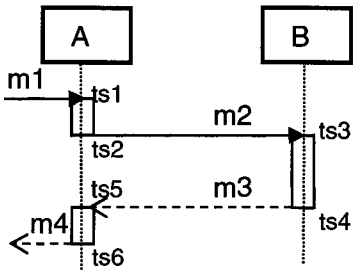
overlapping matches between any two of the five patterns. Therefore, algorithm 3-4 guarantees that each Step instance only gets one chance to generate one predecessor and one successor.

## 3.7 Detect Synchronous Communications and Generate Instances of Message

Three communication patterns may occur during the interactions between tasks or components, which are synchronous, asynchronous and forwarding communication patterns.

### Synchronous Communication Pattern

In a synchronous communication shown in the sequence diagram Figure 3-12, client component A is blocked after sending *m2* until it receives the reply *m3*. The *ts* in the figure denotes the timestamp at which a certain event happens. For example, A sends *m2* at *ts2*; B process the request from *ts3* to *ts4*. The service demand for B to process request *m2* is therefore (t4-t3), and the service demand for A to process request *m1*is (ts2-ts1) + (ts6-ts5).




ts timestamp

**Figure 3-13 Synchronous Communication Pattern**

### Asynchronous Communication Pattern

An asynchronous communication is shown in Figure 3-13. Component A sends message *m2* at time *ts0* and continues in parallel with component B. The service demand for B to process *m2* is (t4-t3) and the service demand for A to process request *m1* is (ts2-ts1).

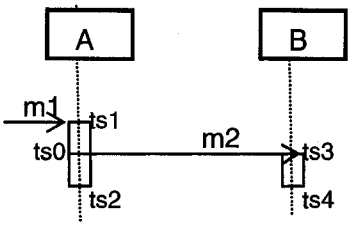



**Figure 3-14 Asynchronous Communication Patter**

### *Forwarding Communication Pattern*

A forwarding communication is shown in Figure 3-14. Component A sends a request *m2*, denoted by a solid arrow with a line arrowhead, and will be blocked until a reply is received. When B receives the request *m2*, it will forward its reply as request *m3* to C. After C processes the request *m3,* it sends reply *m4* to A, and then A will be unblocked. The service demand for A to process request *m1* is (ts0-ts1) + (ts7-ts6); the service demand for B to process *m2* is (t3-t2); the service demand for C to process *m3* is (t5-t4).

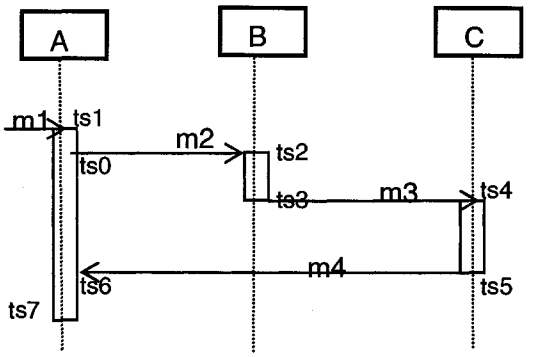



**Figure 3-15 Forwarding Communication Patter**

## 3.7.1 Detecting Synchronous Communications

In the CSM, three message types are defined: *async*, *sync* and *reply*. *Sync* means the message is a synchronous request message, like *m2* in Figure 3-12 while *reply* is a reply message to a synchronous request, like m3 in Figure 3-12. Therefore, *sync* and *reply* messages always occur in pairs. *Async* means the message is an asynchronous request message, like *m2* in Figure 3-13. There is no forwarding message type is CSM model. The messages involved in a forwarding communication will be treated as asynchronous,

synchronous, and reply messages. Therefore, if we can find all the synchronous communications, we will find all the *sync* and *reply* messages, thus all the other messages are *async* messages.

In order to detect synchronous communications, the following two assumptions are needed.

### *Assumption 1 Determining Synchronous Communications*

From the output of Algorithm 3-2, a list of CoupledMessage instances, the message types are unknown even in the situation where two messages like *m2* and *m3* in Figure 3-12 are present, because they may form a synchronous communication, thus *m2* is *sync* and *m3* is *reply* message, or they may form two asynchronous communications, thus both *m2* and *m3* are *async* messages.

Therefore, if two messages *m1* and *m2* satisfy the following conditions, they form a synchronous communication where *m1* has type *sync* and *m2* has type *reply:*

1. *m1* belongs to a group of sending messages *sendingGroup* of component A. Messages in this group are time consecutive. This group may only contain *m1. m1* is sent from component A to B.
2. *m2* belongs to a group of receiving messages *receivingGroup* of component A. Messages in this group are time consecutive. This group may only contain *m2. ReceivingGroup* follows *sendingGroup* in terms of time. *m2* is sent from component B to A.
3. *m1* is not a reply message from A to B.

Figure 3-15 is an example that explains assumption 1. Stealth arrowheads are used to show that the message types are unknown. *m1* and *m3* form a sending message group for

component A, while *m2* and *m4* form a receiving message group for A, which follows the sending message group. *m1* is sent from A to B, while *m2* is sent from B to A. And *m1* is not a reply message from A to B because it is the first message between these two components. Therefore we assume that *m1* and *m2* form a synchronous communication. In this example, it is very easy to determine that *m1* is not a reply message from A to B, however, for general cases another assumption is needed.

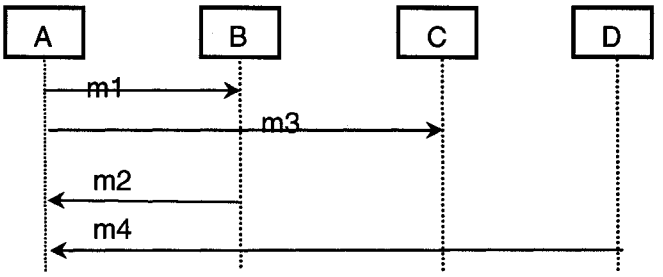



**Figure 3-16 Assumption 1 for Detecting Synchronous Communications**

### *Assumption 2 Determining Reply Message*

In Figure 3-16, message *m1* can be a reply message to component B as (2) shows, or it can be a synchronous request message to B as (3) shows. In this situation, we will consider (2) as the real case in design, because in a client-server system, a pure client will send a request to a server, then the server will do the task and reply to the client, and then a client can have another request, it is exactly (2) describes. In (3), however, B asks A to process request *m0*, which means A is a server, then A asks B to process *m1*, and later B reply to A via *m2*, this means B can not be a pure client, but needs to act as a sever for A, this will make design complicated. If we have two designs completing the same task, we will choose the simple one, that is (2) in this example.

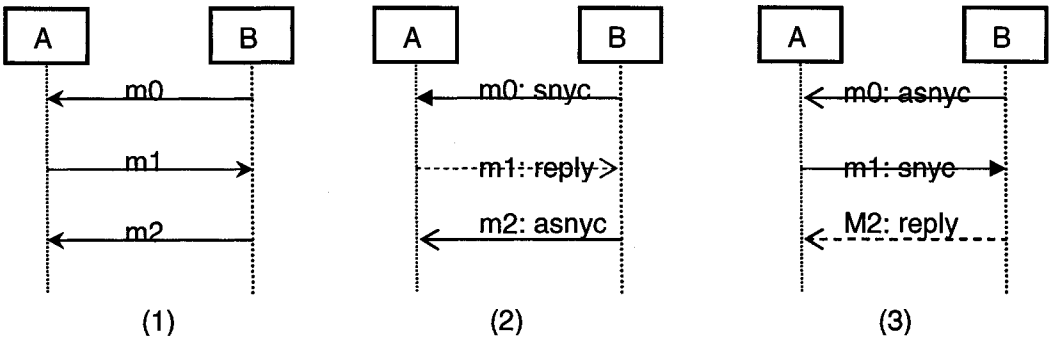



**Figure 3-17 Controversial Synchronous Communications**

### 3.7.2 Synchronous Communications Detection and Message Instances Generation Algorithm

To find synchronous communications the algorithm loops through the CoupledMessage list for each ETComponent, and finds matches of the pattern defined in assumption 1 of subsection 3.7.2. If it can not determine if a message is a reply message or a synchronous request message as described in assumption 2, it will abandon the current match and continue searching for matches in the current ETComponent message list. The discarded match will be taken care of by the message receiving component.

In one situation a non-reply message is determinative: the previous message is not received from the same component, or there is no previous message like *m1* in figure 3-15. If the message type has been determined when the algorithm examine other components, the current match will be discarded too. If a match is found and the sending message of the current ETComponent is not a reply message to the message receiving ETComponent like *m1* and *m2* in Figure 3-15, then a synchronous communication is detected. The first message will be assigned a type of *sync*, and the second one will be a *reply* message, and they will be added to the classifier list of corresponding PathConnection instances. When the algorithm finishes looping all the ETComponent objects, all the message without any type are *async* messages.

Take Figure 3-16 as an example, if the algorithm first examines component A, it will find *m1 and m2* match assumption 1, however, it can not determine if *m1* is a reply message from A to B, so it discards the match. Because there are no other matches for component A, the algorithm will then examine component B. It will find a pattern match composed of *m0* and *m1*, and because *m0* is the first message for B, it will determine that *m0* and *m1* belong to a synchronous communication. However, if the algorithm first examines component B, it will find that *m0* and *m1* form a synchronous communication, thus, when the algorithm examines component B, even though it will find a match of *m1* and *m2*, it will not consider it is a candidate for a synchronous communication, because *m1* has type *reply* now.

**Algorithm 3-6 Synchronous Communications Detection and Message Instances Generation Algorithm**

**Input:** An instance of CSM, *csm,* (output from Algorithm 3-4), and a list of CoupledMessage instances *coupledMessageList* (output from Algorithm 3-2)

**Output:** An instance of CSM, *csm,* composed of instances of Component and an instance of Scenario that contains instances of Step connected by PathConnection instances, which have been assigned message types of *sync, async* and *reply.*

1. Preprocess: same as the preprocess in Algorithm 3-4
2. For each message group *potentialSync* with pattern "1+0+1*" of each *direction* (one or more sending messages followed by one or more receiving messages, which can be followed by zero or more sending messages)
   2.1. Find sending message group *sGroup*, and exclude all the messages with type *reply.*
   2.2. Find receiving message group *rGroup*, and exclude all the messages with type *sync.*
   2.3. For each message *sMessage* in *sGroup*
      2.3.1. For each message r*Message* in R*Group*
         2.3.1.1. set *isSyncFound* to be false

2.3.1.2. If *sMessage* is sent from component A to B, and *rMessage* is sent from B to A.

2.3.1.2.1. if *sMessage* is the first message for B

2.3.1.2.1.1. set *isSyncFound* to be true

2.3.1.2.2. else

2.3.1.2.2.1. if B's last message is receiving message

2.3.1.2.2.1.1. set *isSyncFound* to be true

2.3.1.2.2.2. else

2.3.1.2.2.2.1. Find all the B's sending messages consecutive in time earlier than *sMessage* and store in *bSends*

2.3.1.2.2.2.2. if no message in *bSends* is sent from B to A

2.3.1.2.2.2.2.1. set *isSyncFound* to be true

2.3.2. if *isSyncFound* is true

2.3.2.1. set type of *sMessage* to be *sync*, delete it from *sGroup*

2.3.2.2. set type of *rMessage* to be *reply*, delete it from *rGroup*

2.3.2.3. find corresponding PathConnection *sPath* and *rPath* instances from *CSM* for *sMessage* and *rMessage*

2.3.3. add *sync* to *sPath's* classifier list (implement associations between Classifier and PathConnection)

2.3.3.1. add *reply* to *rPath's* classifier list (implement associations between Classifier and PathConnection)

3. For each CoupledMessage instance, if its message type is *unknown*, add *async* to the classifier list of its corresponding PathConnection instance.

## *3.8 Calculate hostDemand Value for Each Step Instance*

Figure 3-12 shows how to calculate hostDemand for request *m2*, a synchronous request, and Figure 3-13 shows how to calculate hostDemand for request m2, an asynchronous request. From Section 3.7 we have found all the synchronous and asynchronous communications, so we can now calculate hostDemand for each corresponding Step instance. However, the execution traces we can not find all the timestamps needed to calculate hostDemand for an asynchronous communication. For example we do not know what *ts4* is for request *m2* in Figure 3-13, and we do not know what *ts2* is for request *m1*. Also we need to know how to calculate hostDemand if there is a nested communication, like the synchronous communication composed of *m2* and *m3* nested in the one composed of *m1* and *m4* in Figure 3-12. This is covered in this section.

### 3.8.1 Single Synchronous Request

A single request means that after a server component receives a request; it will not send or receive any other requests until it finishes the current request, like *m2* in Figure 3-12 and *m2* in Figure 3-13. A single synchronous request means the server component keeps busy until it sends reply to the client component. In this situation, the hostDemand is the service time between component's receiving request and sending reply, that is, *(ts4-ts3)* for Figure 3-13.

### 3.8.2 Single Asynchronous Request

As we mentioned above, we do not know when an asynchronous request is finished. The worst case is, it will continue working until it sends a request to other components, or it receives a new request like the request *m2* in Figure 3-17. After component B receives request *m2* and process *m2* for *(t4-t3)*, it sends request *m3* to component C. Therefore, the longest time for component B to deal with request m2 is *(t4-t3)*. The best case is component B spends a very limited time on *m2*, and this time can be ignored. Therefore, any value

between 0 and *(t4-t3)* can be the hostDemand. Here, we use *(t4-t3)*, the value from worst cases as hostDemand for request *m2*.

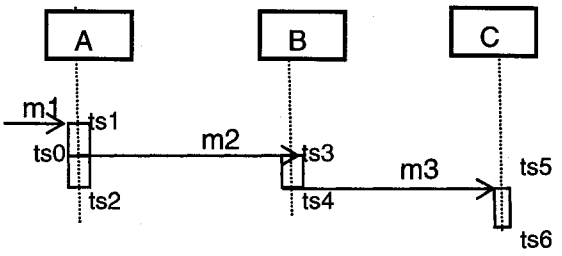



**Figure 3-18 hostDemand for Single Asynchronous Request**

### 3.8.3 Nested Communications

Nested communication means that when the server component is processing a request it sends requests to other components. Figure 3-12 is an example of nested synchronous communication, where a synchronous request *m2* is sent to component B when A is processing a synchronous request *m1*. Because after A sends *m1* to B it will be blocked, the blocked time, that is *(t5-t2)* for Figure 3-12, needs to be deducted from total time, therefore, the hostDemand from request *m1* is *(ts6-ts1) – (ts5-ts2)*. If the nested communication is an asynchronous request, like m2 in Figure 3-17, because the client component A will not be blocked, so the total hostDemand is the same for single asynchronous or synchronous requests. To summarize, for a nested communication, the hostDemand for a server component is the service time as there is no nested communication minus the blocking time, that is, the time between it sends synchronous request and receives reply.

### 3.8.4 Nested Communications with Join and Fork

It is possible that a server component sends multiple requests to other components when it is processing a request. Take Figure 3-18 as an example. After receiving request *m1*, component A sends two synchronous request *m2* and *m3* to component B and C respectively, and it receives two replies from them later. The service time for A to deal with request A should contain the time between two requests, *m2* and *m3*, that is, *(ts3-ts2)* and

contain time between two replies *m4* and *m5*, that is, *(t9-t8)*, because only after sending *m3* to component C it is blocked, and for the worst case when it receives *m4*, it will continue processing *m1* until *ts9*. Therefore, the hostDemand for A to process *m1* is (ts10-ts1) – *(ts8-ts3)*.

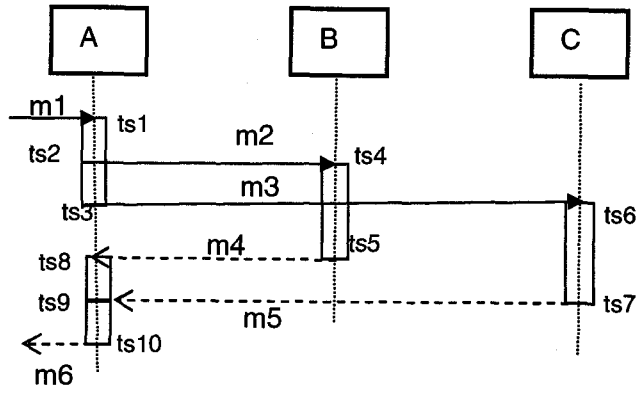



**Figure 3-19 Nest Communications with Fork and Join**

### 3.8.5 hostDemand Calculation Algorithm

For each ETComponent instance, the algorithm will go through all of its request messages, that is, CoupledMessage instances that are incoming messages with type *sync* and *async*. For each request, we will check if it is followed by one or a group of sending messages (fork), if so the formulas in subsection 3.8.2, 3.8.3 and 3.8.4 are used to calculate hostDemand; if not, it means the component receives multiple requests at the same time, and the hostDemand will be calculated for each request separately.

**Algorithm 3-7 hostDemand Calculation Algorithm**

**Input:** An instance of CSM, *csm,* (output from Algorithm 3-5), and a list of CoupledMessage instances *coupledMessageList* (output from Algorithm 3-2).

**Output**: *csm* with hostDemand for each step

1. Preprocess: same as the preprocess in Algorithm 3-4
2. for each substring *incomingMPattern* of each *direction* with pattern 0+
   2.1. if the corresponding CoupledMessage instance *syncRequest* has type *sync*
      2.1.1. if *syncRequest* is followed by one or group outgoing *sync* or *async* messages

2.1.1.1. find *syncRequest*'s corresponding (nearest) *reply* message *syncReply* and their timestamps *syncRequestTime* and *syncReplyTime*

2.1.1.2. find all the *sync* messages between *syncRequest* and *syncReply* *syncBetween* and their corresponding reply messages *replyBetween*, and their timestamps *syncBetweenTime* and *replyBetweenTime*

2.1.1.3. hostDemand for *syncRequest*'s corresponding Step instance is *(syncReplyTime - syncRequestTime) – (∑replyBetweenTime - ∑syncBetweenTime)*

2.1.2. else

2.1.2.1. for each incoming *sync* and *async* message remove all other messages and their corresponding reply messages if applicable in the incoming message group, and invoke 2.1

2.2. else if the corresponding CoupledMessage instance *asyncRequest* has type *async*, find its timestamp *tBegin*; set *deduction* to be 0; set *currentMessage* to be *asyncRequest*'s next message

2.2.1. loop

2.2.1.1. if *currentMessage* is outgoing *sync* message *outSync*, find its timestamp *tOutSync*, and timestamp of its reply message *tOutReply; deduction = deduction+ (tOutReply- tOutSync)*

2.2.1.2. else if *currentMessage* is incoming *reply* message, do nothing

2.2.1.3. else

2.2.1.3.1. find timestamp *tCurrentMessage* of *currentMessage*

2.2.1.3.2. set *hostDemand* of *asyncRequest* to be *(tCurrentMessage-tBegin) – deduction*

2.2.1.3.3. exit loop

## *3.9 Generate ResourceAcquire and ResourceRelease Instances*

For a synchronous request, a resource, that is, a server component in the transformation from execution traces to CSM models, has to be acquired, and the reply message will release the resource. For an asynchronous request, similarly, a server component has to be acquired before the request can be processed, and is released right after the request is satisfied. Therefore, for each CoupledMessage instance of each ETComponent object, we will first check if it is a request to the component. If so we will generate and insert ResourceAcquire instances before corresponding Step objects. If the request is synchronous request, we will find its reply message and its corresponding Step object, and then change this object to be of type ResourceRelease; if the request is asynchronous request, we will generate and insert a ResourceRelease instance after the corresponding Step object.

**Algorithm 3-8 ResourceAcquire and ResourceRelease Instances Generation Algorithm**

**Input:** *csm* with hostDemand for each step (output from Algorithm 3-6), and a list of CoupledMessage instances, *coupledMessageList* (output from Algorithm 3-2).

**Output**: *csm* with ResourceAcquire and ResourceRelease instances

1. for each CoupledMessage *coupledMessage* in the message list of each ETComponent object et*Component*
   1.1. if *coupledMessage* is a request message for et*Component* (sync or async request)
      1.1.1. find *coupledMessage*'s corresponding Step instance *step* in *csm;* find *step*'s predecessor and successor PathConnection instance *pre* and *post*
      1.1.2. create an object *sequence* of Sequence
      1.1.3. Create an object *resoucreAcquire* of ResoucreAcquire, and set its attribute "acquired component" to be et*Component*'s corresponding CSM component object (implement associations between ResoucreAcquire and Component).

1.1.4. set *pre*'s target to be *resoucreAcquire*; set *resoucreAcquire*'s predecessor to be *pre* (implement associations between ResoucreAcquire and PathConnection)

1.1.5. set *resoucreAcquire*'s successor to be *sequence*; set *sequence*'s source to be *resoucreAcquire* (implement associations between ResoucreAcquire and PathConnection)

1.1.6. set *sequence*'s target to be *step;* set *step*'s predecessor to be *sequence pre* (implement associations between ResoucreAcquire and PathConnection)

1.2. if *coupledMessage* is a *async* request

1.2.1. create an object *sequence* of Sequence

1.2.2. Create an object *resourceRelease* of ResourceRelease, and set its attribute "released component" to be et*Component*'s corresponding CSM component object (implement associations between ResoucreAcquire and Component).

1.2.3. set *sequence*'s target to be *resourceRelease*; set *resourceRelease*'s predecessor to be *sequence* (implement associations between ResoucreAcquire and PathConnection)

1.2.4. set *step*'s successor to be *sequence*; set *sequence*'s source to be *step* (implement associations between ResoucreAcquire and PathConnection)

1.2.5. set *post*'s source to be *resourceRelease;* set *resourceRelease*'s successor to be *post* (implement associations between ResoucreAcquire and PathConnection)

1.3. if *coupledMessage* is a *sync* request

1.3.1. find *coupledMessage*'s reply message and its corresponding Step instance *reply*

1.3.2. Change *reply* to type ResourceRelease, and set its attribute "released component" to be et*Component*'s corresponding CSM component object (implement associations between ResoucreAcquire and Component).

## 3.10 Summary

This Chapter has described an algorithm which translates execution traces to a CSM model. The algorithm has six consecutive phases. First, it preprocesses the trace data stored in a file, and couples two messages into an instance of CoupledMessage, from which the algorithm extracts instances of Component and Step in the second phase. In the third phase, instances of PathConnection are generated, which connects instances of Steps into a Scenario object. The algorithm detects synchronous communications in the fourth phase, thus instances of Messages can be created. The fifth phase is used to calculate hostDemand for each Step object, and then instances of ResourceAcquire and ResourceRelease are inserted into the CSM model during the last phase.

# Chapter 4: Validation

This chapter validates the algorithms described in chapter 3. There are eight test cases to examine whether a CSM model is constructed.

1. Various communication patterns: synchronous communication, asynchronous communication, and forwarding communication.
2. Various control flow patterns: sequence, fork and join.
3. Nested communication patterns: nested synchronous communication pattern, nested asynchronous communication pattern, and nested forwarding communication pattern.
4. Combinations of communication patterns and control flows: Synchronous communication pattern with fork, nested synchronous communications pattern with fork and join, and so on.
5. Service demand: verify if service demands are correctly calculated for each step in every test case.

Each test case will include:

1. A description which describes the purpose of this test case.
2. Execution traces used as input to the transformation.
3. A sequence diagram which shows messages sending between components including message names, timestamps, and service demand.
4. The output of the transformation in XML format and a CSM diagram.

## *4.1 Synchronous Communication Pattern*

### 4.1.1 Description

Execution traces are generated when client sends a request *browse* to server, and then server sends a request *display* to Inventory, and when they receive replies. From these traces, corresponding CSM model is created. This test case is to verify if we can automatically generate CSM model from execution trace of synchronous communications, and to verify if the CSM elements and their parameters are correct.

### 4.1.2 Execution traces

```
10001031 send Client browse_STARTC
10001050 receive Server browse_STARTC
10001057 send Server display_START
10001068 receive Inventory display_START
10001121 send Inventory display_END
10001131 receive Server display_END
10001149 send Server browse_ENDC
10001157 receive Client browse_ENDC
```

### 4.1.3 Sequence Diagram

In the following diagram, the number at the left or right of a message (arrow) is the timestamp that this message is sending or receiving. The number enclosed behind a message name is the host demand that corresponding step needs.

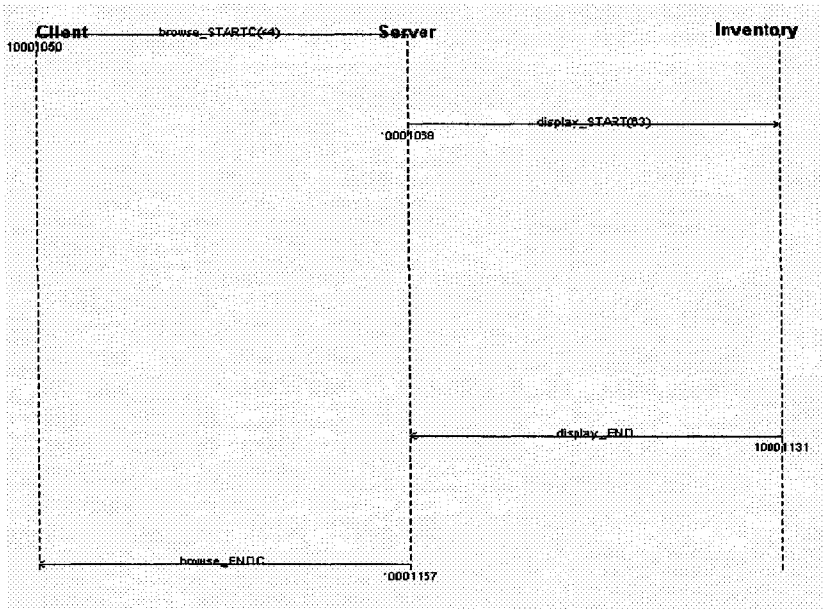



**Figure 4-1 Sequence Diagram of Synchronous Communication**

### 4.1.4 CSM Diagram

Figure 4.2 shows that the scenario starts when object Client gets a resource request and it in turn requests object Server. After the necessary resources are obtained, an event, i.e., Step browse_STARTC is executed. Without being released, object Sever sends a synchronous request to object Inventory, and all other steps follow the same procedure. Go to "Appendix B1Synchronous Communication Pattern" for corresponding XML file.

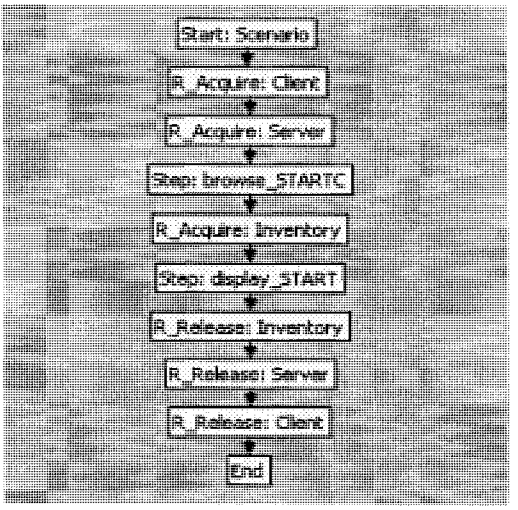



**Figure 4-2 CSM Diagram of Synchronous Communication**

### 4.1.5 LQN Model

See "Appendix C1 Synchronous Communication Pattern" for corresponding LQN file.

## *4.2 Asynchronous Communication Pattern*

### 4.2.1 Description

Execution traces are generated when client sends a request *browse* to server, and then server sends a request *display* to Inventory. After sending a request, corresponding components are released. From these traces, corresponding CSM model is created. This test case is to verify if we can automatically generate CSM model from execution trace of asynchronous communications, and to verify if the CSM elements and their parameters are correct.

### 4.2.2 Execution traces

| |
|---|
| 10001031 send Client browse_STARTC |
| 10001050 receive Server browse_STARTC |
| 10001057 send Server display_START |
| 10001068 receive Inventory display_START |

### 4.2.3 Sequence Diagram

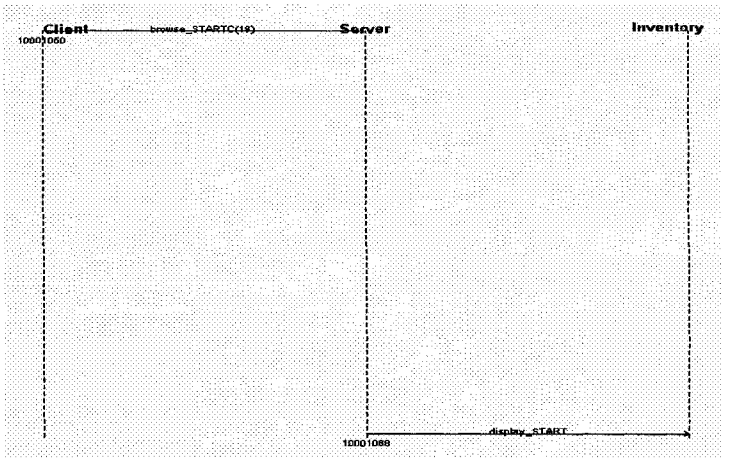



**Figure 4-3 Sequence Diagram of Asynchronous Communication**

### 4.2.4 CSM Diagram

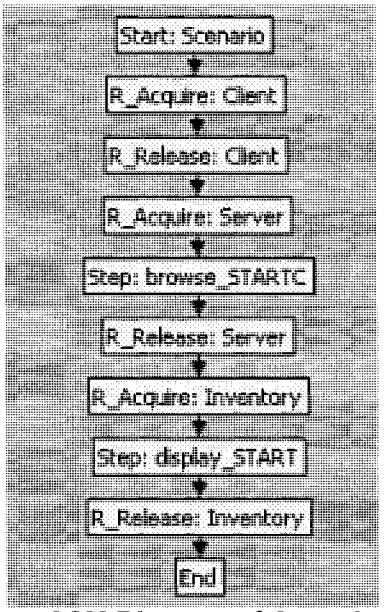



**Figure 4-4 CSM Diagram of Asynchronous Communication**

Go to "Appendix B2 Asynchronous Communication Pattern" for corresponding CSM file.

### 4.2.5 LQN Model

See "Appendix C2 Asynchronous Communication Pattern" for corresponding LQN file.

## *4.3 Forwarding Communication Pattern*

### 4.3.1 Description

Execution traces are generated when client sends a request *browse* to server, which is forward to Inventory, from which Client receive reply. From these traces, corresponding CSM model is created. This test case is to verify if we can automatically generate CSM model from execution trace of forwarding communications, and to verify if the CSM elements and their parameters are correct.

### 4.3.2 Execution traces

| |
|---|
| 10001031 send Client browse_STARTC |
| 10001050 receive Server browse_STARTC |

| |
|---|
| 10001057 send Server display_START |
| 10001068 receive Inventory display_START |
| 10001076 send Inventory getName_START |
| 10001087 receive Client getName_START |

### 4.3.3 Sequence Diagram

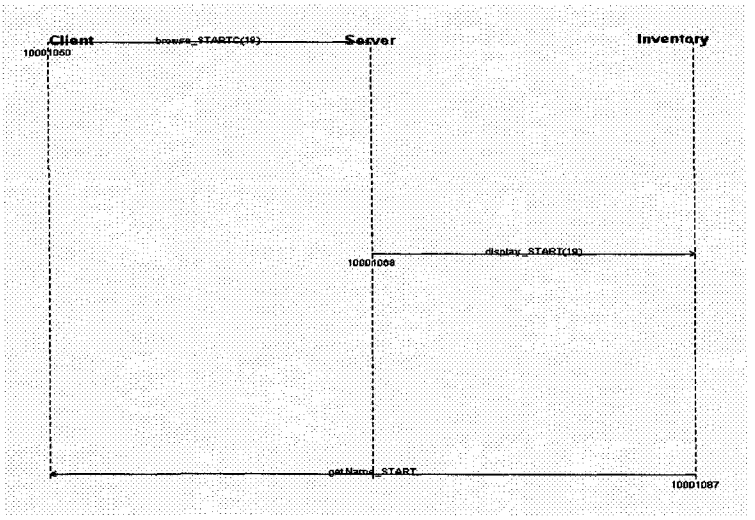



**Figure 4-5 Sequence Diagram of Forwarding Communication**

### 4.3.4 CSM Diagram

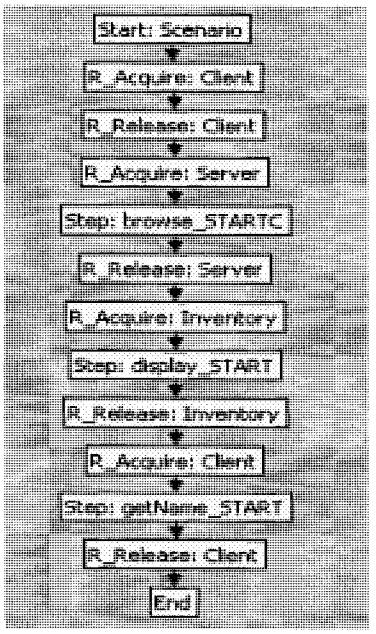



**Figure 4-6 CSM Diagram of Forwarding Communication**

Go to "B3 Forwarding Communication Pattern" for corresponding CSM file.

### 4.3.5 LQN Model

See "Appendix C3 Forwarding Communication Pattern" for corresponding LQN file.

## *4.4 Synchronous pattern with nested interaction*

### 4.4.1 Description

Execution traces are generated when client sends a synchronous request *browse* to server, which sends a synchronous request display to Inventory, which in turn sends a synchronous request getName to Book1. From these traces, corresponding CSM model is created. This test case is to verify if we can automatically generate CSM model from execution trace of nested synchronous communications, and to verify if the CSM elements and their parameters are correct.

### 4.4.2 Execution traces

```
10001031 send Client browse_STARTC
10001050 receive Server browse_STARTC
10001057 send Server display_START
10001068 receive Inventory display_START
10001076 send Inventory getName_START
10001087 receive Book1 getName_START
10001098 send Book1 getName_END
10001102 receive Inventory getName_END
10001121 send Inventory display_END
10001131 receive Server display_END
10001149 send Server browse_ENDC
10001157 receive Client browse_ENDC
```

## 4.4.3 Sequence Diagram

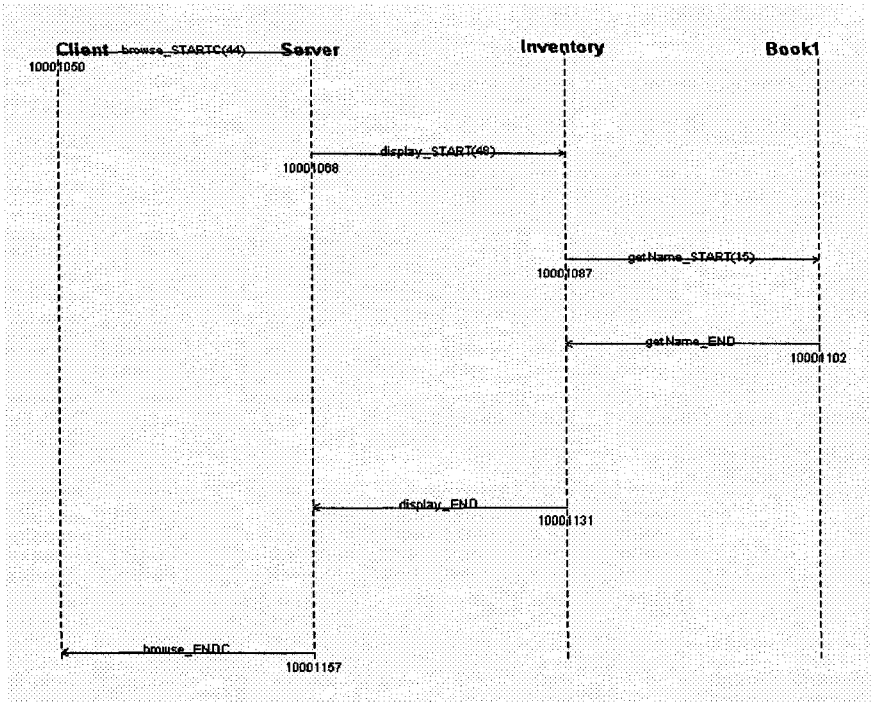



**Figure 4-7 Sequence Diagram of Synchronous Pattern with Nested Interaction**

## 4.4.4 CSM Diagram

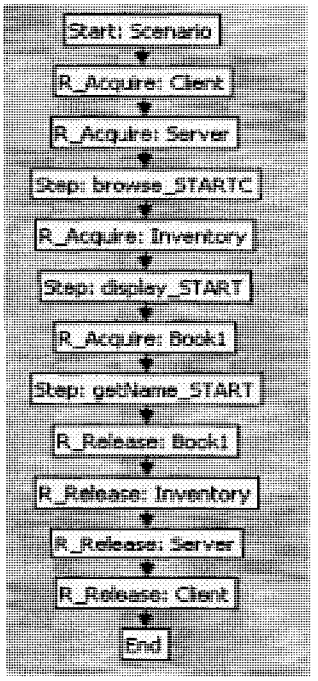



**Figure 4-8 CSM Diagram of Synchronous Pattern with Nested Interaction**

Go to "Appendix B4 Synchronous pattern with nested interaction" for corresponding CSM file.

### 4.4.5 LQN Model

See "Appendix C4 Synchronous pattern with nested interaction" for corresponding LQN file.

## *4.5 Synchronous Communication Pattern with Fork*

### 4.5.1 Description

Execution traces are generated when client sends a synchronous request *browse* to server, which sends an asynchronous request display to Inventory and a synchronous request displayBook to Book respectively. Therefore, the request to Server is forked to two components. From these traces, corresponding CSM model is created. This test case is to verify if we can automatically generate CSM model from execution trace of synchronous communication pattern with fork, and to verify if the CSM elements and their parameters are correct.

### 4.5.2 Execution traces

```
10001031 send Client browse_STARTC
10001050 receive Server browse_STARTC
10001057 send Server display_START
10001068 receive Inventory display_START
10001069 send Server displayBook_STARTC
10001073 receive Book displayBook_STARTC
10001135 send Book displayBook_END
10001140 receive Server displayBook_END
10001149 send Server browse_ENDC
10001157 receive Client browse_ENDC
```

## 4.5.3 Sequence Diagram

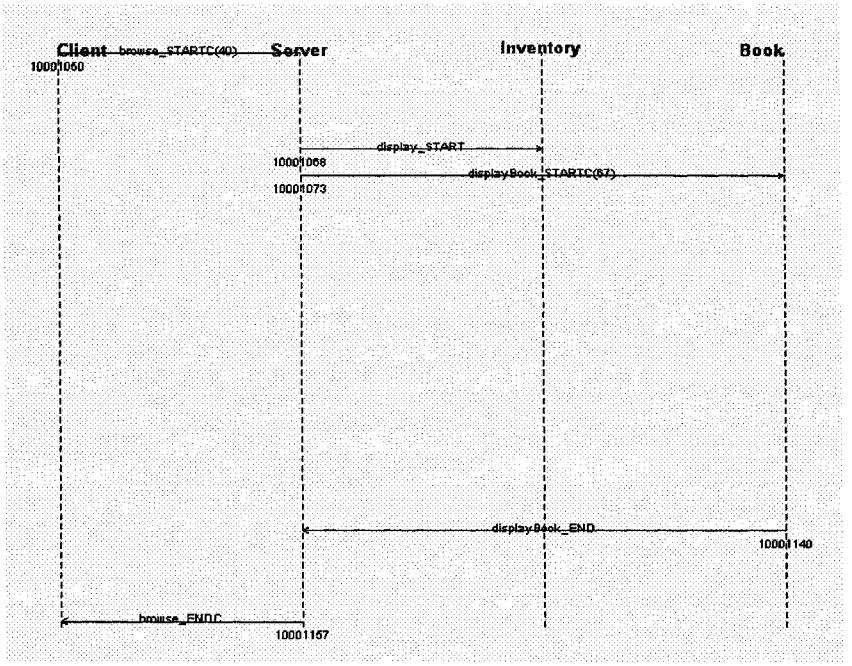



**Figure 4-9 Sequence Diagram of Synchronous Pattern with Fork**

## 4.5.4 CSM Diagram

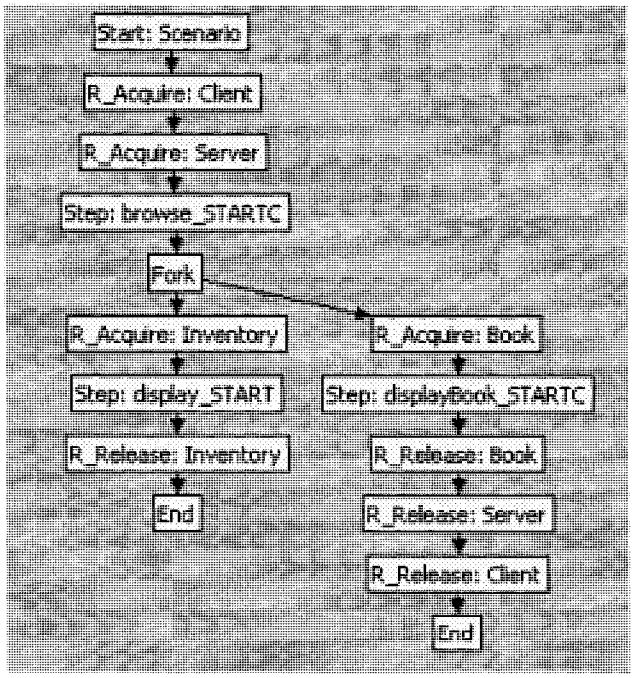



**Figure 4-10 CSM Diagram of Synchronous Pattern with Fork**

Go to "Appendix B5 Synchronous Communication Pattern with Fork" for corresponding CSM file.

### 4.5.5 LQN Model

See "Appendix C5 Synchronous Communication Pattern with Fork" for corresponding LQN file.

## *4.6 Nested Synchronous Communication Pattern with Fork and Join*

### 4.6.1 Description

Execution traces are generated when client sends a synchronous request *browse* to server, which sends two synchronous requests display and displayBook to Inventory and Book respectively. Therefore, the request to Server is forked to two components, and later replies from these two components are joined. From these traces, corresponding CSM model is created. This test case is to verify if we can automatically generate CSM model from execution trace of nested synchronous communication pattern with fork and join, and to verify if the CSM elements and their parameters are correct.

### 4.6.2 Execution traces

```
10001031 send Client browse_STARTC
10001050 receive Server browse_STARTC
10001057 send Server display_START
10001068 receive Inventory display_START
10001069 send Server displayBook_STARTC
10001073 receive Book displayBook_STARTC
10001110 send Inventory display_END
10001111 receive Server display_END
10001135 send Book displayBook_END
10001140 receive Server displayBook_END
10001149 send Server browse_ENDC
10001157 receive Client browse_ENDC
```

## 4.6.3 Sequence Diagram

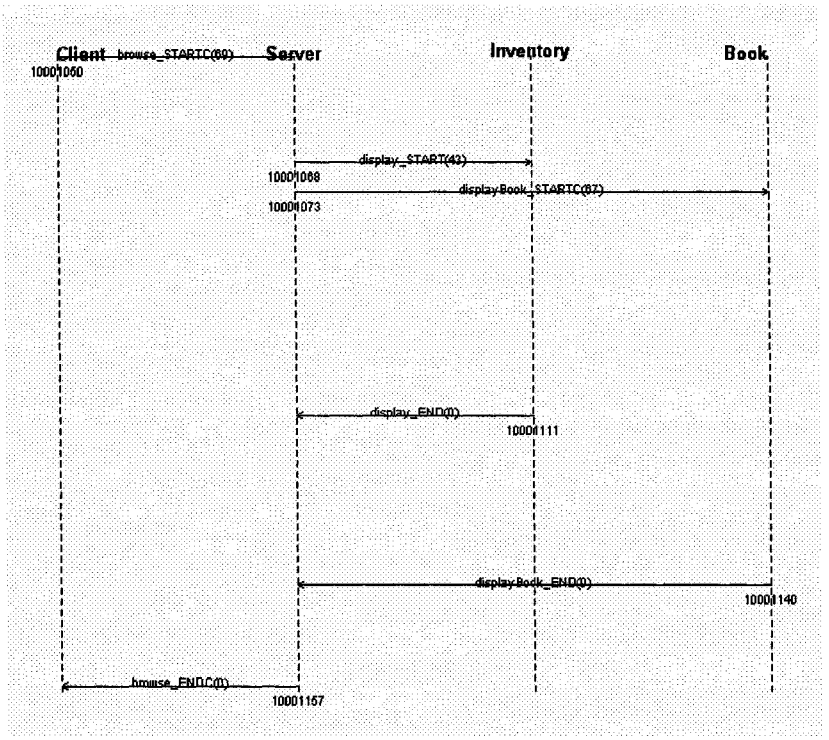



**Figure 4-11 Sequence Diagram of Nested Synchronous Pattern with Fork and Join**

## 4.6.4 CSM Diagram

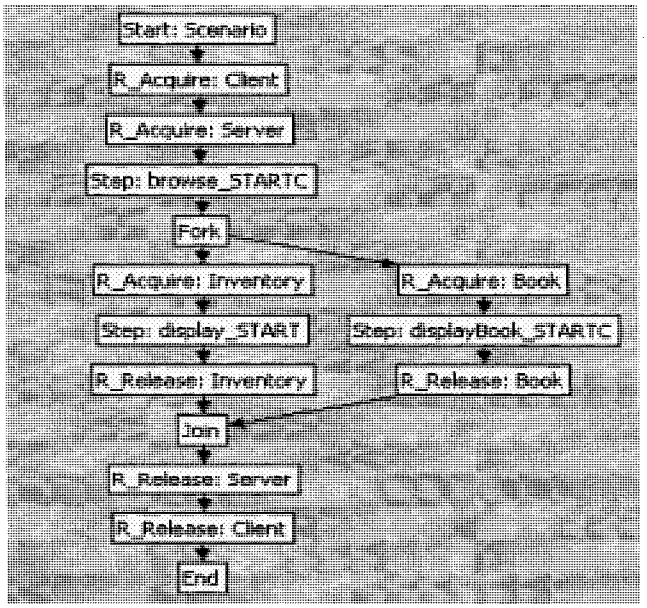



**Figure 4-12 CSM Diagram of Nested Synchronous Pattern with Fork and Join**

Go to "Appendix B6 Nested Synchronous Communication Pattern with Fork and Join" for corresponding CSM file.

### 4.6.5 LQN Model

See "Appendix C6 Nested Synchronous Communication Pattern with Fork and Join" for corresponding LQN file.

## *4.7 Asynchronous Communication Pattern with Fork*

### 4.7.1 Description

Execution traces are generated when client sends a synchronous request *browse* to server, which sends two asynchronous request display and displayBook to Inventory and Book respectively. Therefore, the request to Server is forked to two components. From these traces, corresponding CSM model is created. This test case is to verify if we can automatically generate CSM model from execution trace of asynchronous communication pattern with fork, and to verify if the CSM elements and their parameters are correct.

### 4.7.2 Execution traces

```
10001031 send Client browse_STARTC
10001050 receive Server browse_STARTC
10001057 send Server display_START
10001068 receive Inventory display_START
10001070 send Server displayBook_START
10001075 receive Book displayBook_START
```

### 4.7.3 Sequence Diagram

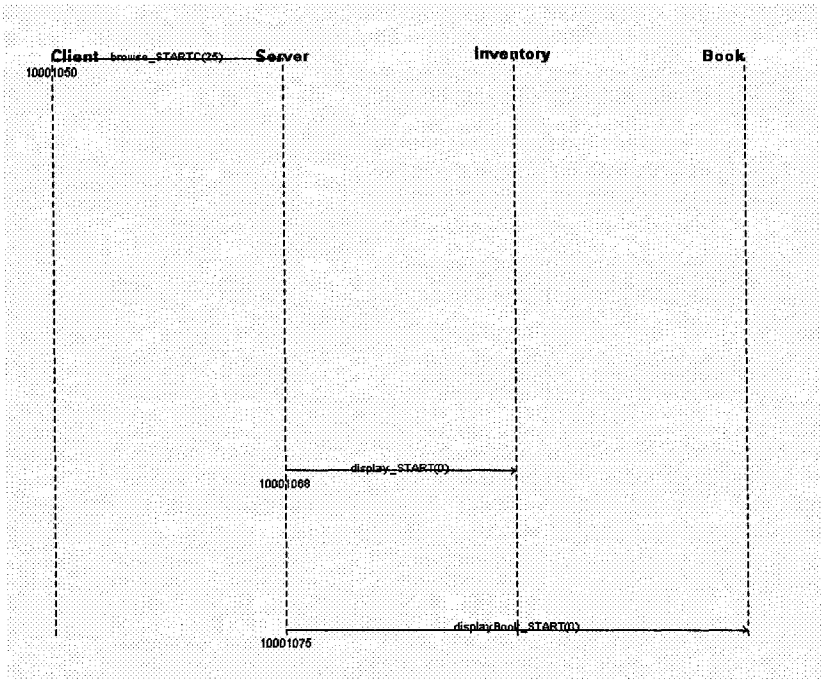



**Figure 4-13 Sequence Diagram of an Asynchronous Pattern with Fork**

## 4.7.4 CSM Diagram

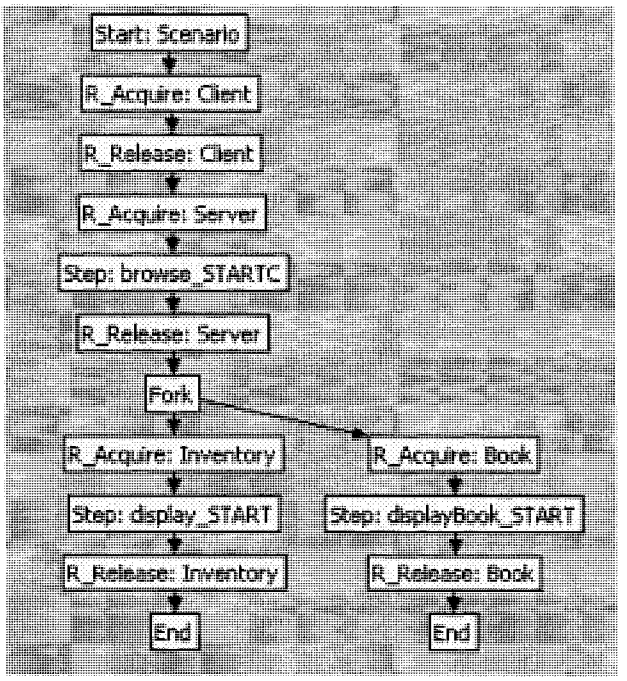



**Figure 4-14 CSM Diagram of an Asynchronous Pattern with Fork**

Go to "See Appendix B7 Asynchronous Communication Pattern with Fork" for corresponding CSM file.

### 4.7.5 LQN Model

See "Appendix C7 Asynchronous Communication Pattern with Fork" for corresponding LQN file.

## *4.8 A Complicated Example*

### 4.8.1 Description

The scenario begins with a nested synchronous communication that involves component client, server, and inventory. After server sends a reply to client, it sends an asynchronous request, createAccount, to CustomerAccount, therefore, a fork is generated. After receiving the request, CustomerAccount sends a synchronous request to CAccount1, and then a serial synchronous communications are generated. These communications need a Fork and Join.

This test case is to verify if we can automatically generate CSM model from execution trace recorded during this complicated scenario, and to verify if the CSM elements and their parameters are correct.

### 4.8.2 Execution traces

```
10001031 send Client browse_STARTC
10001050 receive Server browse_STARTC
10001057 send Server display_START
10001068 receive Inventory display_START
10001121 send Inventory display_END
10001131 receive Server display_END
10001149 send Server browse_ENDC
```

10001157 receive Client browse_ENDC

10001195 send Server createAccount

10001203 receive CustomerAccount createAccount

10001216 send CustomerAccount getPassword_START

10001229 receive CAccount1 getPassword_START

10001236 send CAccount1 getPassword_END

10001250 receive CustomerAccount getPassword_END

10001257 send CustomerAccount getLogin_START

10001260 receive Inventory getLogin_START

10001277 send Inventory getLogin_END

10001280 receive CustomerAccount getLogin_END

10001267 send CustomerAccount getBalance_START

10001269 receive CAccount1 getBalance_START

10001287 send CAccount1 getBalance_END

10001289 receive CustomerAccount getBalance_END

10001310 send CustomerAccount getList_START

10001312 receive Inventory getList_START

10001330 send Inventory getList_END

10001332 receive CustomerAccount getList_END

### 4.8.3 Sequence Diagram

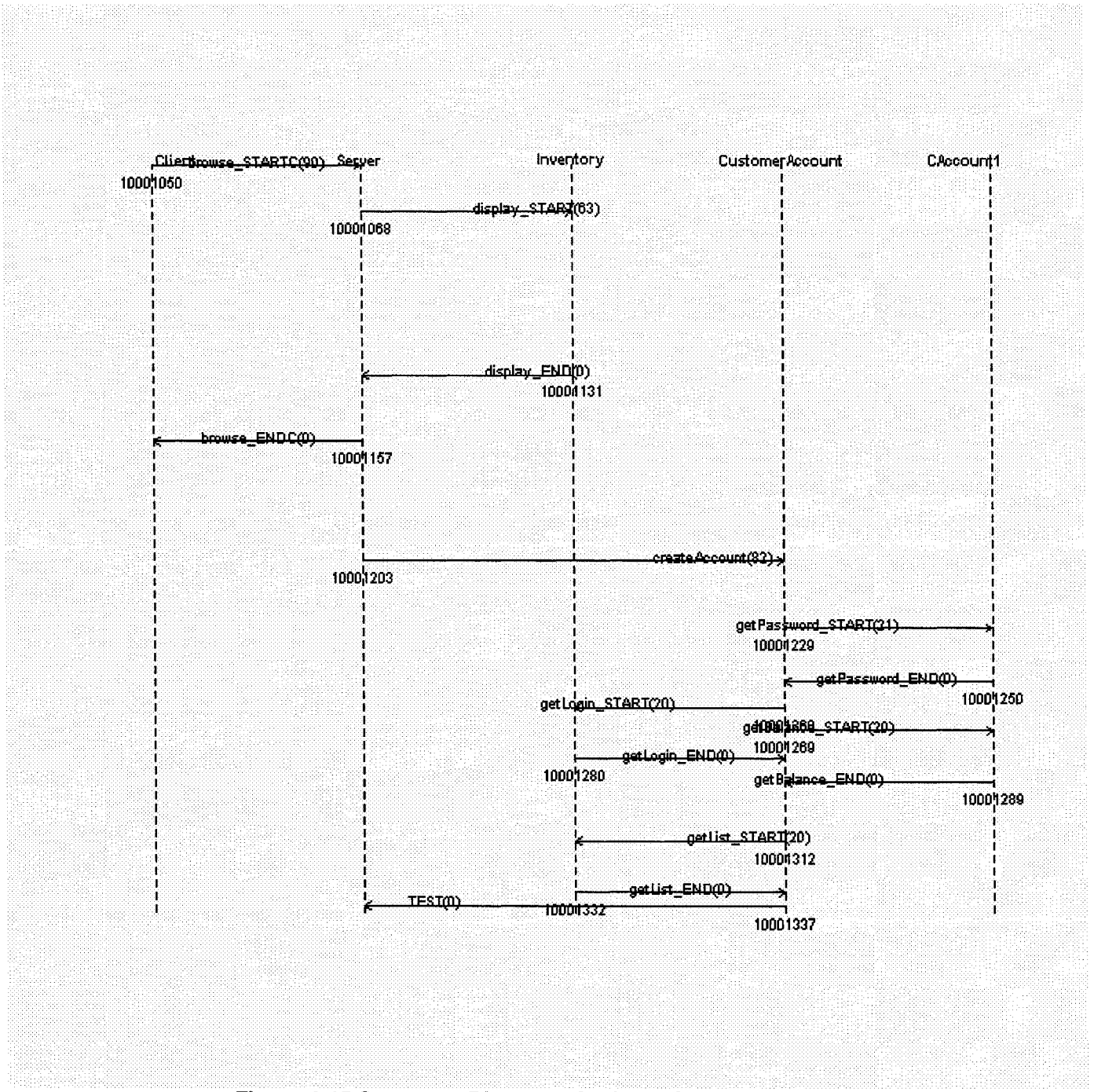



**Figure 4-15 Sequence Diagram of a Complicated Example**

### 4.8.4 CSM Diagram

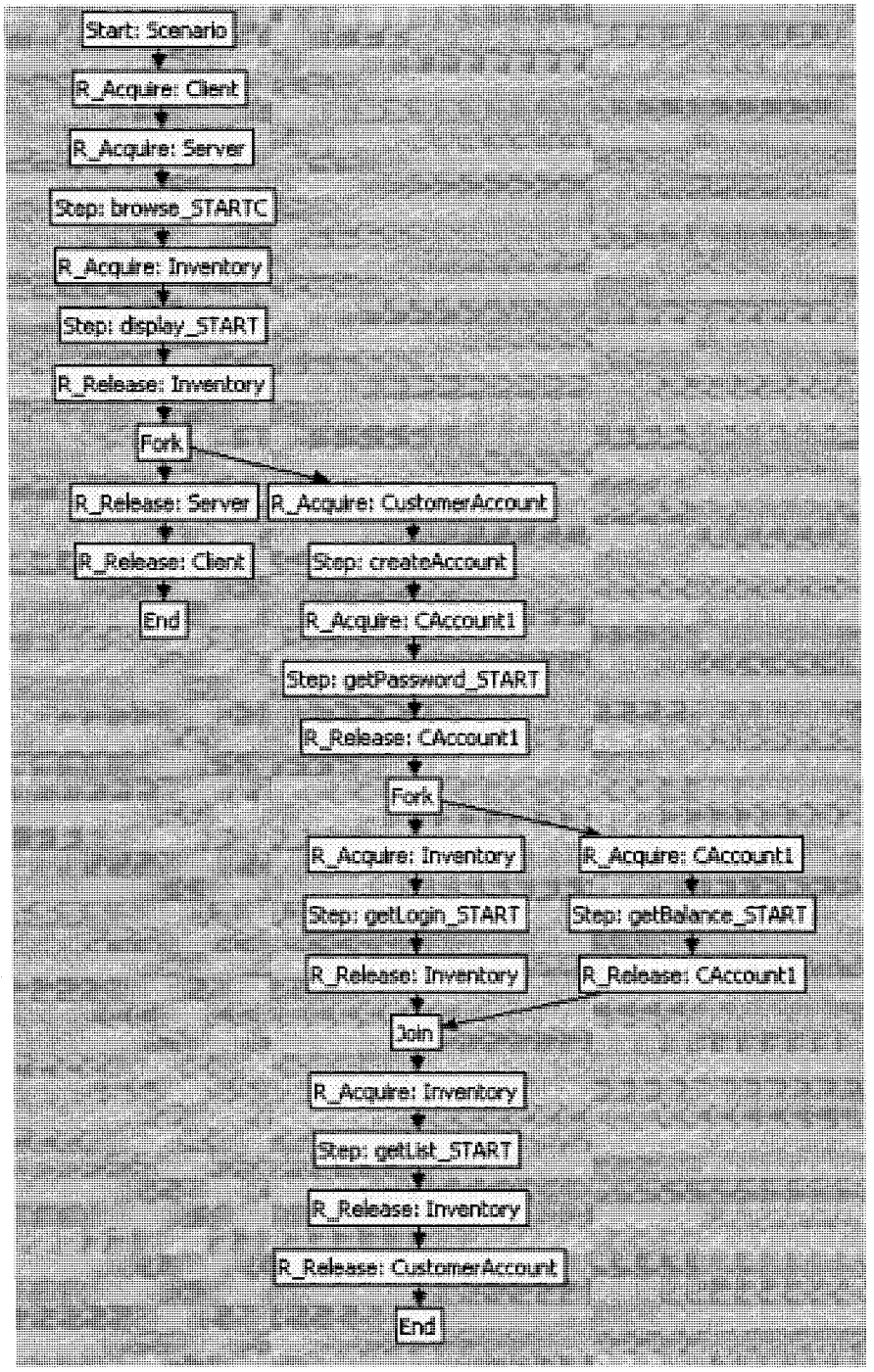



**Figure 4-16 CSM Diagram of a Complicated Example**

Go to "Appendix B8 A Complicated Example" for corresponding CSM file.

# Chapter 5: Case Study - RADS Bookstore

To test the algorithm on traces created from a running prototype system, an application called "RADSBookstore" was prototyped and run in a laboratory setting [Tauseef 07]. It consisted of multiple processes running on a single CPU, with a separate event trace per process.

RADSBookstore has a three-tier architecture

- the user interface, implemented as a test driver which presents inputs to the bookstore,
- a business application layer with the logic of the bookstore,
- A third tier that stores the data on inventory and orders.

Objects play the role of components; some objects are called remotely via Java RMI.

Before tracing, we know the system structure as described above, and use cases, i.e., we know the functionalities of RADSBookstore, but we do not have to know what components are, how they interact, and other details. All these will be abstracted or deduced during the construction of CSM models. The implementation was instrumented to capture a message trace between components, when executing one use case, traces are recorded, from which we will generate:

- A sequence diagram
- A CSM diagram
- A CSM XML file

At the time of this thesis being written, the tool that transforms CSM models to LQN models has not been fully implemented; otherwise we can generate LQN models from CSM XML files.

## *5.1 Use Cases*

Fig. 5.1.1 shows the fourteen use cases that can be executed by a client, and by a manager charged with replenishing the inventory. Some use cases may include others. For example, the use case "Check out" may need Login, because in order to check out a client must first login the system; also it may need "Browse products", "View Product Info", "Changes Settings", 'View Shopping Cart", 'Add to cart" and "Remove From Cart". Therefore, in our case study, we will exam all the fourteen use cases in fewer scenarios. The procedure for each scenario is similar, that is:

- Run the program to obtain execution traces
- Generate corresponding sequence diagram and CSM XML file
- Generate CSM diagram using CSM viewer.

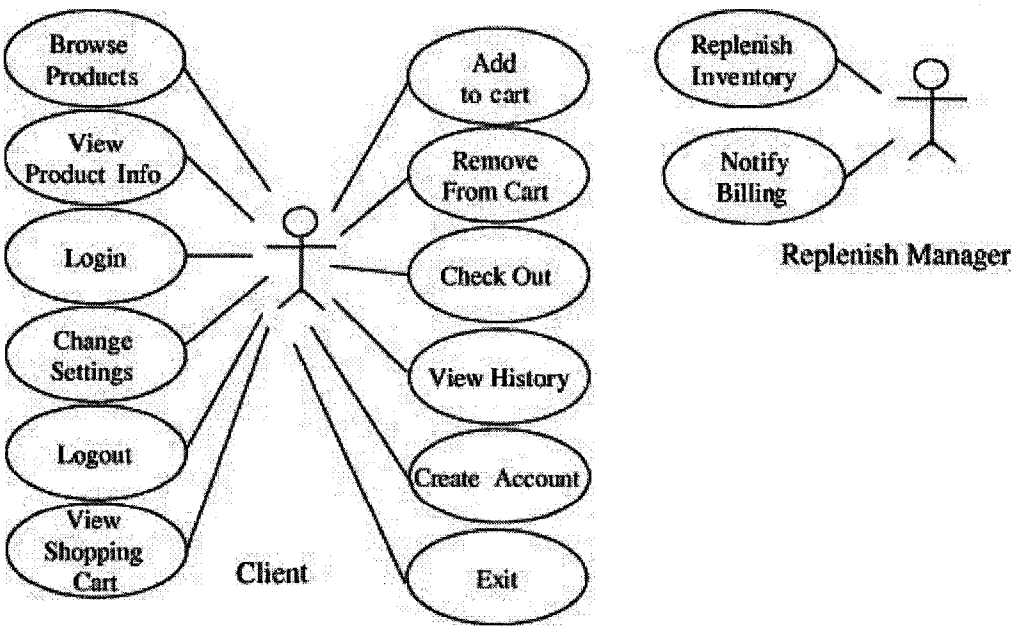



**Figure 5-1 Use Case Diagram of RADSBookstore**

## 5.2 Use Case: Replenish Inventory and Notify Billing

When replenish managers find that books in inventory need to be replenished, they will ask the program to order books, add them to inventory, and pay the bill. This is what the use case "Replenish Inventory" and "Notify Billing" describe. For execution traces generated from this scenario see Appendix A1; for CSM XML file transformed from traces, see Appendix B9. Following are corresponding sequence and CSM diagram.

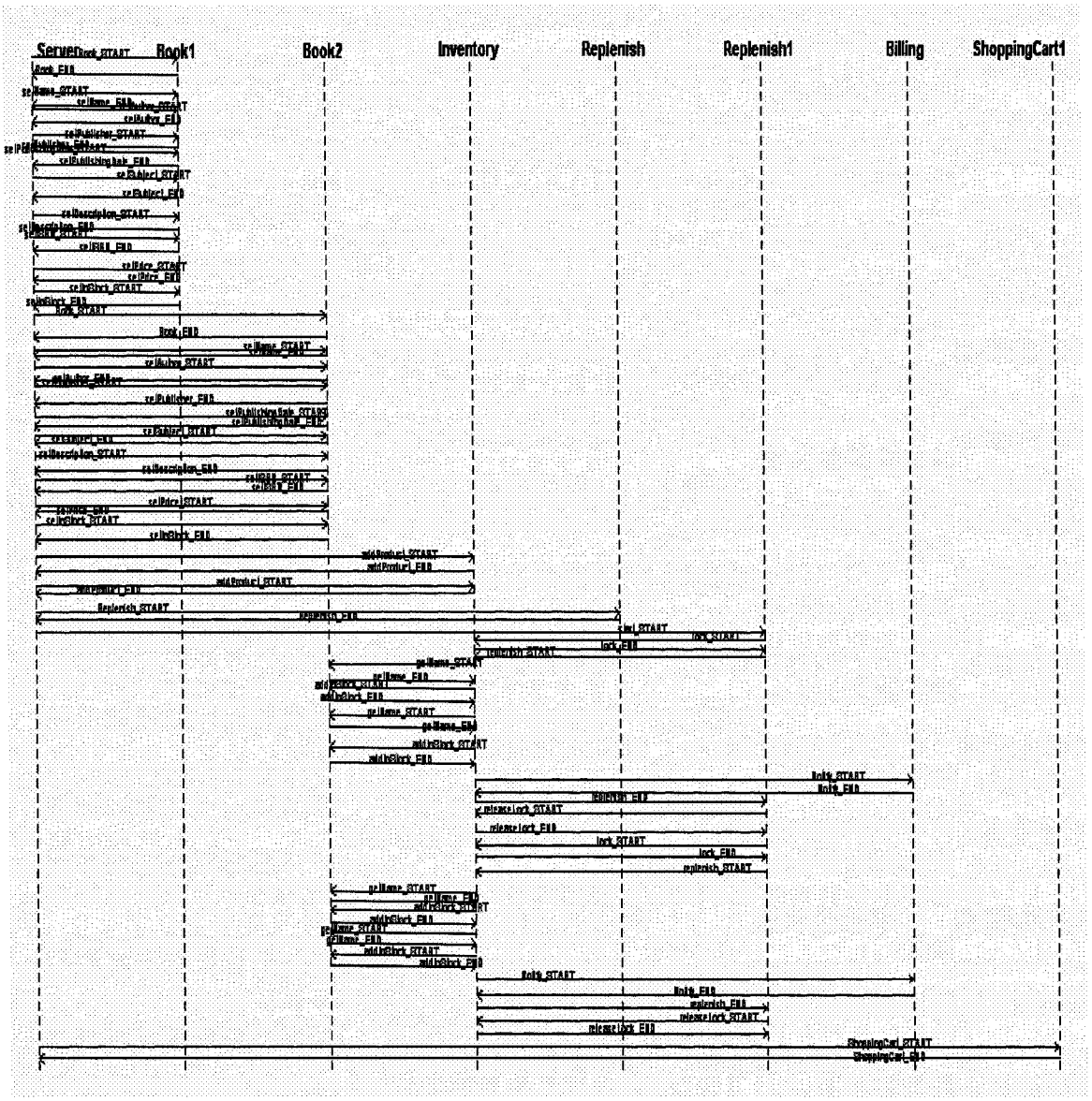



**Figure 5-2 Sequence Diagram of Use Case: Replenish Inventory and Notify Billing**

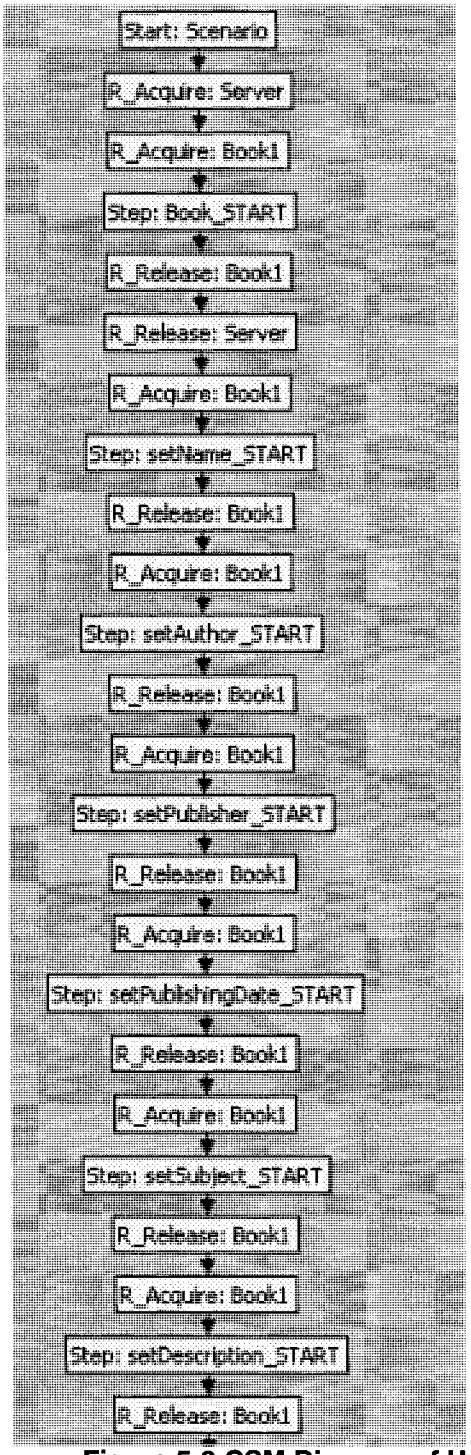



**Figure 5-3 CSM Diagram of Use Case: Replenish Inventory and Notify Billing**

## 5.3 Use Case: Create Account, Login and Add to Cart

For new clients, they need to first create an account, and then login system using this account, thus they can put some books in their shopping carts. For execution traces generated from this scenario see Appendix A2; for CSM XML file transformed from traces, see Appendix B10. Following are corresponding sequence and CSM diagram.

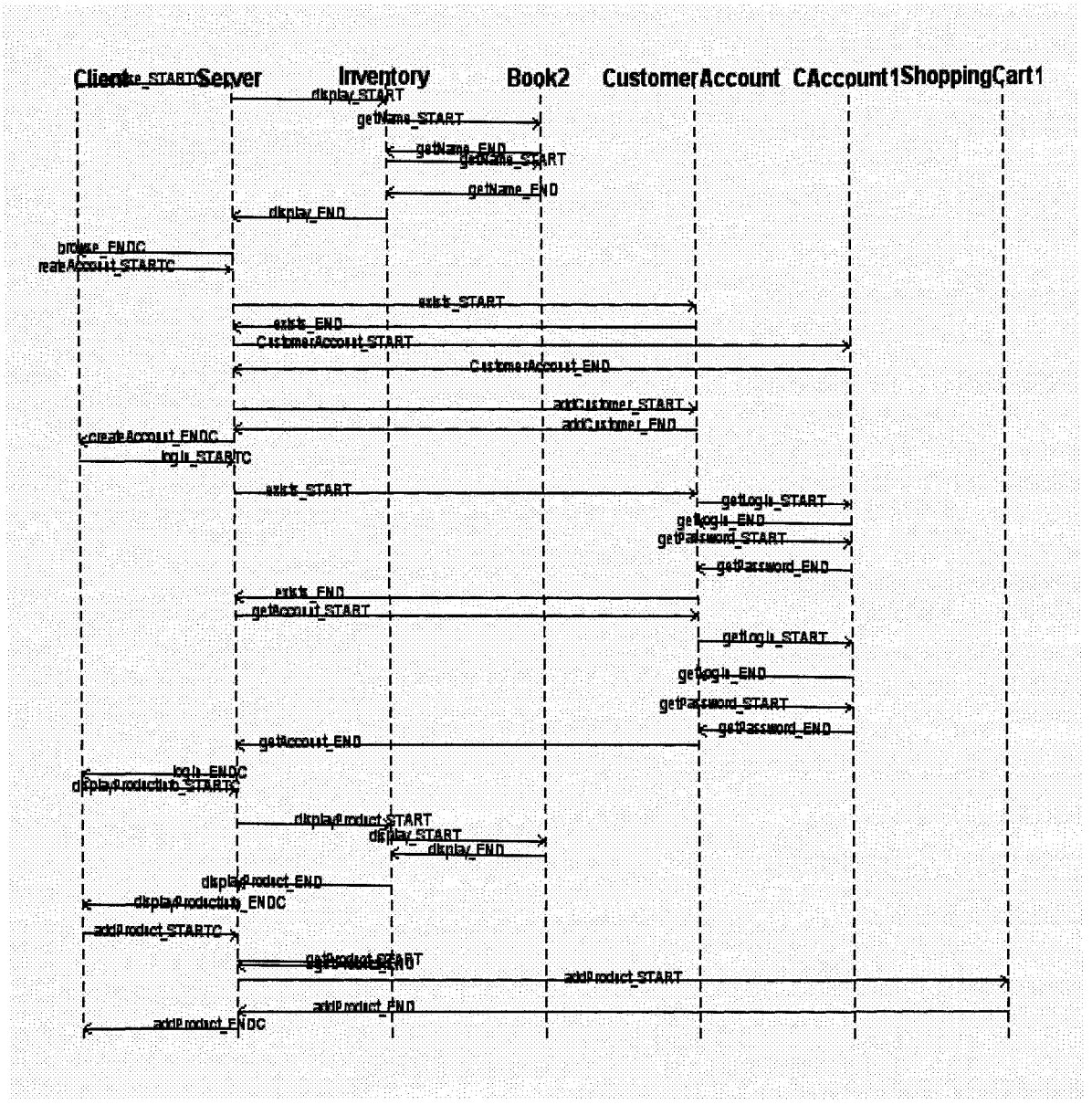



**Figure 5-4 Sequence Diagram of Use Case: Create Account, Login and Add to Cart**

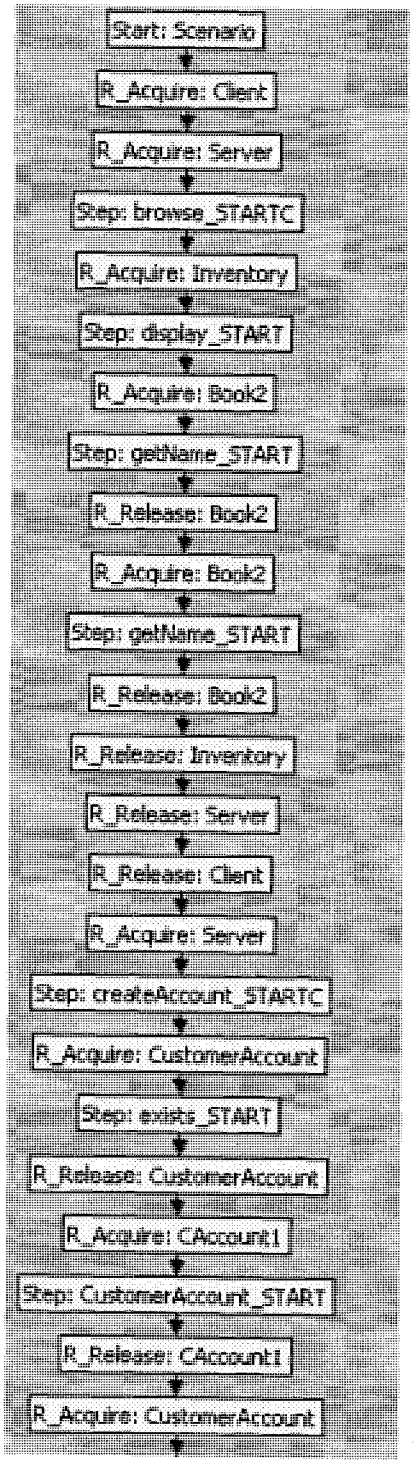



**Figure 5-5 CSM Diagram of Create Account, Login and Add to Cart**

## 5.4 Checkout scenario

When clients have put some books into their shopping cart, and decide to purchase these books, they need to check out. This scenario may involve multiple use cases, such as view Product Info, Remove From Cart, and Check Out. For execution traces generated from this scenario see Appendix A3; for CSM XML file transformed from traces, see Appendix B11. Following are corresponding sequence and CSM diagram.

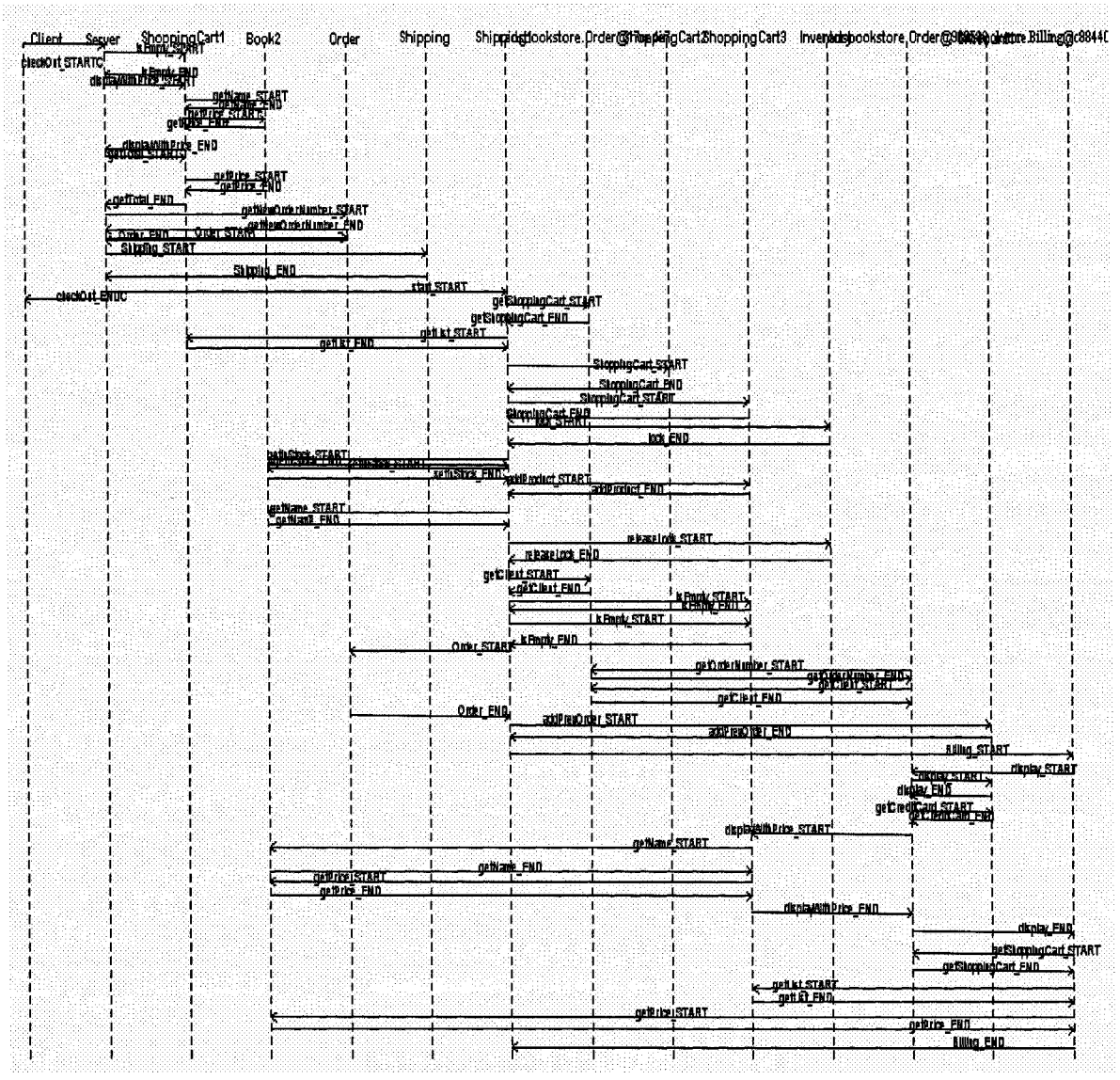



**Figure 5-6 Sequence Diagram of Checkout Scenario**

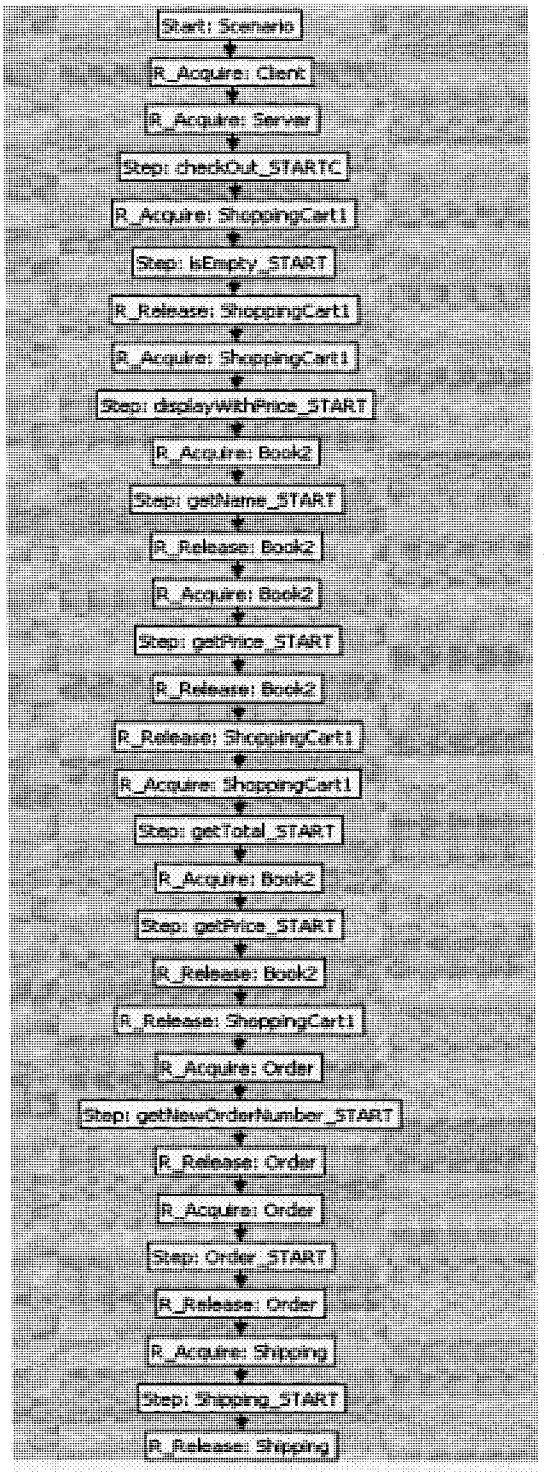



**Figure 5-7 CSM Diagram of Scenario Checkout**

# Chapter 6: Conclusion and Future Work

## *6.1 Conclusion*

This thesis has focused on the transforming of execution traces to CSM models, which can be converted to LQN using existing technique. The transformation algorithm is basically to extract instances of certain CSM elements first (such as Component and Step instances) from execution traces, and then deduce instances of other CSM elements (such as ResourceAcquire, ResourceRelease, PathConnection, and Classifier instances) from execution traces or extracted instances of other CSM elements. The generated CSM models only cover a subset of CSM elements defined in CSM meta-model, because the information in the execution traces is limited.

The thesis also presents how to detect synchronous communication pattern and calculate service time. The transformation algorithms defined in chapter 3 are validated in chapter 4 by various communications patterns. The transformation algorithm is implemented in Java programming language, and applied to RADSBookstore, a distributed application. Execution traces generated from all the use cases are successfully transformed to CSM models.

## *6.2 Future Work*

This technique has following limitations, which needs to be solved in the future.

### 6.2.1 Algorithm Limitations

**repCount of Step cannot be calculated**

The attribute repCount of CSM class Step is used to indicate the number of repetitions for the Step instance. In order to obtain repCount, repetition needs to be defined. That is to say,

a pattern or a rule should be defined to differentiate single and repetitive occurrence of a step or a series of steps. Since different rules may give different results, the repCount is ignored. Instead, all the steps have a repCount one and are shown in CSM model.

### 6.2.2 Limitations due to Insufficient Information

**Multiple Threads (Multiple Processors) are not Supported**

When two messages are coupled, the assumption, "A message sent first will be received first for all the components," is necessary. However, in an environment with multiple threads (since each thread works independently) it is very common that one message which was sent earlier can be received later. Therefore, this assumption is too strong. If thread ids are inserted into the execution traces, the algorithm can be modified and used for multiple threads. The same reasoning and solution apply to multiple processors.

**Content Switching is not Supported**

When content switching happens, no information is written into the execution traces. Therefore, when hostDemand is calculated extra time will be included in corresponding the Step instance. If information regarding the beginning and ending timestamp, and the related component and step name is written into execution traces, the algorithm can be modified in a way similar to deal with nested synchronous communication.

**Branch and Merge can not be Generated**

When Branch (or Fork) and Merge (or Join) appears in a single execution, the execution records generated are same as there is no Branch and Merge. One way to remove this limitation is to add workflow information into execution traces. Another way is to process multiple executions separately, then compare the traces and find the points at which executions records become different (Branch) and the same (Merge).

**Message Coupling may be Wrong**

As mentioned in section 3.1.4, because two execution records generated at both ends of a message invocation do not contain the name of the other component, we may not be able to couple the two records to a message correctly. Therefore, the CSM model generated may not be correct. The best way is solve this is to add both component names into one execution record.

### 6.2.3 External Limitations

The CSM models generated in section 4, Validation, and section 5, Case Study have not been converted to LQN models because the tool that transforms CSM to LQN has not been fully implemented.

## References


[Amer thesis] Amer, H., "*Automatic Transformation of UML Software Specifications into LQN Performance Models by using Graph Grammar Techniques*", Master Thesis, Carleton University, Ottawa, Canada, 2001

[Amer] Hoda Amer, Dorina C. Petriu, "*Software Performance Evaluation: Graph Grammar-based Transformation of UML Design Models into Performance Models*", submitted for publication, 2002

[Arief 00] Arief, L. B. and N. A. Speirs, "*A UML tool for an automatic generation of simulation programs*," Proceedings of WOSP 2000, pp. 71–76.

[Arief 99 A] Arief, L. B. and N. A. Speirs, "*Automatic generation of distributed system simulations from UML*," Proceedings of ESM '99, 13th European Simulation Multiconference, Warsaw, Poland, pp. 85–91.

[Arief 99 B] Arief, L. B. and N. A. Speirs, *"Using SimML to bridge the transformation from UML to simulation*," Proc. of One DayWorkshop on Software Performance and Prediction extracted from Design, Heriot-Watt University, Edinburgh, Scotland.

[Balsamo 01 A] Simonetta Balsamo, Marta Simeoni, "*On transforming UML models into performance models*," Univ.d. Venezia, WTUML, ETAPS 2001,

[Balsamo 01 B] S. Balsamo and M. Simeoni. "Deriving performance models from software architecture specifications," Proceedings of ESM'01 [82].

[Balsamo 03] S. and M. Marzolla, "*Simulation Modeling of UML Software Architectures*", Proc. ESM'03, Nottingham (UK), June 2003

[Balsamo 04] Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M., *"Model-based performance prediction in software development: a survey*" IEEE

Transactions on Software Engineering, Vol 30, N.5, pp.295-310, May 2004.

[Balsamo 05] Simonetta Balsamo, Moreno Marzolla, "*Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models*," WOSP'05 July 12–14, 2005, Palma de Mallorca, Spain

[Barber] Scott Barber, "*Creating Effective Load Models for Performance Testing with Incomplete Empirical Data*," in Proc. 6th IEEE Int. Workshop on Web Site Evolution, 2004, pp. 51-59.

[Bernardi] S. Bernardi, S. Donatelli, and J. Merseguer, *"From UML sequence diagrams and statecharts to analysable Petri net models*," in Proc. 3rd Int. Workshop on Software and Performance (WOSP02), Rome, July 2002, pp. 35-45.

[Buhr 96] R.J.A. Buhr and R.S. Casselman, "*Use Case Maps for Object-Oriented Systems*", Prentice Hall, 1996

[Buhr 98] R. J. A. Buhr, "*Use Case Maps as Architectural Entities for Complex Systems*," IEEE Transactions on Software Engineering, vol. 24, no. 12 pp. 1131 - 1155, 1998.

[Canevet 03] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens. *"Performance modelling with UML and stochastic process algebras*," IEE Proceedings: Computers and Digital Techniques, 150(2): 107-120, Mar. 2003.

[Cavenet 04] C. Cavenet, S. Gilmore, J. Hillston, L. Kloul, and P. Stevens, "*Analysing UML 2.0 activity diagrams in the software performance engineering process*," in Proc. 4th Int. Workshop on Software and Performance (WOSP 2004), Redwood City, CA, Jan 2004, pp. 7

[Cortellesa] V. Cortellesa, "*Deriving a Queueing Network Based Performance Model from UML Diagrams*," in Proc. Second Int. Workshop on Software and Performance (WOSP2000), Ottawa, Canada, 2000, pp. 58-70

[D.B.Petriu 02] Dorin Petriu, Murray Woodside, "*Software Performance Models from System Scenarios in Use Case Maps*", Proc. 12 Int. Conf. on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation (Performance TOOLS 2002), Lon-don,

[D.B.Petriu 04] Dorin B. Petriu, Murray Woodside, "*A Metamodel for Generating Performance Models from UML Designs*", Proc UML 2004, Lisbon, v. 3273 of Lecture Notes in Computer Science (LNCS 3273), pp 41-53, Oct 2004.

[D.C.Petriu 00] D.C. Petriu, X. Wang, "*From UML descriptions of high-level software architecture to LQN performance models*", in "Applications of Graph Transformations with Industrial Relevance AGTIVE'99" (eds. M.Nagl, A. Schuerr, M. Muench), Lecture Notes in Computer Science Vol. 1779, pp. 47-62, Springer, 2000.

[D.C.Petriu 02] D.C.Petriu, H.Shen, "*Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications*", in Computer Performance Evaluation - Modelling Techniques and Tools, (Tony Fields, Peter Harrison, Jeremy Bradley, Uli Harder, Eds.) Lecture Notes in Computer Science 2324, pp. 159-177, Springer Verlag, 2002.

[D'Ambrogio] Andrea D'Ambrogio, *"A Model Transformation Framework for the Automated Building of Performance Models from UML Models*," Proc. of WOSP'05 July 1214, 2005, Palma de Mallorca, Spain

[Dauphin] P. Dauphin, R. Hofmann, R. Klar, B. Mohr, A. Quick, M. Siegle, and F. Sotz, "ZM4/ Simple, "*A General Approach to Performance Measurement and Evaluation of Distributed Systems*," T. Casavant and M. Singhal, eds., Readings in Distributed Computing System.

[Geist] G.A. Geist et al., "*PICL: A Potable Instrumented Communication Library*", C Reference Manual, Tech. Report ORNL/TM-I1130,Oak Ridge Nat'l Lab., Oak Ridge,Tenn.1990.

[Grassi] Vincenzo Grassi, Raffaela Mirandola, Antoine Sabetta, "*From design to analysis models: a kernel language for performance and reliability analysis of component-based systems*", Proceedings of the 5th international workshop on Software and performance WOSP '05, pp. 25-36, July 2005

[Gu 02] Gordon Gu, D. C. Petriu, "*XSLT Transformation from UML Models to LQN Performance Models*", Proc. of 3rd Int. Workshop on Software and Performance WOSP'2002, pp.227- 234, Rome, Italy, 2002.

[Gu 05] Gordon P. Gu, D.C.Petriu, "*From UML to LQN by XML algebra-based model transformations*", Proceedings of the 5th international workshop on Software and performance WOSP '05, pp. 99-110, July 2005

[Heath 91] M. Heath and J. Etheridge, "*Visualizing the Performance of Parallel Programs*," IEEE Software, vol. 8, no. 5, pp. 29–39, Sept. 1991.

[Heath 95] Michael T. Heath , Allen D. Malony , Diane T. Rover, "*The Visual Display of Parallel Performance Data*", Computer, v.28 n.11, p.21-28, November 1995

[Herzog] U. Herzog, "*Formal description, time and performance analysis: A framework*," Technical Report 15/90, IMMD VII, Friedrich-Alexander-Universitat, Erlangen-Nurnberg, Germany, September 1990.

[Hillston 96] J. Hillston, "*A Compositional Approach to Performance Modelling*," Cambridge University Press, 1996.

[Hillston 98] Jane Hillston, Rob Pooley, "*Stochastic Process Algebras and their Application to Performance Modelling*," Proc. Tutorial, TOOLS'98, Palma de Mallorca, Spain, Sept. 1998.

[Hrischuk 95] Curtis E. Hrischuk, Jerome A. Rolia, C. Murray Woodside, "*Automatic Generation of a Software Performance Model Using an Object-Oriented Prototype*", Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 399-409, Durham, NC, January 1995.

[Hrischuk 99] Curtis E. Hrischuk, C. Murray Woodside, Jerome A. Rolia, Rod Iversen, "*Trace-Based Load Characterization for Generating Performance Software Models*," IEEE Transactions on Software Engineering, VOL. 25, NO. 1, January 1999.

[Hrischuk Exe] C. E. Hrischuk, "*Principles for the Automated Construction of Distributed Application Software Execution Models*," The Pennsylvania State University CiteSeer Archives, 2001.

[Jensen 92] K. Jensen, Coloured Petri Nets, Vol 1: *Basic Concepts, Springer-Verlag* 1992.

[King 99] King, P., & Pooley, R., "*Derivation of Petri Net Performance Models from UML Specifications of Communication Software*," Proc. of XV UK Performance Engineering Workshop, 1999.

[Lakos 94] C.A.Lakos, "*Object Petri nets – Definition and relationship to colored nets*," Tech Report TR 94- 3, Computer Science Dept, University of Tasmania.

[Lazowska] E.D. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, "*Quantitative System Performance*," Englewood Cliffs, N.J.: Prentice Hall,1984.

[Lo'pez-Grao] J. P. Lo'pez-Grao, J. Merseguer, and J. Campos, "*From UML Activity Diagrams To Sto-chastic Petri Nets: Application To Software Performance Engineering*," in Fourth Int. Workshop on Software and Performance (WOSP 2004), Redwood City, CA, Jan. 2004, pp.

[Malony] Allen D. Malony , B. Robert Helm, "*A theory and architecture for automating performance diagnosis*," Future Generation Computer Systems, v.18 n.1, p.189-200, September 2001

[Maqbool] Shahbaz Maqbool, "*Transformation of a Core Scenario Model and Activity Diagrams into Petri Nets*," Master Thesis, University of Ottawa, Ottawa, Canada, 2005

[Marsan] M. Ajmone Marsan, G. Balbo, and G. Conte, "*A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems*," ACM Trans. Computer Syst., vol. 2, no. 1, May 1984

[Marzolla] Moreno Marzolla, "*Simulation-Based Performance Modeling of UML Software Architectures*," PHD thesis, Universit`a Ca' Foscari di Venezia, Febuary 2004

[Mehra] P. Mehra, M. Gower, and M.A. Bass, *"Automated Modeling of Message-Passing Programs*," Proc. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems MASCOTS'94, pp. 187–192. IEEE CS Press, 1994.

[Murata 89] T. Murata, "*Petri Nets: Properties, Analysis and Applications*," Proceedings of the IEEE, Vol.77, No.4 pp.541-580, April 1989.

[OMG SPT] Object Management Group (2002a, March), "*UML profile for schedulability, performance and time specification*," Final Adopted Specification ptc/02-03-02, OMG.

[Ouardani] Adel Ouardani, Philippe Esteban, Mario Paludetto, Jean-Claude Pascal, "*A Meta-modeling Approach for Sequence Diagrams to Petri Nets Transformation within the requirements validation process*," 2006 European Simulation and Modeling Conférence (ESM'2006), Toulouse (France), 23-25 Octobre 2006, pp.345-349

[Pooley 99] Pooley, R., *"Using UML to Derive Stochastic Process Algebra Models,"* Proc. of XV UK Performance Engineering Workshop, 1999.

[Saldhana] John Anil Saldhana, Sol M. Shatz, "*UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis*," Proceedings of the Int. Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, pages 103-110. July 2000.

[Sarukkai] S.R. Sarukkai, P. Mehra, and R.J. Block, "*Automated Scalability Analysis of Message-Passing Parallel Programs*," IEEE Parallel and Distributed Technology, vol. 3, no. 4, pp. 21–32, Winter 1995.

[Schürr] Schürr, A., 1994. *"PROGRES: A Visual Language and Environment for PROgramming with Graph Rewrite Systems"*, Technical Report AIB 94-11, RWTH Aachen, Germany.

[Smith 90] C. Smith, "*Performance Engineering of Software Systems*," Addison-Wesley Publishing Co., New York, NY, 1990.

[Smith 93] C. U. Smith, L. G. Williams, "*Software performance engineering: A case study including performance comparison with design alternatives*," IEEE Transactions on Software Engineering, 19(7):72&741, July 1993.

[Tauseef 05] Tauseef A. Israr, Danny H. Lau, Greg Franks, Murray Woodside, "*Automatic Generation of Layered Queuing Software Performance Models from Commonly Available Traces*," Proceedings of the Fifth International Workshop on Software and Performancee, pp. 147-158, Palma, Illes Balears, Spain, July 12 – 14, 2005

[Tauseef 07] Tauseef Israr, Murray Woodside, and Greg Franks, "*Interaction tree algorithms to extract effective architecture and layered performance models from traces*," Journal of Systems and Software Volume 80, Issue 4, April 2007, Pages 474-492. Software Performance, 5th International Workshop on Software and Performance

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

[Woodside 05] Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Hui Shen, Toqeer Israr, Jose Merseguer , "*Performance by Unified Model Analysis (PUMA),*" Proceedings of the 5th international workshop on Software and performance WOSP '05, pp. 1-12, July 2005

[Woodside 95] Woodside, C.M., Neilson, J.E., Petriu, D.C., & Majumdar, S., "*The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-Like Distributed Software,*" IEEE Transactions on Computers, Vol. 44, Nb. 1, January 1995.

[Yan] J. Yan, "*Performance Tuning with AIMS - an Automated Instrumentation and Monitoring System for Multicomputers*," Proc. 27th Hawaii Int'l Conf. System Sciences, Jan. 1994.

[Zeng] Yong Xiang Zeng, "*Transforming Use Case Maps to the Core Scenario Model Representation*," Master Thesis, University of Ottawa, Ottawa, Canada, 2005

# Appendix A Execution Traces

## *Appendix A1 Use Case: Replenish Inventory and Notify Billing*

10000004 send Server Book_START
10000005 receive Book1 Book_START
10000012 send Book1 Book_END
10000021 receive Server Book_END
10000031 send Server setName_START
10000039 receive Book1 setName_START
10000043 send Book1 setName_END
10000050 receive Server setName_END
10000052 send Server setAuthor_START
10000054 receive Book1 setAuthor_START
10000064 send Book1 setAuthor_END
10000066 receive Server setAuthor_END
10000076 send Server setPublisher_START
10000079 receive Book1 setPublisher_START
10000083 send Book1 setPublisher_END
10000090 receive Server setPublisher_END
10000093 send Server setPublishingDate_START
10000094 receive Book1 setPublishingDate_START
10000101 send Book1 setPublishingDate_END
10000107 receive Server setPublishingDate_END
10000116 send Server setSubject_START
10000119 receive Book1 setSubject_START
10000128 send Book1 setSubject_END
10000137 receive Server setSubject_END
10000146 send Server setDescription_START
10000155 receive Book1 setDescription_START
10000157 send Book1 setDescription_END
10000167 receive Server setDescription_END
10000173 send Server setISBN_START
10000175 receive Book1 setISBN_START
10000181 send Book1 setISBN_END
10000188 receive Server setISBN_END
10000198 send Server setPrice_START
10000205 receive Book1 setPrice_START
10000212 send Book1 setPrice_END
10000216 receive Server setPrice_END
10000223 send Server setInStock_START
10000226 receive Book1 setInStock_START
10000235 send Book1 setInStock_END
10000239 receive Server setInStock_END
10000245 send Server Book_START
10000248 receive Book2 Book_START
10000258 send Book2 Book_END
10000268 receive Server Book_END
10000275 send Server setName_START
10000281 receive Book2 setName_START

10000284 send Book2 setName_END
10000286 receive Server setName_END
10000287 send Server setAuthor_START
10000296 receive Book2 setAuthor_START
10000302 send Book2 setAuthor_END
10000310 receive Server setAuthor_END
10000313 send Server setPublisher_START
10000314 receive Book2 setPublisher_START
10000321 send Book2 setPublisher_END
10000330 receive Server setPublisher_END
10000337 send Server setPublishingDate_START
10000342 receive Book2 setPublishingDate_START
10000344 send Book2 setPublishingDate_END
10000351 receive Server setPublishingDate_END
10000357 send Server setSubject_START
10000360 receive Book2 setSubject_START
10000362 send Book2 setSubject_END
10000367 receive Server setSubject_END
10000374 send Server setDescription_START
10000380 receive Book2 setDescription_START
10000390 send Book2 setDescription_END
10000394 receive Server setDescription_END
10000401 send Server setISBN_START
10000403 receive Book2 setISBN_START
10000409 send Book2 setISBN_END
10000413 receive Server setISBN_END
10000420 send Server setPrice_START
10000427 receive Book2 setPrice_START
10000430 send Book2 setPrice_END
10000433 receive Server setPrice_END
10000435 send Server setInStock_START
10000444 receive Book2 setInStock_START
10000453 send Book2 setInStock_END
10000458 receive Server setInStock_END
10000468 send Server addProduct_START
10000475 receive Inventory addProduct_START
10000483 send Inventory addProduct_END
10000488 receive Server addProduct_END
10000493 send Server addProduct_START
10000503 receive Inventory addProduct_START
10000509 send Inventory addProduct_END
10000510 receive Server addProduct_END
10000520 send Server Replenish_START
10000528 receive Replenish Replenish_START
10000532 send Replenish Replenish_END
10000534 receive Server Replenish_END
10000542 send Server start_START
10000546 receive Replenish1 start_START
10000552 send Replenish1 lock_START
10000554 receive Inventory lock_START
10000556 send Inventory lock_END
10000563 receive Replenish1 lock_END

10000565 send Replenish1 replenish_START
10000569 receive Inventory replenish_START
10000570 send Inventory getName_START
10000577 receive Book2 getName_START
10000583 send Book2 getName_END
10000592 receive Inventory getName_END
10000595 send Inventory addInStock_START
10000599 receive Book2 addInStock_START
10000602 send Book2 addInStock_END
10000612 receive Inventory addInStock_END
10000619 send Inventory getName_START
10000625 receive Book2 getName_START
10000630 send Book2 getName_END
10000637 receive Inventory getName_END
10000645 send Inventory addInStock_START
10000655 receive Book2 addInStock_START
10000662 send Book2 addInStock_END
10000670 receive Inventory addInStock_END
10000680 send Inventory Notify_START
10000685 receive Billing Notify_START
10000691 send Billing Notify_END
10000698 receive Inventory Notify_END
10000703 send Inventory replenish_END
10000706 receive Replenish1 replenish_END
10000710 send Replenish1 releaseLock_START
10000717 receive Inventory releaseLock_START
10000725 send Inventory releaseLock_END
10000735 receive Replenish1 releaseLock_END
10000739 send Replenish1 lock_START
10000747 receive Inventory lock_START
10000754 send Inventory lock_END
10000758 receive Replenish1 lock_END
10000767 send Replenish1 replenish_START
10000772 receive Inventory replenish_START
10000782 send Inventory getName_START
10000791 receive Book2 getName_START
10000793 send Book2 getName_END
10000800 receive Inventory getName_END
10000805 send Inventory addInStock_START
10000809 receive Book2 addInStock_START
10000812 send Book2 addInStock_END
10000821 receive Inventory addInStock_END
10000826 send Inventory getName_START
10000830 receive Book2 getName_START
10000833 send Book2 getName_END
10000840 receive Inventory getName_END
10000841 send Inventory addInStock_START
10000851 receive Book2 addInStock_START
10000859 send Book2 addInStock_END
10000861 receive Inventory addInStock_END
10000869 send Inventory Notify_START
10000874 receive Billing Notify_START

10000881 send Billing Notify_END
10000890 receive Inventory Notify_END
10000899 send Inventory replenish_END
10000902 receive Replenish1 replenish_END
10000908 send Replenish1 releaseLock_START
10000915 receive Inventory releaseLock_START
10000920 send Inventory releaseLock_END
10000926 receive Replenish1 releaseLock_END
10000934 send Server ShoppingCart_START
10000939 receive ShoppingCart1 ShoppingCart_START
10000948 send ShoppingCart1 ShoppingCart_END
10000950 receive Server ShoppingCart_END

## *Appendix A2 Use Case: Create Account, Login and Add to Cart*

10000957 send Client browse_STARTC
10000962 receive Server browse_STARTC
10000963 send Server display_START
10000970 receive Inventory display_START
10000975 send Inventory getName_START
10000981 receive Book2 getName_START
10000986 send Book2 getName_END
10000996 receive Inventory getName_END
10000997 send Inventory getName_START
10001001 receive Book2 getName_START
10001007 send Book2 getName_END
10001016 receive Inventory getName_END
10001026 send Inventory display_END
10001028 receive Server display_END
10001036 send Server browse_ENDC
10001045 receive Client browse_ENDC
10001048 send Client createAccount_STARTC
10001053 receive Server createAccount_STARTC
10001061 send Server exists_START
10001071 receive CustomerAccount exists_START
10001073 send CustomerAccount exists_END
10001082 receive Server exists_END
10001090 send Server CustomerAccount_START
10001091 receive CAccount1 CustomerAccount_START
10001100 send CAccount1 CustomerAccount_END
10001103 receive Server CustomerAccount_END
10001113 send Server addCustomer_START
10001122 receive CustomerAccount addCustomer_START
10001128 send CustomerAccount addCustomer_END
10001132 receive Server addCustomer_END
10001133 send Server createAccount_ENDC
10001138 receive Client createAccount_ENDC
10001145 send Client login_STARTC
10001148 receive Server login_STARTC
10001154 send Server exists_START
10001163 receive CustomerAccount exists_START

10001166 send CustomerAccount getLogin_START
10001168 receive CAccount1 getLogin_START
10001170 send CAccount1 getLogin_END
10001178 receive CustomerAccount getLogin_END
10001186 send CustomerAccount getPassword_START
10001187 receive CAccount1 getPassword_START
10001195 send CAccount1 getPassword_END
10001201 receive CustomerAccount getPassword_END
10001207 send CustomerAccount exists_END
10001214 receive Server exists_END
10001219 send Server getAccount_START
10001223 receive CustomerAccount getAccount_START
10001230 send CustomerAccount getLogin_START
10001236 receive CAccount1 getLogin_START
10001244 send CAccount1 getLogin_END
10001253 receive CustomerAccount getLogin_END
10001260 send CustomerAccount getPassword_START
10001268 receive CAccount1 getPassword_START
10001276 send CAccount1 getPassword_END
10001280 receive CustomerAccount getPassword_END
10001281 send CustomerAccount getAccount_END
10001287 receive Server getAccount_END
10001295 send Server login_ENDC
10001302 receive Client login_ENDC
10001304 send Client displayProductInfo_STARTC
10001309 receive Server displayProductInfo_STARTC
10001319 send Server displayProduct_START
10001326 receive Inventory displayProduct_START
10001331 send Inventory display_START
10001334 receive Book2 display_START
10001338 send Book2 display_END
10001341 receive Inventory display_END
10001349 send Inventory displayProduct_END
10001356 receive Server displayProduct_END
10001362 send Server displayProductInfo_ENDC
10001366 receive Client displayProductInfo_ENDC
10001372 send Client addProduct_STARTC
10001380 receive Server addProduct_STARTC
10001386 send Server getProduct_START
10001394 receive Inventory getProduct_START
10001395 send Inventory getProduct_END
10001396 receive Server getProduct_END
10001400 send Server addProduct_START
10001403 receive ShoppingCart1 addProduct_START
10001409 send ShoppingCart1 addProduct_END
10001418 receive Server addProduct_END
10001425 send Server addProduct_ENDC
10001427 receive Client addProduct_ENDC

## *Appendix A3 Checkout scenario*

10001431 send Client checkOut_STARTC
10001438 receive Server checkOut_STARTC
10001440 send Server isEmpty_START
10001445 receive ShoppingCart1 isEmpty_START
10001454 send ShoppingCart1 isEmpty_END
10001463 receive Server isEmpty_END
10001464 send Server displayWithPrice_START
10001470 receive ShoppingCart1 displayWithPrice_START
10001473 send ShoppingCart1 getName_START
10001483 receive Book2 getName_START
10001488 send Book2 getName_END
10001490 receive ShoppingCart1 getName_END
10001493 send ShoppingCart1 getPrice_START
10001498 receive Book2 getPrice_START
10001499 send Book2 getPrice_END
10001505 receive ShoppingCart1 getPrice_END
10001513 send ShoppingCart1 displayWithPrice_END
10001523 receive Server displayWithPrice_END
10001525 send Server getTotal_START
10001529 receive ShoppingCart1 getTotal_START
10001539 send ShoppingCart1 getPrice_START
10001548 receive Book2 getPrice_START
10001552 send Book2 getPrice_END
10001556 receive ShoppingCart1 getPrice_END
10001566 send ShoppingCart1 getTotal_END
10001567 receive Server getTotal_END
10001569 send Server getNewOrderNumber_START
10001575 receive Order getNewOrderNumber_START
10001579 send Order getNewOrderNumber_END
10001587 receive Server getNewOrderNumber_END
10001593 send Server Order_START
10001594 receive Order Order_START
10001595 send Order Order_END
10001596 receive Server Order_END
10001603 send Server Shipping_START
10001606 receive Shipping Shipping_START
10001615 send Shipping Shipping_END
10001624 receive Server Shipping_END
10001627 send Server start_START
10001628 send Server checkOut_ENDC
10001637 receive Shipping1 start_START
10001643 receive Client checkOut_ENDC
10001644 send Shipping1 getShoppingCart_START
10001648 receive radsbookstore.Order@17ce4e7 getShoppingCart_START
10001651 send radsbookstore.Order@17ce4e7 getShoppingCart_END
10001661 receive Shipping1 getShoppingCart_END
10001671 send Shipping1 getList_START
10001674 receive ShoppingCart1 getList_START
10001676 send ShoppingCart1 getList_END
10001682 receive Shipping1 getList_END
10001689 send Shipping1 ShoppingCart_START
10001698 receive ShoppingCart2 ShoppingCart_START

10001706 send ShoppingCart2 ShoppingCart_END
10001715 receive Shipping1 ShoppingCart_END
10001717 send Shipping1 ShoppingCart_START
10001726 receive ShoppingCart3 ShoppingCart_START
10001735 send ShoppingCart3 ShoppingCart_END
10001739 receive Shipping1 ShoppingCart_END
10001741 send Shipping1 lock_START
10001745 receive Inventory lock_START
10001752 send Inventory lock_END
10001759 receive Shipping1 lock_END
10001762 send Shipping1 getInStock_START
10001772 receive Book2 getInStock_START
10001775 send Book2 getInStock_END
10001776 receive Shipping1 getInStock_END
10001777 send Shipping1 setInStock_START
10001778 receive Book2 setInStock_START
10001785 send Book2 setInStock_END
10001787 receive Shipping1 setInStock_END
10001790 send Shipping1 addProduct_START
10001791 receive ShoppingCart3 addProduct_START
10001798 send ShoppingCart3 addProduct_END
10001799 receive Shipping1 addProduct_END
10001804 send Shipping1 getName_START
10001814 receive Book2 getName_START
10001818 send Book2 getName_END
10001824 receive Shipping1 getName_END
10001829 send Shipping1 releaseLock_START
10001839 receive Inventory releaseLock_START
10001847 send Inventory releaseLock_END
10001852 receive Shipping1 releaseLock_END
10001860 send Shipping1 getClient_START
10001868 receive radsbookstore.Order@17ce4e7 getClient_START
10001875 send radsbookstore.Order@17ce4e7 getClient_END
10001878 receive Shipping1 getClient_END
10001884 send Shipping1 isEmpty_START
10001886 receive ShoppingCart3 isEmpty_START
10001890 send ShoppingCart3 isEmpty_END
10001892 receive Shipping1 isEmpty_END
10001902 send Shipping1 isEmpty_START
10001903 receive ShoppingCart3 isEmpty_START
10001912 send ShoppingCart3 isEmpty_END
10001920 receive Shipping1 isEmpty_END
10001924 send Shipping1 Order_START
10001926 receive Order Order_START
10001933 send radsbookstore.Order@982589 getOrderNumber_START
10001940 receive radsbookstore.Order@17ce4e7 getOrderNumber_START
10001941 send radsbookstore.Order@17ce4e7 getOrderNumber_END
10001948 receive radsbookstore.Order@982589 getOrderNumber_END
10001951 send radsbookstore.Order@982589 getClient_START
10001955 receive radsbookstore.Order@17ce4e7 getClient_START
10001965 send radsbookstore.Order@17ce4e7 getClient_END
10001967 receive radsbookstore.Order@982589 getClient_END

10001973 send Order Order_END
10001977 receive Shipping1 Order_END
10001983 send Shipping1 addPrevOrder_START
10001984 receive CAccount1 addPrevOrder_START
10001987 send CAccount1 addPrevOrder_END
10001994 receive Shipping1 addPrevOrder_END
10002001 send Shipping1 Billing_START
10002008 receive radsbookstore.Billing@c88440 Billing_START
10002013 send radsbookstore.Billing@c88440 display_START
10002023 receive radsbookstore.Order@982589 display_START
10002029 send radsbookstore.Order@982589 display_START
10002030 receive CAccount1 display_START
10002034 send CAccount1 display_END
10002042 receive radsbookstore.Order@982589 display_END
10002052 send radsbookstore.Order@982589 getCreditCard_START
10002055 receive CAccount1 getCreditCard_START
10002056 send CAccount1 getCreditCard_END
10002062 receive radsbookstore.Order@982589 getCreditCard_END
10002066 send radsbookstore.Order@982589 displayWithPrice_START
10002073 receive ShoppingCart3 displayWithPrice_START
10002075 send ShoppingCart3 getName_START
10002083 receive Book2 getName_START
10002093 send Book2 getName_END
10002102 receive ShoppingCart3 getName_END
10002108 send ShoppingCart3 getPrice_START
10002111 receive Book2 getPrice_START
10002113 send Book2 getPrice_END
10002122 receive ShoppingCart3 getPrice_END
10002132 send ShoppingCart3 displayWithPrice_END
10002135 receive radsbookstore.Order@982589 displayWithPrice_END
10002145 send radsbookstore.Order@982589 display_END
10002151 receive radsbookstore.Billing@c88440 display_END
10002159 send radsbookstore.Billing@c88440 getShoppingCart_START
10002169 receive radsbookstore.Order@982589 getShoppingCart_START
10002174 send radsbookstore.Order@982589 getShoppingCart_END
10002182 receive radsbookstore.Billing@c88440 getShoppingCart_END
10002189 send radsbookstore.Billing@c88440 getList_START
10002198 receive ShoppingCart3 getList_START
10002200 send ShoppingCart3 getList_END
10002209 receive radsbookstore.Billing@c88440 getList_END
10002212 send radsbookstore.Billing@c88440 getPrice_START
10002221 receive Book2 getPrice_START
10002228 send Book2 getPrice_END
10002231 receive radsbookstore.Billing@c88440 getPrice_END
10002240 send radsbookstore.Billing@c88440 Billing_END
10002245 receive Shipping1 Billing_END

# Appendix B CSM XML File

## *Appendix B1Synchronous Communication Pattern*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id7" hostDemand="44" name="browse_STARTC"
predecessor="id14" successor="id10"/>
  <Step id="id3" Component="id8" hostDemand="63" name="display_START" predecessor="id18"
successor="id12"/>
  <ResourceAcquire id="id13" predecessor="id22" successor="id14" acquire="id7"/>
  <ResourceAcquire id="id17" predecessor="id10" successor="id18" acquire="id8"/>
  <ResourceAcquire id="id21" predecessor="id1" successor="id22" acquire="id6"/>
  <ResourceRelease id="id5" predecessor="id11" successor="id24" release="id7"/>
  <ResourceRelease id="id4" predecessor="id12" successor="id11" release="id8"/>
  <ResourceRelease id="id23" predecessor="id24" successor="id9" release="id6"/>
  <Start id="id1" target="id21">
   <ClosedWorkload id="id25"/>
  </Start>
  <End id="id9" Classifier="id16" source="id23"/>
  <Sequence id="id10" source="id2" target="id17"/>
  <Sequence id="id11" Classifier="id20" source="id4" target="id5"/>
  <Sequence id="id12" source="id3" target="id4"/>
  <Sequence id="id14" Classifier="id15" source="id13" target="id2"/>
  <Sequence id="id18" Classifier="id19" source="id17" target="id3"/>
  <Sequence id="id22" source="id21" target="id13"/>
  <Sequence id="id24" source="id5" target="id23"/>
  <Classifier xsi:type="CSM:MessageType" id="id15" name="browse_STARTC" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id16" name="browse_ENDC" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id19" name="display_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id20" name="display_END" kind="reply"/>
 </Scenario>
 <Component id="id6" name="Client"/>
 <Component id="id7" name="Server"/>
 <Component id="id8" name="Inventory"/>
</CSM:CSMType>
```

## *Appendix B2Asynchronous Communication Pattern*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id5" hostDemand="18" name="browse_STARTC"
predecessor="id10" successor="id13"/>
  <Step id="id3" Component="id6" name="display_START" predecessor="id15"
successor="id18"/>
  <ResourceAcquire id="id9" predecessor="id22" successor="id10" acquire="id5"/>
```

```
  <ResourceAcquire id="id14" predecessor="id8" successor="id15" acquire="id6"/>
  <ResourceAcquire id="id19" predecessor="id1" successor="id20" acquire="id4"/>
  <ResourceRelease id="id12" predecessor="id13" successor="id8" release="id5"/>
  <ResourceRelease id="id17" predecessor="id18" successor="id7" release="id6"/>
  <ResourceRelease id="id21" predecessor="id20" successor="id22" release="id4"/>
  <Start id="id1" target="id19">
   <ClosedWorkload id="id23"/>
  </Start>
  <End id="id7" source="id17"/>
  <Sequence id="id8" source="id12" target="id14"/>
  <Sequence id="id10" Classifier="id11" source="id9" target="id2"/>
  <Sequence id="id13" source="id2" target="id12"/>
  <Sequence id="id15" Classifier="id16" source="id14" target="id3"/>
  <Sequence id="id18" source="id3" target="id17"/>
  <Sequence id="id20" source="id19" target="id21"/>
  <Sequence id="id22" source="id21" target="id9"/>
  <Classifier xsi:type="CSM:MessageType" id="id11" name="browse_STARTC" kind="async"/>
  <Classifier xsi:type="CSM:MessageType" id="id16" name="display_START" kind="async"/>
 </Scenario>
 <Component id="id4" name="Client"/>
 <Component id="id5" name="Server"/>
 <Component id="id6" name="Inventory"/>
</CSM:CSMType>
```

## *Appendix B3 Forwarding Communication Pattern*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id6" hostDemand="18" name="browse_STARTC"
predecessor="id17" successor="id20"/>
  <Step id="id4" Component="id5" name="getName_START" predecessor="id12"
successor="id15"/>
  <Step id="id3" Component="id7" hostDemand="19" name="display_START" predecessor="id22"
successor="id25"/>
  <ResourceAcquire id="id11" predecessor="id10" successor="id12" acquire="id5"/>
  <ResourceAcquire id="id16" predecessor="id29" successor="id17" acquire="id6"/>
  <ResourceAcquire id="id21" predecessor="id9" successor="id22" acquire="id7"/>
  <ResourceAcquire id="id26" predecessor="id1" successor="id27" acquire="id5"/>
  <ResourceRelease id="id14" predecessor="id15" successor="id8" release="id5"/>
  <ResourceRelease id="id19" predecessor="id20" successor="id9" release="id6"/>
  <ResourceRelease id="id24" predecessor="id25" successor="id10" release="id7"/>
  <ResourceRelease id="id28" predecessor="id27" successor="id29" release="id5"/>
  <Start id="id1" target="id26">
   <ClosedWorkload id="id30"/>
  </Start>
  <End id="id8" source="id14"/>
  <Sequence id="id9" source="id19" target="id21"/>
  <Sequence id="id10" source="id24" target="id11"/>
  <Sequence id="id12" Classifier="id13" source="id11" target="id4"/>
  <Sequence id="id15" source="id4" target="id14"/>
```

```
  <Sequence id="id17" Classifier="id18" source="id16" target="id2"/>
  <Sequence id="id20" source="id2" target="id19"/>
  <Sequence id="id22" Classifier="id23" source="id21" target="id3"/>
  <Sequence id="id25" source="id3" target="id24"/>
  <Sequence id="id27" source="id26" target="id28"/>
  <Sequence id="id29" source="id28" target="id16"/>
  <Classifier xsi:type="CSM:MessageType" id="id13" name="getName_START" kind="async"/>
  <Classifier xsi:type="CSM:MessageType" id="id18" name="browse_STARTC" kind="async"/>
  <Classifier xsi:type="CSM:MessageType" id="id23" name="display_START" kind="async"/>
 </Scenario>
 <Component id="id5" name="Client"/>
 <Component id="id6" name="Server"/>
 <Component id="id7" name="Inventory"/>
</CSM:CSMType>
```

## *Appendix B4 Synchronous pattern with nested interaction*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id9" hostDemand="44" name="browse_STARTC"
predecessor="id19" successor="id13"/>
  <Step id="id3" Component="id10" hostDemand="48" name="display_START"
predecessor="id23" successor="id15"/>
  <Step id="id4" Component="id11" hostDemand="15" name="getName_START"
predecessor="id27" successor="id17"/>
  <ResourceAcquire id="id18" predecessor="id31" successor="id19" acquire="id9"/>
  <ResourceAcquire id="id22" predecessor="id13" successor="id23" acquire="id10"/>
  <ResourceAcquire id="id26" predecessor="id15" successor="id27" acquire="id11"/>
  <ResourceAcquire id="id30" predecessor="id1" successor="id31" acquire="id8"/>
  <ResourceRelease id="id7" predecessor="id14" successor="id33" release="id9"/>
  <ResourceRelease id="id6" predecessor="id16" successor="id14" release="id10"/>
  <ResourceRelease id="id5" predecessor="id17" successor="id16" release="id11"/>
  <ResourceRelease id="id32" predecessor="id33" successor="id12" release="id8"/>
  <Start id="id1" target="id30">
   <ClosedWorkload id="id34"/>
  </Start>
  <End id="id12" Classifier="id21" source="id32"/>
  <Sequence id="id13" source="id2" target="id22"/>
  <Sequence id="id14" Classifier="id25" source="id6" target="id7"/>
  <Sequence id="id15" source="id3" target="id26"/>
  <Sequence id="id16" Classifier="id29" source="id5" target="id6"/>
  <Sequence id="id17" source="id4" target="id5"/>
  <Sequence id="id19" Classifier="id20" source="id18" target="id2"/>
  <Sequence id="id23" Classifier="id24" source="id22" target="id3"/>
  <Sequence id="id27" Classifier="id28" source="id26" target="id4"/>
  <Sequence id="id31" source="id30" target="id18"/>
  <Sequence id="id33" source="id7" target="id32"/>
  <Classifier xsi:type="CSM:MessageType" id="id20" name="browse_STARTC" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id21" name="browse_ENDC" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id24" name="display_START" kind="sync"/>
```

```
  <Classifier xsi:type="CSM:MessageType" id="id25" name="display_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id28" name="getName_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id29" name="getName_END" kind="reply"/>
 </Scenario>
 <Component id="id8" name="Client"/>
 <Component id="id9" name="Server"/>
 <Component id="id10" name="Inventory"/>
 <Component id="id11" name="Book1"/>
</CSM:CSMType>
```

## *Appendix B5 Synchronous Communication Pattern with Fork*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id8" hostDemand="40" name="browse_STARTC"
predecessor="id17" successor="id13"/>
  <Step id="id3" Component="id9" name="display_START" predecessor="id21"
successor="id24"/>
  <Step id="id4" Component="id10" hostDemand="67" name="displayBook_STARTC"
predecessor="id26" successor="id15"/>
  <ResourceAcquire id="id16" predecessor="id30" successor="id17" acquire="id8"/>
  <ResourceAcquire id="id20" predecessor="id13" successor="id21" acquire="id9"/>
  <ResourceAcquire id="id25" predecessor="id13" successor="id26" acquire="id10"/>
  <ResourceAcquire id="id29" predecessor="id1" successor="id30" acquire="id7"/>
  <ResourceRelease id="id6" predecessor="id14" successor="id32" release="id8"/>
  <ResourceRelease id="id23" predecessor="id24" successor="id12" release="id9"/>
  <ResourceRelease id="id5" predecessor="id15" successor="id14" release="id10"/>
  <ResourceRelease id="id31" predecessor="id32" successor="id11" release="id7"/>
  <Start id="id1" target="id29">
   <ClosedWorkload id="id33"/>
  </Start>
  <End id="id11" Classifier="id19" source="id31"/>
  <End id="id12" source="id23"/>
  <Sequence id="id14" Classifier="id28" source="id5" target="id6"/>
  <Sequence id="id15" source="id4" target="id5"/>
  <Sequence id="id17" Classifier="id18" source="id16" target="id2"/>
  <Sequence id="id21" Classifier="id22" source="id20" target="id3"/>
  <Sequence id="id24" source="id3" target="id23"/>
  <Sequence id="id26" Classifier="id27" source="id25" target="id4"/>
  <Sequence id="id30" source="id29" target="id16"/>
  <Sequence id="id32" source="id6" target="id31"/>
  <Fork id="id13" source="id2" target="id20 id25"/>
  <Classifier xsi:type="CSM:MessageType" id="id18" name="browse_STARTC" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id19" name="browse_ENDC" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id22" name="display_START" kind="async"/>
  <Classifier xsi:type="CSM:MessageType" id="id27" name="displayBook_STARTC"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id28" name="displayBook_END" kind="reply"/>
 </Scenario>
 <Component id="id7" name="Client"/>
```

```
 <Component id="id8" name="Server"/>
 <Component id="id9" name="Inventory"/>
 <Component id="id10" name="Book"/>
</CSM:CSMType>
```

## *Appendix B6 Nested Synchronous Communication Pattern with Fork and Join*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id9" hostDemand="69" name="browse_STARTC"
predecessor="id18" successor="id13"/>
  <Step id="id3" Component="id10" hostDemand="43" name="display_START"
predecessor="id22" successor="id15"/>
  <Step id="id4" Component="id11" hostDemand="67" name="displayBook_STARTC"
predecessor="id26" successor="id16"/>
  <ResourceAcquire id="id17" predecessor="id30" successor="id18" acquire="id9"/>
  <ResourceAcquire id="id21" predecessor="id13" successor="id22" acquire="id10"/>
  <ResourceAcquire id="id25" predecessor="id13" successor="id26" acquire="id11"/>
  <ResourceAcquire id="id29" predecessor="id1" successor="id30" acquire="id8"/>
  <ResourceRelease id="id7" predecessor="id14" successor="id32" release="id9"/>
  <ResourceRelease id="id5" predecessor="id15" successor="id14" release="id10"/>
  <ResourceRelease id="id6" predecessor="id16" successor="id14" release="id11"/>
  <ResourceRelease id="id31" predecessor="id32" successor="id12" release="id8"/>
  <Start id="id1" target="id29">
   <ClosedWorkload id="id33"/>
  </Start>
  <End id="id12" Classifier="id20" source="id31"/>
  <Sequence id="id15" source="id3" target="id5"/>
  <Sequence id="id16" source="id4" target="id6"/>
  <Sequence id="id18" Classifier="id19" source="id17" target="id2"/>
  <Sequence id="id22" Classifier="id23" source="id21" target="id3"/>
  <Sequence id="id26" Classifier="id27" source="id25" target="id4"/>
  <Sequence id="id30" source="id29" target="id17"/>
  <Sequence id="id32" source="id7" target="id31"/>
  <Fork id="id13" source="id2" target="id21 id25"/>
  <Join id="id14" Classifier="id24 id28" source="id5 id6" target="id7"/>
  <Classifier xsi:type="CSM:MessageType" id="id19" name="browse_STARTC" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id20" name="browse_ENDC" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id23" name="display_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id24" name="display_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id27" name="displayBook_STARTC"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id28" name="displayBook_END" kind="reply"/>
 </Scenario>
 <Component id="id8" name="Client"/>
 <Component id="id9" name="Server"/>
 <Component id="id10" name="Inventory"/>
 <Component id="id11" name="Book"/>
</CSM:CSMType>
```

## Appendix B7 Asynchronous Communication Pattern with Fork

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id6" hostDemand="25" name="browse_STARTC"
predecessor="id13" successor="id16"/>
  <Step id="id3" Component="id7" name="display_START" predecessor="id18"
successor="id21"/>
  <Step id="id4" Component="id8" name="displayBook_START" predecessor="id23"
successor="id26"/>
  <ResourceAcquire id="id12" predecessor="id30" successor="id13" acquire="id6"/>
  <ResourceAcquire id="id17" predecessor="id11" successor="id18" acquire="id7"/>
  <ResourceAcquire id="id22" predecessor="id11" successor="id23" acquire="id8"/>
  <ResourceAcquire id="id27" predecessor="id1" successor="id28" acquire="id5"/>
  <ResourceRelease id="id15" predecessor="id16" successor="id11" release="id6"/>
  <ResourceRelease id="id20" predecessor="id21" successor="id9" release="id7"/>
  <ResourceRelease id="id25" predecessor="id26" successor="id10" release="id8"/>
  <ResourceRelease id="id29" predecessor="id28" successor="id30" release="id5"/>
  <Start id="id1" target="id27">
   <ClosedWorkload id="id31"/>
  </Start>
  <End id="id9" source="id20"/>
  <End id="id10" source="id25"/>
  <Sequence id="id13" Classifier="id14" source="id12" target="id2"/>
  <Sequence id="id16" source="id2" target="id15"/>
  <Sequence id="id18" Classifier="id19" source="id17" target="id3"/>
  <Sequence id="id21" source="id3" target="id20"/>
  <Sequence id="id23" Classifier="id24" source="id22" target="id4"/>
  <Sequence id="id26" source="id4" target="id25"/>
  <Sequence id="id28" source="id27" target="id29"/>
  <Sequence id="id30" source="id29" target="id12"/>
  <Fork id="id11" source="id15" target="id17 id22"/>
  <Classifier xsi:type="CSM:MessageType" id="id14" name="browse_STARTC" kind="async"/>
  <Classifier xsi:type="CSM:MessageType" id="id19" name="display_START" kind="async"/>
  <Classifier xsi:type="CSM:MessageType" id="id24" name="displayBook_START"
kind="async"/>
 </Scenario>
 <Component id="id5" name="Client"/>
 <Component id="id6" name="Server"/>
 <Component id="id7" name="Inventory"/>
 <Component id="id8" name="Book"/>
</CSM:CSMType>
```

## Appendix B8 A Complicated Example

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id17" hostDemand="90" name="browse_STARTC"
predecessor="id35" successor="id26"/>
```

```
  <Step id="id6" Component="id19" hostDemand="82" name="createAccount" predecessor="id51"
successor="id30"/>
  <Step id="id9" Component="id18" hostDemand="20" name="getLogin_START"
predecessor="id43" successor="id28"/>
  <Step id="id10" Component="id20" hostDemand="20" name="getBalance_START"
predecessor="id59" successor="id33"/>
  <Step id="id13" Component="id18" hostDemand="20" name="getList_START"
predecessor="id47" successor="id29"/>
  <Step id="id3" Component="id18" hostDemand="63" name="display_START"
predecessor="id39" successor="id27"/>
  <Step id="id7" Component="id20" hostDemand="21" name="getPassword_START"
predecessor="id55" successor="id32"/>
  <ResourceAcquire id="id34" predecessor="id63" successor="id35" acquire="id17"/>
  <ResourceAcquire id="id38" predecessor="id26" successor="id39" acquire="id18"/>
  <ResourceAcquire id="id42" predecessor="id24" successor="id43" acquire="id18"/>
  <ResourceAcquire id="id46" predecessor="id25" successor="id47" acquire="id18"/>
  <ResourceAcquire id="id50" predecessor="id23" successor="id51" acquire="id19"/>
  <ResourceAcquire id="id54" predecessor="id30" successor="id55" acquire="id20"/>
  <ResourceAcquire id="id58" predecessor="id24" successor="id59" acquire="id20"/>
  <ResourceAcquire id="id62" predecessor="id1" successor="id63" acquire="id16"/>
  <ResourceRelease id="id5" predecessor="id23" successor="id65" release="id17"/>
  <ResourceRelease id="id4" predecessor="id27" successor="id23" release="id18"/>
  <ResourceRelease id="id11" predecessor="id28" successor="id25" release="id18"/>
  <ResourceRelease id="id14" predecessor="id29" successor="id31" release="id18"/>
  <ResourceRelease id="id15" predecessor="id31" successor="id22" release="id19"/>
  <ResourceRelease id="id8" predecessor="id32" successor="id24" release="id20"/>
  <ResourceRelease id="id12" predecessor="id33" successor="id25" release="id20"/>
  <ResourceRelease id="id64" predecessor="id65" successor="id21" release="id16"/>
  <Start id="id1" target="id62">
   <ClosedWorkload id="id66"/>
  </Start>
  <End id="id21" Classifier="id37" source="id64"/>
  <End id="id22" Classifier="id53" source="id15"/>
  <Sequence id="id26" source="id2" target="id38"/>
  <Sequence id="id27" source="id3" target="id4"/>
  <Sequence id="id28" source="id9" target="id11"/>
  <Sequence id="id29" source="id13" target="id14"/>
  <Sequence id="id30" source="id6" target="id54"/>
  <Sequence id="id31" Classifier="id49" source="id14" target="id15"/>
  <Sequence id="id32" source="id7" target="id8"/>
  <Sequence id="id33" source="id10" target="id12"/>
  <Sequence id="id35" Classifier="id36" source="id34" target="id2"/>
  <Sequence id="id39" Classifier="id40" source="id38" target="id3"/>
  <Sequence id="id43" Classifier="id44" source="id42" target="id9"/>
  <Sequence id="id47" Classifier="id48" source="id46" target="id13"/>
  <Sequence id="id51" Classifier="id52" source="id50" target="id6"/>
  <Sequence id="id55" Classifier="id56" source="id54" target="id7"/>
  <Sequence id="id59" Classifier="id60" source="id58" target="id10"/>
  <Sequence id="id63" source="id62" target="id34"/>
  <Sequence id="id65" source="id5" target="id64"/>
  <Fork id="id23" Classifier="id41" source="id4" target="id5 id50"/>
  <Fork id="id24" Classifier="id57" source="id8" target="id42 id58"/>
```

```
  <Join id="id25" Classifier="id45 id61" source="id11 id12" target="id46"/>
  <Classifier xsi:type="CSM:MessageType" id="id36" name="browse_STARTC" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id37" name="browse_ENDC" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id40" name="display_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id41" name="display_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id44" name="getLogin_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id45" name="getLogin_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id48" name="getList_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id49" name="getList_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id52" name="createAccount" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id53" name="TEST" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id56" name="getPassword_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id57" name="getPassword_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id60" name="getBalance_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id61" name="getBalance_END" kind="reply"/>
 </Scenario>
 <Component id="id16" name="Client"/>
 <Component id="id17" name="Server"/>
 <Component id="id18" name="Inventory"/>
 <Component id="id19" name="CustomerAccount"/>
 <Component id="id20" name="CAccount1"/>
</CSM:CSMType>
```

## *Appendix B9 Use Case: Replenish Inventory and Notify Billing*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id84" hostDemand="16" name="Book_START" predecessor="id173"
successor="id116"/>
  <Step id="id48" Component="id88" hostDemand="64" name="start_START" predecessor="id321"
successor="id324"/>
  <Step id="id81" Component="id90" hostDemand="11" name="ShoppingCart_START"
predecessor="id334" successor="id171"/>
  <Step id="id4" Component="id84" hostDemand="11" name="setName_START"
predecessor="id177" successor="id117"/>
  <Step id="id6" Component="id84" hostDemand="12" name="setAuthor_START"
predecessor="id181" successor="id118"/>
  <Step id="id8" Component="id84" hostDemand="11" name="setPublisher_START"
predecessor="id185" successor="id119"/>
  <Step id="id10" Component="id84" hostDemand="13" name="setPublishingDate_START"
predecessor="id189" successor="id120"/>
  <Step id="id12" Component="id84" hostDemand="18" name="setSubject_START"
predecessor="id193" successor="id121"/>
  <Step id="id14" Component="id84" hostDemand="12" name="setDescription_START"
predecessor="id197" successor="id122"/>
  <Step id="id16" Component="id84" hostDemand="13" name="setISBN_START"
predecessor="id201" successor="id123"/>
  <Step id="id18" Component="id84" hostDemand="11" name="setPrice_START"
predecessor="id205" successor="id124"/>
```

<Step id="id20" Component="id84" hostDemand="13" name="setInStock_START" predecessor="id209" successor="id125"/>
<Step id="id22" Component="id85" hostDemand="20" name="Book_START" predecessor="id213" successor="id126"/>
<Step id="id24" Component="id85" hostDemand="5" name="setName_START" predecessor="id217" successor="id127"/>
<Step id="id26" Component="id85" hostDemand="14" name="setAuthor_START" predecessor="id221" successor="id128"/>
<Step id="id28" Component="id85" hostDemand="16" name="setPublisher_START" predecessor="id225" successor="id129"/>
<Step id="id30" Component="id85" hostDemand="9" name="setPublishingDate_START" predecessor="id229" successor="id130"/>
<Step id="id32" Component="id85" hostDemand="7" name="setSubject_START" predecessor="id233" successor="id131"/>
<Step id="id34" Component="id85" hostDemand="14" name="setDescription_START" predecessor="id237" successor="id132"/>
<Step id="id36" Component="id85" hostDemand="10" name="setISBN_START" predecessor="id241" successor="id133"/>
<Step id="id38" Component="id85" hostDemand="6" name="setPrice_START" predecessor="id245" successor="id134"/>
<Step id="id40" Component="id85" hostDemand="14" name="setInStock_START" predecessor="id249" successor="id135"/>
<Step id="id42" Component="id86" hostDemand="13" name="addProduct_START" predecessor="id285" successor="id144"/>
<Step id="id44" Component="id86" hostDemand="7" name="addProduct_START" predecessor="id289" successor="id145"/>
<Step id="id46" Component="id87" hostDemand="6" name="Replenish_START" predecessor="id317" successor="id162"/>
<Step id="id52" Component="id85" hostDemand="15" name="getName_START" predecessor="id253" successor="id136"/>
<Step id="id54" Component="id85" hostDemand="13" name="addInStock_START" predecessor="id257" successor="id137"/>
<Step id="id56" Component="id85" hostDemand="12" name="getName_START" predecessor="id261" successor="id138"/>
<Step id="id58" Component="id85" hostDemand="15" name="addInStock_START" predecessor="id265" successor="id139"/>
<Step id="id68" Component="id85" hostDemand="9" name="getName_START" predecessor="id269" successor="id140"/>
<Step id="id70" Component="id85" hostDemand="12" name="addInStock_START" predecessor="id273" successor="id141"/>
<Step id="id72" Component="id85" hostDemand="10" name="getName_START" predecessor="id277" successor="id142"/>
<Step id="id74" Component="id85" hostDemand="10" name="addInStock_START" predecessor="id281" successor="id143"/>
<Step id="id49" Component="id86" hostDemand="9" name="lock_START" predecessor="id293" successor="id146"/>
<Step id="id51" Component="id86" hostDemand="69" name="replenish_START" predecessor="id297" successor="id147"/>
<Step id="id60" Component="id89" hostDemand="13" name="Notify_START" predecessor="id326" successor="id169"/>
<Step id="id63" Component="id86" hostDemand="18" name="releaseLock_START" predecessor="id301" successor="id153"/>

<Step id="id65" Component="id86" hostDemand="11" name="lock_START" predecessor="id305" successor="id154"/>
<Step id="id67" Component="id86" hostDemand="73" name="replenish_START" predecessor="id309" successor="id155"/>
<Step id="id76" Component="id89" hostDemand="16" name="Notify_START" predecessor="id330" successor="id170"/>
<Step id="id79" Component="id86" hostDemand="11" name="releaseLock_START" predecessor="id313" successor="id161"/>
<ResourceAcquire id="id172" predecessor="id338" successor="id173" acquire="id84"/>
<ResourceAcquire id="id176" predecessor="id94" successor="id177" acquire="id84"/>
<ResourceAcquire id="id180" predecessor="id95" successor="id181" acquire="id84"/>
<ResourceAcquire id="id184" predecessor="id96" successor="id185" acquire="id84"/>
<ResourceAcquire id="id188" predecessor="id97" successor="id189" acquire="id84"/>
<ResourceAcquire id="id192" predecessor="id98" successor="id193" acquire="id84"/>
<ResourceAcquire id="id196" predecessor="id99" successor="id197" acquire="id84"/>
<ResourceAcquire id="id200" predecessor="id100" successor="id201" acquire="id84"/>
<ResourceAcquire id="id204" predecessor="id101" successor="id205" acquire="id84"/>
<ResourceAcquire id="id208" predecessor="id102" successor="id209" acquire="id84"/>
<ResourceAcquire id="id212" predecessor="id103" successor="id213" acquire="id85"/>
<ResourceAcquire id="id216" predecessor="id104" successor="id217" acquire="id85"/>
<ResourceAcquire id="id220" predecessor="id105" successor="id221" acquire="id85"/>
<ResourceAcquire id="id224" predecessor="id106" successor="id225" acquire="id85"/>
<ResourceAcquire id="id228" predecessor="id107" successor="id229" acquire="id85"/>
<ResourceAcquire id="id232" predecessor="id108" successor="id233" acquire="id85"/>
<ResourceAcquire id="id236" predecessor="id109" successor="id237" acquire="id85"/>
<ResourceAcquire id="id240" predecessor="id110" successor="id241" acquire="id85"/>
<ResourceAcquire id="id244" predecessor="id111" successor="id245" acquire="id85"/>
<ResourceAcquire id="id248" predecessor="id112" successor="id249" acquire="id85"/>
<ResourceAcquire id="id252" predecessor="id147" successor="id253" acquire="id85"/>
<ResourceAcquire id="id256" predecessor="id148" successor="id257" acquire="id85"/>
<ResourceAcquire id="id260" predecessor="id149" successor="id261" acquire="id85"/>
<ResourceAcquire id="id264" predecessor="id150" successor="id265" acquire="id85"/>
<ResourceAcquire id="id268" predecessor="id155" successor="id269" acquire="id85"/>
<ResourceAcquire id="id272" predecessor="id156" successor="id273" acquire="id85"/>
<ResourceAcquire id="id276" predecessor="id157" successor="id277" acquire="id85"/>
<ResourceAcquire id="id280" predecessor="id158" successor="id281" acquire="id85"/>
<ResourceAcquire id="id284" predecessor="id113" successor="id285" acquire="id86"/>
<ResourceAcquire id="id288" predecessor="id114" successor="id289" acquire="id86"/>
<ResourceAcquire id="id292" predecessor="id163" successor="id293" acquire="id86"/>
<ResourceAcquire id="id296" predecessor="id164" successor="id297" acquire="id86"/>
<ResourceAcquire id="id300" predecessor="id165" successor="id301" acquire="id86"/>
<ResourceAcquire id="id304" predecessor="id166" successor="id305" acquire="id86"/>
<ResourceAcquire id="id308" predecessor="id167" successor="id309" acquire="id86"/>
<ResourceAcquire id="id312" predecessor="id168" successor="id313" acquire="id86"/>
<ResourceAcquire id="id316" predecessor="id115" successor="id317" acquire="id87"/>
<ResourceAcquire id="id320" predecessor="id93" successor="id321" acquire="id88"/>
<ResourceAcquire id="id325" predecessor="id151" successor="id326" acquire="id89"/>
<ResourceAcquire id="id329" predecessor="id159" successor="id330" acquire="id89"/>
<ResourceAcquire id="id333" predecessor="id93" successor="id334" acquire="id90"/>
<ResourceAcquire id="id337" predecessor="id1" successor="id338" acquire="id83"/>
<ResourceRelease id="id3" predecessor="id116" successor="id340" release="id84"/>
<ResourceRelease id="id5" predecessor="id117" successor="id95" release="id84"/>

```
<ResourceRelease id="id7" predecessor="id118" successor="id96" release="id84"/>
<ResourceRelease id="id9" predecessor="id119" successor="id97" release="id84"/>
<ResourceRelease id="id11" predecessor="id120" successor="id98" release="id84"/>
<ResourceRelease id="id13" predecessor="id121" successor="id99" release="id84"/>
<ResourceRelease id="id15" predecessor="id122" successor="id100" release="id84"/>
<ResourceRelease id="id17" predecessor="id123" successor="id101" release="id84"/>
<ResourceRelease id="id19" predecessor="id124" successor="id102" release="id84"/>
<ResourceRelease id="id21" predecessor="id125" successor="id103" release="id84"/>
<ResourceRelease id="id23" predecessor="id126" successor="id104" release="id85"/>
<ResourceRelease id="id25" predecessor="id127" successor="id105" release="id85"/>
<ResourceRelease id="id27" predecessor="id128" successor="id106" release="id85"/>
<ResourceRelease id="id29" predecessor="id129" successor="id107" release="id85"/>
<ResourceRelease id="id31" predecessor="id130" successor="id108" release="id85"/>
<ResourceRelease id="id33" predecessor="id131" successor="id109" release="id85"/>
<ResourceRelease id="id35" predecessor="id132" successor="id110" release="id85"/>
<ResourceRelease id="id37" predecessor="id133" successor="id111" release="id85"/>
<ResourceRelease id="id39" predecessor="id134" successor="id112" release="id85"/>
<ResourceRelease id="id41" predecessor="id135" successor="id113" release="id85"/>
<ResourceRelease id="id53" predecessor="id136" successor="id148" release="id85"/>
<ResourceRelease id="id55" predecessor="id137" successor="id149" release="id85"/>
<ResourceRelease id="id57" predecessor="id138" successor="id150" release="id85"/>
<ResourceRelease id="id59" predecessor="id139" successor="id151" release="id85"/>
<ResourceRelease id="id69" predecessor="id140" successor="id156" release="id85"/>
<ResourceRelease id="id71" predecessor="id141" successor="id157" release="id85"/>
<ResourceRelease id="id73" predecessor="id142" successor="id158" release="id85"/>
<ResourceRelease id="id75" predecessor="id143" successor="id159" release="id85"/>
<ResourceRelease id="id43" predecessor="id144" successor="id114" release="id86"/>
<ResourceRelease id="id45" predecessor="id145" successor="id115" release="id86"/>
<ResourceRelease id="id50" predecessor="id146" successor="id164" release="id86"/>
<ResourceRelease id="id62" predecessor="id152" successor="id165" release="id86"/>
<ResourceRelease id="id64" predecessor="id153" successor="id166" release="id86"/>
<ResourceRelease id="id66" predecessor="id154" successor="id167" release="id86"/>
<ResourceRelease id="id78" predecessor="id160" successor="id168" release="id86"/>
<ResourceRelease id="id80" predecessor="id161" successor="id92" release="id86"/>
<ResourceRelease id="id47" predecessor="id162" successor="id93" release="id87"/>
<ResourceRelease id="id323" predecessor="id324" successor="id163" release="id88"/>
<ResourceRelease id="id61" predecessor="id169" successor="id152" release="id89"/>
<ResourceRelease id="id77" predecessor="id170" successor="id160" release="id89"/>
<ResourceRelease id="id82" predecessor="id171" successor="id91" release="id90"/>
<ResourceRelease id="id339" predecessor="id340" successor="id94" release="id83"/>
<Start id="id1" target="id337">
 <ClosedWorkload id="id341"/>
</Start>
<End id="id91" Classifier="id336" source="id82"/>
<End id="id92" Classifier="id315" source="id80"/>
<Sequence id="id94" Classifier="id175" source="id339" target="id176"/>
<Sequence id="id95" Classifier="id179" source="id5" target="id180"/>
<Sequence id="id96" Classifier="id183" source="id7" target="id184"/>
<Sequence id="id97" Classifier="id187" source="id9" target="id188"/>
<Sequence id="id98" Classifier="id191" source="id11" target="id192"/>
<Sequence id="id99" Classifier="id195" source="id13" target="id196"/>
<Sequence id="id100" Classifier="id199" source="id15" target="id200"/>
```

<Sequence id="id101" Classifier="id203" source="id17" target="id204"/>
<Sequence id="id102" Classifier="id207" source="id19" target="id208"/>
<Sequence id="id103" Classifier="id211" source="id21" target="id212"/>
<Sequence id="id104" Classifier="id215" source="id23" target="id216"/>
<Sequence id="id105" Classifier="id219" source="id25" target="id220"/>
<Sequence id="id106" Classifier="id223" source="id27" target="id224"/>
<Sequence id="id107" Classifier="id227" source="id29" target="id228"/>
<Sequence id="id108" Classifier="id231" source="id31" target="id232"/>
<Sequence id="id109" Classifier="id235" source="id33" target="id236"/>
<Sequence id="id110" Classifier="id239" source="id35" target="id240"/>
<Sequence id="id111" Classifier="id243" source="id37" target="id244"/>
<Sequence id="id112" Classifier="id247" source="id39" target="id248"/>
<Sequence id="id113" Classifier="id251" source="id41" target="id284"/>
<Sequence id="id114" Classifier="id287" source="id43" target="id288"/>
<Sequence id="id115" Classifier="id291" source="id45" target="id316"/>
<Sequence id="id116" source="id2" target="id3"/>
<Sequence id="id117" source="id4" target="id5"/>
<Sequence id="id118" source="id6" target="id7"/>
<Sequence id="id119" source="id8" target="id9"/>
<Sequence id="id120" source="id10" target="id11"/>
<Sequence id="id121" source="id12" target="id13"/>
<Sequence id="id122" source="id14" target="id15"/>
<Sequence id="id123" source="id16" target="id17"/>
<Sequence id="id124" source="id18" target="id19"/>
<Sequence id="id125" source="id20" target="id21"/>
<Sequence id="id126" source="id22" target="id23"/>
<Sequence id="id127" source="id24" target="id25"/>
<Sequence id="id128" source="id26" target="id27"/>
<Sequence id="id129" source="id28" target="id29"/>
<Sequence id="id130" source="id30" target="id31"/>
<Sequence id="id131" source="id32" target="id33"/>
<Sequence id="id132" source="id34" target="id35"/>
<Sequence id="id133" source="id36" target="id37"/>
<Sequence id="id134" source="id38" target="id39"/>
<Sequence id="id135" source="id40" target="id41"/>
<Sequence id="id136" source="id52" target="id53"/>
<Sequence id="id137" source="id54" target="id55"/>
<Sequence id="id138" source="id56" target="id57"/>
<Sequence id="id139" source="id58" target="id59"/>
<Sequence id="id140" source="id68" target="id69"/>
<Sequence id="id141" source="id70" target="id71"/>
<Sequence id="id142" source="id72" target="id73"/>
<Sequence id="id143" source="id74" target="id75"/>
<Sequence id="id144" source="id42" target="id43"/>
<Sequence id="id145" source="id44" target="id45"/>
<Sequence id="id146" source="id49" target="id50"/>
<Sequence id="id147" source="id51" target="id252"/>
<Sequence id="id148" Classifier="id255" source="id53" target="id256"/>
<Sequence id="id149" Classifier="id259" source="id55" target="id260"/>
<Sequence id="id150" Classifier="id263" source="id57" target="id264"/>
<Sequence id="id151" Classifier="id267" source="id59" target="id325"/>
<Sequence id="id152" Classifier="id328" source="id61" target="id62"/>

<Sequence id="id153" source="id63" target="id64"/>
<Sequence id="id154" source="id65" target="id66"/>
<Sequence id="id155" source="id67" target="id268"/>
<Sequence id="id156" Classifier="id271" source="id69" target="id272"/>
<Sequence id="id157" Classifier="id275" source="id71" target="id276"/>
<Sequence id="id158" Classifier="id279" source="id73" target="id280"/>
<Sequence id="id159" Classifier="id283" source="id75" target="id329"/>
<Sequence id="id160" Classifier="id332" source="id77" target="id78"/>
<Sequence id="id161" source="id79" target="id80"/>
<Sequence id="id162" source="id46" target="id47"/>
<Sequence id="id163" source="id323" target="id292"/>
<Sequence id="id164" Classifier="id295" source="id50" target="id296"/>
<Sequence id="id165" Classifier="id299" source="id62" target="id300"/>
<Sequence id="id166" Classifier="id303" source="id64" target="id304"/>
<Sequence id="id167" Classifier="id307" source="id66" target="id308"/>
<Sequence id="id168" Classifier="id311" source="id78" target="id312"/>
<Sequence id="id169" source="id60" target="id61"/>
<Sequence id="id170" source="id76" target="id77"/>
<Sequence id="id171" source="id81" target="id82"/>
<Sequence id="id173" Classifier="id174" source="id172" target="id2"/>
<Sequence id="id177" Classifier="id178" source="id176" target="id4"/>
<Sequence id="id181" Classifier="id182" source="id180" target="id6"/>
<Sequence id="id185" Classifier="id186" source="id184" target="id8"/>
<Sequence id="id189" Classifier="id190" source="id188" target="id10"/>
<Sequence id="id193" Classifier="id194" source="id192" target="id12"/>
<Sequence id="id197" Classifier="id198" source="id196" target="id14"/>
<Sequence id="id201" Classifier="id202" source="id200" target="id16"/>
<Sequence id="id205" Classifier="id206" source="id204" target="id18"/>
<Sequence id="id209" Classifier="id210" source="id208" target="id20"/>
<Sequence id="id213" Classifier="id214" source="id212" target="id22"/>
<Sequence id="id217" Classifier="id218" source="id216" target="id24"/>
<Sequence id="id221" Classifier="id222" source="id220" target="id26"/>
<Sequence id="id225" Classifier="id226" source="id224" target="id28"/>
<Sequence id="id229" Classifier="id230" source="id228" target="id30"/>
<Sequence id="id233" Classifier="id234" source="id232" target="id32"/>
<Sequence id="id237" Classifier="id238" source="id236" target="id34"/>
<Sequence id="id241" Classifier="id242" source="id240" target="id36"/>
<Sequence id="id245" Classifier="id246" source="id244" target="id38"/>
<Sequence id="id249" Classifier="id250" source="id248" target="id40"/>
<Sequence id="id253" Classifier="id254" source="id252" target="id52"/>
<Sequence id="id257" Classifier="id258" source="id256" target="id54"/>
<Sequence id="id261" Classifier="id262" source="id260" target="id56"/>
<Sequence id="id265" Classifier="id266" source="id264" target="id58"/>
<Sequence id="id269" Classifier="id270" source="id268" target="id68"/>
<Sequence id="id273" Classifier="id274" source="id272" target="id70"/>
<Sequence id="id277" Classifier="id278" source="id276" target="id72"/>
<Sequence id="id281" Classifier="id282" source="id280" target="id74"/>
<Sequence id="id285" Classifier="id286" source="id284" target="id42"/>
<Sequence id="id289" Classifier="id290" source="id288" target="id44"/>
<Sequence id="id293" Classifier="id294" source="id292" target="id49"/>
<Sequence id="id297" Classifier="id298" source="id296" target="id51"/>
<Sequence id="id301" Classifier="id302" source="id300" target="id63"/>

```
  <Sequence id="id305" Classifier="id306" source="id304" target="id65"/>
  <Sequence id="id309" Classifier="id310" source="id308" target="id67"/>
  <Sequence id="id313" Classifier="id314" source="id312" target="id79"/>
  <Sequence id="id317" Classifier="id318" source="id316" target="id46"/>
  <Sequence id="id321" Classifier="id322" source="id320" target="id48"/>
  <Sequence id="id324" source="id48" target="id323"/>
  <Sequence id="id326" Classifier="id327" source="id325" target="id60"/>
  <Sequence id="id330" Classifier="id331" source="id329" target="id76"/>
  <Sequence id="id334" Classifier="id335" source="id333" target="id81"/>
  <Sequence id="id338" source="id337" target="id172"/>
  <Sequence id="id340" source="id3" target="id339"/>
  <Fork id="id93" Classifier="id319" source="id47" target="id320 id333"/>
  <Classifier xsi:type="CSM:MessageType" id="id174" name="Book_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id175" name="Book_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id178" name="setName_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id179" name="setName_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id182" name="setAuthor_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id183" name="setAuthor_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id186" name="setPublisher_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id187" name="setPublisher_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id190" name="setPublishingDate_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id191" name="setPublishingDate_END"
kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id194" name="setSubject_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id195" name="setSubject_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id198" name="setDescription_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id199" name="setDescription_END"
kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id202" name="setISBN_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id203" name="setISBN_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id206" name="setPrice_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id207" name="setPrice_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id210" name="setInStock_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id211" name="setInStock_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id214" name="Book_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id215" name="Book_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id218" name="setName_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id219" name="setName_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id222" name="setAuthor_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id223" name="setAuthor_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id226" name="setPublisher_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id227" name="setPublisher_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id230" name="setPublishingDate_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id231" name="setPublishingDate_END"
kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id234" name="setSubject_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id235" name="setSubject_END" kind="reply"/>
```

<Classifier xsi:type="CSM:MessageType" id="id238" name="setDescription_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id239" name="setDescription_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id242" name="setISBN_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id243" name="setISBN_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id246" name="setPrice_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id247" name="setPrice_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id250" name="setInStock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id251" name="setInStock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id254" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id255" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id258" name="addInStock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id259" name="addInStock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id262" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id263" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id266" name="addInStock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id267" name="addInStock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id270" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id271" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id274" name="addInStock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id275" name="addInStock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id278" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id279" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id282" name="addInStock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id283" name="addInStock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id286" name="addProduct_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id287" name="addProduct_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id290" name="addProduct_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id291" name="addProduct_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id294" name="lock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id295" name="lock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id298" name="replenish_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id299" name="replenish_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id302" name="releaseLock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id303" name="releaseLock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id306" name="lock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id307" name="lock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id310" name="replenish_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id311" name="replenish_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id314" name="releaseLock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id315" name="releaseLock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id318" name="Replenish_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id319" name="Replenish_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id322" name="start_START" kind="async"/>
<Classifier xsi:type="CSM:MessageType" id="id327" name="Notify_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id328" name="Notify_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id331" name="Notify_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id332" name="Notify_END" kind="reply"/>

```
  <Classifier xsi:type="CSM:MessageType" id="id335" name="ShoppingCart_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id336" name="ShoppingCart_END" kind="reply"/>
 </Scenario>
 <Component id="id83" name="Server"/>
 <Component id="id84" name="Book1"/>
 <Component id="id85" name="Book2"/>
 <Component id="id86" name="Inventory"/>
 <Component id="id87" name="Replenish"/>
 <Component id="id88" name="Replenish1"/>
 <Component id="id89" name="Billing"/>
 <Component id="id90" name="ShoppingCart1"/>
</CSM:CSMType>
```

## *Appendix B10 Use Case: Create Account, Login and Add to Cart*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id45" hostDemand="25" name="browse_STARTC"
predecessor="id94" successor="id56"/>
  <Step id="id10" Component="id45" hostDemand="52" name="createAccount_STARTC"
predecessor="id98" successor="id58"/>
  <Step id="id18" Component="id45" hostDemand="39" name="login_STARTC"
predecessor="id102" successor="id62"/>
  <Step id="id32" Component="id45" hostDemand="27" name="displayProductInfo_STARTC"
predecessor="id106" successor="id65"/>
  <Step id="id38" Component="id45" hostDemand="30" name="addProduct_STARTC"
predecessor="id110" successor="id67"/>
  <Step id="id3" Component="id46" hostDemand="28" name="display_START"
predecessor="id114" successor="id70"/>
  <Step id="id11" Component="id48" hostDemand="11" name="exists_START"
predecessor="id138" successor="id79"/>
  <Step id="id13" Component="id49" hostDemand="12" name="CustomerAccount_START"
predecessor="id154" successor="id87"/>
  <Step id="id15" Component="id48" hostDemand="10" name="addCustomer_START"
predecessor="id142" successor="id80"/>
  <Step id="id19" Component="id48" hostDemand="27" name="exists_START"
predecessor="id146" successor="id81"/>
  <Step id="id25" Component="id48" hostDemand="35" name="getAccount_START"
predecessor="id150" successor="id84"/>
  <Step id="id33" Component="id46" hostDemand="23" name="displayProduct_START"
predecessor="id118" successor="id73"/>
  <Step id="id39" Component="id46" hostDemand="2" name="getProduct_START"
predecessor="id122" successor="id75"/>
  <Step id="id41" Component="id50" hostDemand="15" name="addProduct_START"
predecessor="id174" successor="id92"/>
  <Step id="id4" Component="id47" hostDemand="15" name="getName_START"
predecessor="id126" successor="id76"/>
  <Step id="id6" Component="id47" hostDemand="15" name="getName_START"
predecessor="id130" successor="id77"/>
```

<Step id="id34" Component="id47" hostDemand="7" name="display_START" predecessor="id134" successor="id78"/>
<Step id="id20" Component="id49" hostDemand="10" name="getLogin_START" predecessor="id158" successor="id88"/>
<Step id="id22" Component="id49" hostDemand="14" name="getPassword_START" predecessor="id162" successor="id89"/>
<Step id="id26" Component="id49" hostDemand="17" name="getLogin_START" predecessor="id166" successor="id90"/>
<Step id="id28" Component="id49" hostDemand="12" name="getPassword_START" predecessor="id170" successor="id91"/>
<ResourceAcquire id="id93" predecessor="id178" successor="id94" acquire="id45"/>
<ResourceAcquire id="id97" predecessor="id52" successor="id98" acquire="id45"/>
<ResourceAcquire id="id101" predecessor="id53" successor="id102" acquire="id45"/>
<ResourceAcquire id="id105" predecessor="id54" successor="id106" acquire="id45"/>
<ResourceAcquire id="id109" predecessor="id55" successor="id110" acquire="id45"/>
<ResourceAcquire id="id113" predecessor="id56" successor="id114" acquire="id46"/>
<ResourceAcquire id="id117" predecessor="id65" successor="id118" acquire="id46"/>
<ResourceAcquire id="id121" predecessor="id67" successor="id122" acquire="id46"/>
<ResourceAcquire id="id125" predecessor="id70" successor="id126" acquire="id47"/>
<ResourceAcquire id="id129" predecessor="id71" successor="id130" acquire="id47"/>
<ResourceAcquire id="id133" predecessor="id73" successor="id134" acquire="id47"/>
<ResourceAcquire id="id137" predecessor="id58" successor="id138" acquire="id48"/>
<ResourceAcquire id="id141" predecessor="id60" successor="id142" acquire="id48"/>
<ResourceAcquire id="id145" predecessor="id62" successor="id146" acquire="id48"/>
<ResourceAcquire id="id149" predecessor="id63" successor="id150" acquire="id48"/>
<ResourceAcquire id="id153" predecessor="id59" successor="id154" acquire="id49"/>
<ResourceAcquire id="id157" predecessor="id81" successor="id158" acquire="id49"/>
<ResourceAcquire id="id161" predecessor="id82" successor="id162" acquire="id49"/>
<ResourceAcquire id="id165" predecessor="id84" successor="id166" acquire="id49"/>
<ResourceAcquire id="id169" predecessor="id85" successor="id170" acquire="id49"/>
<ResourceAcquire id="id173" predecessor="id68" successor="id174" acquire="id50"/>
<ResourceAcquire id="id177" predecessor="id1" successor="id178" acquire="id44"/>
<ResourceRelease id="id9" predecessor="id57" successor="id180" release="id45"/>
<ResourceRelease id="id17" predecessor="id61" successor="id53" release="id45"/>
<ResourceRelease id="id31" predecessor="id64" successor="id54" release="id45"/>
<ResourceRelease id="id37" predecessor="id66" successor="id55" release="id45"/>
<ResourceRelease id="id43" predecessor="id69" successor="id51" release="id45"/>
<ResourceRelease id="id8" predecessor="id72" successor="id57" release="id46"/>
<ResourceRelease id="id36" predecessor="id74" successor="id66" release="id46"/>
<ResourceRelease id="id40" predecessor="id75" successor="id68" release="id46"/>
<ResourceRelease id="id5" predecessor="id76" successor="id71" release="id47"/>
<ResourceRelease id="id7" predecessor="id77" successor="id72" release="id47"/>
<ResourceRelease id="id35" predecessor="id78" successor="id74" release="id47"/>
<ResourceRelease id="id12" predecessor="id79" successor="id59" release="id48"/>
<ResourceRelease id="id16" predecessor="id80" successor="id61" release="id48"/>
<ResourceRelease id="id24" predecessor="id83" successor="id63" release="id48"/>
<ResourceRelease id="id30" predecessor="id86" successor="id64" release="id48"/>
<ResourceRelease id="id14" predecessor="id87" successor="id60" release="id49"/>
<ResourceRelease id="id21" predecessor="id88" successor="id82" release="id49"/>
<ResourceRelease id="id23" predecessor="id89" successor="id83" release="id49"/>
<ResourceRelease id="id27" predecessor="id90" successor="id85" release="id49"/>
<ResourceRelease id="id29" predecessor="id91" successor="id86" release="id49"/>

<ResourceRelease id="id42" predecessor="id92" successor="id69" release="id50"/>
<ResourceRelease id="id179" predecessor="id180" successor="id52" release="id44"/>
<Start id="id1" target="id177">
 <ClosedWorkload id="id181"/>
</Start>
<End id="id51" Classifier="id112" source="id43"/>
<Sequence id="id52" Classifier="id96" source="id179" target="id97"/>
<Sequence id="id53" Classifier="id100" source="id17" target="id101"/>
<Sequence id="id54" Classifier="id104" source="id31" target="id105"/>
<Sequence id="id55" Classifier="id108" source="id37" target="id109"/>
<Sequence id="id56" source="id2" target="id113"/>
<Sequence id="id57" Classifier="id116" source="id8" target="id9"/>
<Sequence id="id58" source="id10" target="id137"/>
<Sequence id="id59" Classifier="id140" source="id12" target="id153"/>
<Sequence id="id60" Classifier="id156" source="id14" target="id141"/>
<Sequence id="id61" Classifier="id144" source="id16" target="id17"/>
<Sequence id="id62" source="id18" target="id145"/>
<Sequence id="id63" Classifier="id148" source="id24" target="id149"/>
<Sequence id="id64" Classifier="id152" source="id30" target="id31"/>
<Sequence id="id65" source="id32" target="id117"/>
<Sequence id="id66" Classifier="id120" source="id36" target="id37"/>
<Sequence id="id67" source="id38" target="id121"/>
<Sequence id="id68" Classifier="id124" source="id40" target="id173"/>
<Sequence id="id69" Classifier="id176" source="id42" target="id43"/>
<Sequence id="id70" source="id3" target="id125"/>
<Sequence id="id71" Classifier="id128" source="id5" target="id129"/>
<Sequence id="id72" Classifier="id132" source="id7" target="id8"/>
<Sequence id="id73" source="id33" target="id133"/>
<Sequence id="id74" Classifier="id136" source="id35" target="id36"/>
<Sequence id="id75" source="id39" target="id40"/>
<Sequence id="id76" source="id4" target="id5"/>
<Sequence id="id77" source="id6" target="id7"/>
<Sequence id="id78" source="id34" target="id35"/>
<Sequence id="id79" source="id11" target="id12"/>
<Sequence id="id80" source="id15" target="id16"/>
<Sequence id="id81" source="id19" target="id157"/>
<Sequence id="id82" Classifier="id160" source="id21" target="id161"/>
<Sequence id="id83" Classifier="id164" source="id23" target="id24"/>
<Sequence id="id84" source="id25" target="id165"/>
<Sequence id="id85" Classifier="id168" source="id27" target="id169"/>
<Sequence id="id86" Classifier="id172" source="id29" target="id30"/>
<Sequence id="id87" source="id13" target="id14"/>
<Sequence id="id88" source="id20" target="id21"/>
<Sequence id="id89" source="id22" target="id23"/>
<Sequence id="id90" source="id26" target="id27"/>
<Sequence id="id91" source="id28" target="id29"/>
<Sequence id="id92" source="id41" target="id42"/>
<Sequence id="id94" Classifier="id95" source="id93" target="id2"/>
<Sequence id="id98" Classifier="id99" source="id97" target="id10"/>
<Sequence id="id102" Classifier="id103" source="id101" target="id18"/>
<Sequence id="id106" Classifier="id107" source="id105" target="id32"/>
<Sequence id="id110" Classifier="id111" source="id109" target="id38"/>

<Sequence id="id114" Classifier="id115" source="id113" target="id3"/>
<Sequence id="id118" Classifier="id119" source="id117" target="id33"/>
<Sequence id="id122" Classifier="id123" source="id121" target="id39"/>
<Sequence id="id126" Classifier="id127" source="id125" target="id4"/>
<Sequence id="id130" Classifier="id131" source="id129" target="id6"/>
<Sequence id="id134" Classifier="id135" source="id133" target="id34"/>
<Sequence id="id138" Classifier="id139" source="id137" target="id11"/>
<Sequence id="id142" Classifier="id143" source="id141" target="id15"/>
<Sequence id="id146" Classifier="id147" source="id145" target="id19"/>
<Sequence id="id150" Classifier="id151" source="id149" target="id25"/>
<Sequence id="id154" Classifier="id155" source="id153" target="id13"/>
<Sequence id="id158" Classifier="id159" source="id157" target="id20"/>
<Sequence id="id162" Classifier="id163" source="id161" target="id22"/>
<Sequence id="id166" Classifier="id167" source="id165" target="id26"/>
<Sequence id="id170" Classifier="id171" source="id169" target="id28"/>
<Sequence id="id174" Classifier="id175" source="id173" target="id41"/>
<Sequence id="id178" source="id177" target="id93"/>
<Sequence id="id180" source="id9" target="id179"/>
<Classifier xsi:type="CSM:MessageType" id="id95" name="browse_STARTC" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id96" name="browse_ENDC" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id99" name="createAccount_STARTC" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id100" name="createAccount_ENDC" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id103" name="login_STARTC" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id104" name="login_ENDC" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id107" name="displayProductInfo_STARTC" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id108" name="displayProductInfo_ENDC" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id111" name="addProduct_STARTC" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id112" name="addProduct_ENDC" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id115" name="display_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id116" name="display_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id119" name="displayProduct_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id120" name="displayProduct_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id123" name="getProduct_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id124" name="getProduct_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id127" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id128" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id131" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id132" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id135" name="display_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id136" name="display_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id139" name="exists_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id140" name="exists_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id143" name="addCustomer_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id144" name="addCustomer_END" kind="reply"/>

```
  <Classifier xsi:type="CSM:MessageType" id="id147" name="exists_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id148" name="exists_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id151" name="getAccount_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id152" name="getAccount_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id155" name="CustomerAccount_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id156" name="CustomerAccount_END"
kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id159" name="getLogin_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id160" name="getLogin_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id163" name="getPassword_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id164" name="getPassword_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id167" name="getLogin_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id168" name="getLogin_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id171" name="getPassword_START"
kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id172" name="getPassword_END" kind="reply"/>
  <Classifier xsi:type="CSM:MessageType" id="id175" name="addProduct_START" kind="sync"/>
  <Classifier xsi:type="CSM:MessageType" id="id176" name="addProduct_END" kind="reply"/>
 </Scenario>
 <Component id="id44" name="Client"/>
 <Component id="id45" name="Server"/>
 <Component id="id46" name="Inventory"/>
 <Component id="id47" name="Book2"/>
 <Component id="id48" name="CustomerAccount"/>
 <Component id="id49" name="CAccount1"/>
 <Component id="id50" name="ShoppingCart1"/>
</CSM:CSMType>
```

## *Appendix B11 Checkout Scenario*

```
<?xml version="1.0" encoding="ASCII"?>
<CSM:CSMType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:CSM="platform:/resource/edu.carleton.sce.puma/CSM.xsd">
 <Scenario id="id0" name="Scenario">
  <Step id="id2" Component="id78" hostDemand="64" name="checkOut_STARTC"
predecessor="id166" successor="id95"/>
  <Step id="id50" Component="id84" hostDemand="8" name="getOrderNumber_START"
predecessor="id251" successor="id139"/>
  <Step id="id21" Component="id83" hostDemand="168" name="start_START"
predecessor="id238" successor="id241"/>
  <Step id="id58" Component="id88" hostDemand="47" name="display_START"
predecessor="id295" successor="id94"/>
  <Step id="id59" Component="id89" hostDemand="12" name="display_START"
predecessor="id307" successor="id158"/>
  <Step id="id3" Component="id79" hostDemand="18" name="isEmpty_START"
predecessor="id170" successor="id101"/>
  <Step id="id5" Component="id79" hostDemand="39" name="displayWithPrice_START"
predecessor="id174" successor="id102"/>
  <Step id="id11" Component="id79" hostDemand="30" name="getTotal_START"
predecessor="id178" successor="id105"/>
```

<Step id="id15" Component="id81" hostDemand="12" name="getNewOrderNumber_START" predecessor="id222" successor="id117"/>
<Step id="id17" Component="id81" hostDemand="2" name="Order_START" predecessor="id226" successor="id118"/>
<Step id="id19" Component="id82" hostDemand="18" name="Shipping_START" predecessor="id234" successor="id120"/>
<Step id="id6" Component="id80" hostDemand="7" name="getName_START" predecessor="id186" successor="id108"/>
<Step id="id8" Component="id80" hostDemand="7" name="getPrice_START" predecessor="id190" successor="id109"/>
<Step id="id12" Component="id80" hostDemand="8" name="getPrice_START" predecessor="id194" successor="id110"/>
<Step id="id25" Component="id79" hostDemand="8" name="getList_START" predecessor="id182" successor="id107"/>
<Step id="id33" Component="id80" hostDemand="4" name="getInStock_START" predecessor="id198" successor="id111"/>
<Step id="id35" Component="id80" hostDemand="9" name="setInStock_START" predecessor="id202" successor="id112"/>
<Step id="id39" Component="id80" hostDemand="10" name="getName_START" predecessor="id206" successor="id113"/>
<Step id="id64" Component="id80" hostDemand="19" name="getName_START" predecessor="id210" successor="id114"/>
<Step id="id66" Component="id80" hostDemand="11" name="getPrice_START" predecessor="id214" successor="id115"/>
<Step id="id74" Component="id80" hostDemand="10" name="getPrice_START" predecessor="id218" successor="id116"/>
<Step id="id49" Component="id81" hostDemand="51" name="Order_START" predecessor="id230" successor="id119"/>
<Step id="id23" Component="id84" hostDemand="13" name="getShoppingCart_START" predecessor="id243" successor="id137"/>
<Step id="id27" Component="id85" hostDemand="17" name="ShoppingCart_START" predecessor="id259" successor="id141"/>
<Step id="id29" Component="id86" hostDemand="13" name="ShoppingCart_START" predecessor="id263" successor="id142"/>
<Step id="id31" Component="id87" hostDemand="14" name="lock_START" predecessor="id287" successor="id150"/>
<Step id="id37" Component="id86" hostDemand="8" name="addProduct_START" predecessor="id267" successor="id143"/>
<Step id="id41" Component="id87" hostDemand="13" name="releaseLock_START" predecessor="id291" successor="id151"/>
<Step id="id43" Component="id84" hostDemand="10" name="getClient_START" predecessor="id247" successor="id138"/>
<Step id="id45" Component="id86" hostDemand="6" name="isEmpty_START" predecessor="id271" successor="id144"/>
<Step id="id47" Component="id86" hostDemand="17" name="isEmpty_START" predecessor="id275" successor="id145"/>
<Step id="id55" Component="id89" hostDemand="10" name="addPrevOrder_START" predecessor="id303" successor="id157"/>
<Step id="id57" Component="id90" hostDemand="75" name="Billing_START" predecessor="id315" successor="id160"/>
<Step id="id52" Component="id84" hostDemand="12" name="getClient_START" predecessor="id255" successor="id140"/>

<Step id="id63" Component="id86" hostDemand="32" name="displayWithPrice_START" predecessor="id279" successor="id146"/>
<Step id="id72" Component="id86" hostDemand="11" name="getList_START" predecessor="id283" successor="id149"/>
<Step id="id61" Component="id89" hostDemand="7" name="getCreditCard_START" predecessor="id311" successor="id159"/>
<Step id="id70" Component="id88" hostDemand="13" name="getShoppingCart_START" predecessor="id299" successor="id156"/>
<ResourceAcquire id="id165" predecessor="id319" successor="id166" acquire="id78"/>
<ResourceAcquire id="id169" predecessor="id95" successor="id170" acquire="id79"/>
<ResourceAcquire id="id173" predecessor="id96" successor="id174" acquire="id79"/>
<ResourceAcquire id="id177" predecessor="id97" successor="id178" acquire="id79"/>
<ResourceAcquire id="id181" predecessor="id122" successor="id182" acquire="id79"/>
<ResourceAcquire id="id185" predecessor="id102" successor="id186" acquire="id80"/>
<ResourceAcquire id="id189" predecessor="id103" successor="id190" acquire="id80"/>
<ResourceAcquire id="id193" predecessor="id105" successor="id194" acquire="id80"/>
<ResourceAcquire id="id197" predecessor="id126" successor="id198" acquire="id80"/>
<ResourceAcquire id="id201" predecessor="id127" successor="id202" acquire="id80"/>
<ResourceAcquire id="id205" predecessor="id129" successor="id206" acquire="id80"/>
<ResourceAcquire id="id209" predecessor="id146" successor="id210" acquire="id80"/>
<ResourceAcquire id="id213" predecessor="id147" successor="id214" acquire="id80"/>
<ResourceAcquire id="id217" predecessor="id163" successor="id218" acquire="id80"/>
<ResourceAcquire id="id221" predecessor="id98" successor="id222" acquire="id81"/>
<ResourceAcquire id="id225" predecessor="id99" successor="id226" acquire="id81"/>
<ResourceAcquire id="id229" predecessor="id134" successor="id230" acquire="id81"/>
<ResourceAcquire id="id233" predecessor="id100" successor="id234" acquire="id82"/>
<ResourceAcquire id="id237" predecessor="id93" successor="id238" acquire="id83"/>
<ResourceAcquire id="id242" predecessor="id121" successor="id243" acquire="id84"/>
<ResourceAcquire id="id246" predecessor="id131" successor="id247" acquire="id84"/>
<ResourceAcquire id="id250" predecessor="id1" successor="id251" acquire="id84"/>
<ResourceAcquire id="id254" predecessor="id152" successor="id255" acquire="id84"/>
<ResourceAcquire id="id258" predecessor="id123" successor="id259" acquire="id85"/>
<ResourceAcquire id="id262" predecessor="id124" successor="id263" acquire="id86"/>
<ResourceAcquire id="id266" predecessor="id128" successor="id267" acquire="id86"/>
<ResourceAcquire id="id270" predecessor="id132" successor="id271" acquire="id86"/>
<ResourceAcquire id="id274" predecessor="id133" successor="id275" acquire="id86"/>
<ResourceAcquire id="id278" predecessor="id154" successor="id279" acquire="id86"/>
<ResourceAcquire id="id282" predecessor="id162" successor="id283" acquire="id86"/>
<ResourceAcquire id="id286" predecessor="id125" successor="id287" acquire="id87"/>
<ResourceAcquire id="id290" predecessor="id130" successor="id291" acquire="id87"/>
<ResourceAcquire id="id294" predecessor="id160" successor="id295" acquire="id88"/>
<ResourceAcquire id="id298" predecessor="id161" successor="id299" acquire="id88"/>
<ResourceAcquire id="id302" predecessor="id135" successor="id303" acquire="id89"/>
<ResourceAcquire id="id306" predecessor="id94" successor="id307" acquire="id89"/>
<ResourceAcquire id="id310" predecessor="id153" successor="id311" acquire="id89"/>
<ResourceAcquire id="id314" predecessor="id136" successor="id315" acquire="id90"/>
<ResourceAcquire id="id318" predecessor="id1" successor="id319" acquire="id77"/>
<ResourceRelease id="id22" predecessor="id93" successor="id321" release="id78"/>
<ResourceRelease id="id4" predecessor="id101" successor="id96" release="id79"/>
<ResourceRelease id="id10" predecessor="id104" successor="id97" release="id79"/>
<ResourceRelease id="id14" predecessor="id106" successor="id98" release="id79"/>
<ResourceRelease id="id26" predecessor="id107" successor="id123" release="id79"/>

```
<ResourceRelease id="id7" predecessor="id108" successor="id103" release="id80"/>
<ResourceRelease id="id9" predecessor="id109" successor="id104" release="id80"/>
<ResourceRelease id="id13" predecessor="id110" successor="id106" release="id80"/>
<ResourceRelease id="id34" predecessor="id111" successor="id127" release="id80"/>
<ResourceRelease id="id36" predecessor="id112" successor="id128" release="id80"/>
<ResourceRelease id="id40" predecessor="id113" successor="id130" release="id80"/>
<ResourceRelease id="id65" predecessor="id114" successor="id147" release="id80"/>
<ResourceRelease id="id67" predecessor="id115" successor="id148" release="id80"/>
<ResourceRelease id="id75" predecessor="id116" successor="id164" release="id80"/>
<ResourceRelease id="id16" predecessor="id117" successor="id99" release="id81"/>
<ResourceRelease id="id18" predecessor="id118" successor="id100" release="id81"/>
<ResourceRelease id="id54" predecessor="id119" successor="id135" release="id81"/>
<ResourceRelease id="id20" predecessor="id120" successor="id93" release="id82"/>
<ResourceRelease id="id240" predecessor="id241" successor="id121" release="id83"/>
<ResourceRelease id="id24" predecessor="id137" successor="id122" release="id84"/>
<ResourceRelease id="id44" predecessor="id138" successor="id132" release="id84"/>
<ResourceRelease id="id51" predecessor="id139" successor="id152" release="id84"/>
<ResourceRelease id="id53" predecessor="id140" successor="id94" release="id84"/>
<ResourceRelease id="id28" predecessor="id141" successor="id124" release="id85"/>
<ResourceRelease id="id30" predecessor="id142" successor="id125" release="id86"/>
<ResourceRelease id="id38" predecessor="id143" successor="id129" release="id86"/>
<ResourceRelease id="id46" predecessor="id144" successor="id133" release="id86"/>
<ResourceRelease id="id48" predecessor="id145" successor="id134" release="id86"/>
<ResourceRelease id="id68" predecessor="id148" successor="id155" release="id86"/>
<ResourceRelease id="id73" predecessor="id149" successor="id163" release="id86"/>
<ResourceRelease id="id32" predecessor="id150" successor="id126" release="id87"/>
<ResourceRelease id="id42" predecessor="id151" successor="id131" release="id87"/>
<ResourceRelease id="id69" predecessor="id155" successor="id161" release="id88"/>
<ResourceRelease id="id71" predecessor="id156" successor="id162" release="id88"/>
<ResourceRelease id="id56" predecessor="id157" successor="id136" release="id89"/>
<ResourceRelease id="id60" predecessor="id158" successor="id153" release="id89"/>
<ResourceRelease id="id62" predecessor="id159" successor="id154" release="id89"/>
<ResourceRelease id="id76" predecessor="id164" successor="id92" release="id90"/>
<ResourceRelease id="id320" predecessor="id321" successor="id91" release="id77"/>
<Start id="id1" target="id318 id250">
  <ClosedWorkload id="id322"/>
</Start>
<End id="id91" Classifier="id168" source="id320"/>
<End id="id92" Classifier="id317" source="id76"/>
<Sequence id="id95" source="id2" target="id169"/>
<Sequence id="id96" Classifier="id172" source="id4" target="id173"/>
<Sequence id="id97" Classifier="id176" source="id10" target="id177"/>
<Sequence id="id98" Classifier="id180" source="id14" target="id221"/>
<Sequence id="id99" Classifier="id224" source="id16" target="id225"/>
<Sequence id="id100" Classifier="id228" source="id18" target="id233"/>
<Sequence id="id101" source="id3" target="id4"/>
<Sequence id="id102" source="id5" target="id185"/>
<Sequence id="id103" Classifier="id188" source="id7" target="id189"/>
<Sequence id="id104" Classifier="id192" source="id9" target="id10"/>
<Sequence id="id105" source="id11" target="id193"/>
<Sequence id="id106" Classifier="id196" source="id13" target="id14"/>
<Sequence id="id107" source="id25" target="id26"/>
```

<Sequence id="id108" source="id6" target="id7"/>
<Sequence id="id109" source="id8" target="id9"/>
<Sequence id="id110" source="id12" target="id13"/>
<Sequence id="id111" source="id33" target="id34"/>
<Sequence id="id112" source="id35" target="id36"/>
<Sequence id="id113" source="id39" target="id40"/>
<Sequence id="id114" source="id64" target="id65"/>
<Sequence id="id115" source="id66" target="id67"/>
<Sequence id="id116" source="id74" target="id75"/>
<Sequence id="id117" source="id15" target="id16"/>
<Sequence id="id118" source="id17" target="id18"/>
<Sequence id="id119" source="id49" target="id54"/>
<Sequence id="id120" source="id19" target="id20"/>
<Sequence id="id121" source="id240" target="id242"/>
<Sequence id="id122" Classifier="id245" source="id24" target="id181"/>
<Sequence id="id123" Classifier="id184" source="id26" target="id258"/>
<Sequence id="id124" Classifier="id261" source="id28" target="id262"/>
<Sequence id="id125" Classifier="id265" source="id30" target="id286"/>
<Sequence id="id126" Classifier="id289" source="id32" target="id197"/>
<Sequence id="id127" Classifier="id200" source="id34" target="id201"/>
<Sequence id="id128" Classifier="id204" source="id36" target="id266"/>
<Sequence id="id129" Classifier="id269" source="id38" target="id205"/>
<Sequence id="id130" Classifier="id208" source="id40" target="id290"/>
<Sequence id="id131" Classifier="id293" source="id42" target="id246"/>
<Sequence id="id132" Classifier="id249" source="id44" target="id270"/>
<Sequence id="id133" Classifier="id273" source="id46" target="id274"/>
<Sequence id="id134" Classifier="id277" source="id48" target="id229"/>
<Sequence id="id135" Classifier="id232" source="id54" target="id302"/>
<Sequence id="id136" Classifier="id305" source="id56" target="id314"/>
<Sequence id="id137" source="id23" target="id24"/>
<Sequence id="id138" source="id43" target="id44"/>
<Sequence id="id139" source="id50" target="id51"/>
<Sequence id="id140" source="id52" target="id53"/>
<Sequence id="id141" source="id27" target="id28"/>
<Sequence id="id142" source="id29" target="id30"/>
<Sequence id="id143" source="id37" target="id38"/>
<Sequence id="id144" source="id45" target="id46"/>
<Sequence id="id145" source="id47" target="id48"/>
<Sequence id="id146" source="id63" target="id209"/>
<Sequence id="id147" Classifier="id212" source="id65" target="id213"/>
<Sequence id="id148" Classifier="id216" source="id67" target="id68"/>
<Sequence id="id149" source="id72" target="id73"/>
<Sequence id="id150" source="id31" target="id32"/>
<Sequence id="id151" source="id41" target="id42"/>
<Sequence id="id152" Classifier="id253" source="id51" target="id254"/>
<Sequence id="id153" Classifier="id309" source="id60" target="id310"/>
<Sequence id="id154" Classifier="id313" source="id62" target="id278"/>
<Sequence id="id155" Classifier="id281" source="id68" target="id69"/>
<Sequence id="id156" source="id70" target="id71"/>
<Sequence id="id157" source="id55" target="id56"/>
<Sequence id="id158" source="id59" target="id60"/>
<Sequence id="id159" source="id61" target="id62"/>

<Sequence id="id160" source="id57" target="id294"/>
<Sequence id="id161" Classifier="id297" source="id69" target="id298"/>
<Sequence id="id162" Classifier="id301" source="id71" target="id282"/>
<Sequence id="id163" Classifier="id285" source="id73" target="id217"/>
<Sequence id="id164" Classifier="id220" source="id75" target="id76"/>
<Sequence id="id166" Classifier="id167" source="id165" target="id2"/>
<Sequence id="id170" Classifier="id171" source="id169" target="id3"/>
<Sequence id="id174" Classifier="id175" source="id173" target="id5"/>
<Sequence id="id178" Classifier="id179" source="id177" target="id11"/>
<Sequence id="id182" Classifier="id183" source="id181" target="id25"/>
<Sequence id="id186" Classifier="id187" source="id185" target="id6"/>
<Sequence id="id190" Classifier="id191" source="id189" target="id8"/>
<Sequence id="id194" Classifier="id195" source="id193" target="id12"/>
<Sequence id="id198" Classifier="id199" source="id197" target="id33"/>
<Sequence id="id202" Classifier="id203" source="id201" target="id35"/>
<Sequence id="id206" Classifier="id207" source="id205" target="id39"/>
<Sequence id="id210" Classifier="id211" source="id209" target="id64"/>
<Sequence id="id214" Classifier="id215" source="id213" target="id66"/>
<Sequence id="id218" Classifier="id219" source="id217" target="id74"/>
<Sequence id="id222" Classifier="id223" source="id221" target="id15"/>
<Sequence id="id226" Classifier="id227" source="id225" target="id17"/>
<Sequence id="id230" Classifier="id231" source="id229" target="id49"/>
<Sequence id="id234" Classifier="id235" source="id233" target="id19"/>
<Sequence id="id238" Classifier="id239" source="id237" target="id21"/>
<Sequence id="id241" source="id21" target="id240"/>
<Sequence id="id243" Classifier="id244" source="id242" target="id23"/>
<Sequence id="id247" Classifier="id248" source="id246" target="id43"/>
<Sequence id="id251" Classifier="id252" source="id250" target="id50"/>
<Sequence id="id255" Classifier="id256" source="id254" target="id52"/>
<Sequence id="id259" Classifier="id260" source="id258" target="id27"/>
<Sequence id="id263" Classifier="id264" source="id262" target="id29"/>
<Sequence id="id267" Classifier="id268" source="id266" target="id37"/>
<Sequence id="id271" Classifier="id272" source="id270" target="id45"/>
<Sequence id="id275" Classifier="id276" source="id274" target="id47"/>
<Sequence id="id279" Classifier="id280" source="id278" target="id63"/>
<Sequence id="id283" Classifier="id284" source="id282" target="id72"/>
<Sequence id="id287" Classifier="id288" source="id286" target="id31"/>
<Sequence id="id291" Classifier="id292" source="id290" target="id41"/>
<Sequence id="id295" Classifier="id296" source="id294" target="id58"/>
<Sequence id="id299" Classifier="id300" source="id298" target="id70"/>
<Sequence id="id303" Classifier="id304" source="id302" target="id55"/>
<Sequence id="id307" Classifier="id308" source="id306" target="id59"/>
<Sequence id="id311" Classifier="id312" source="id310" target="id61"/>
<Sequence id="id315" Classifier="id316" source="id314" target="id57"/>
<Sequence id="id319" source="id318" target="id165"/>
<Sequence id="id321" source="id22" target="id320"/>
<Fork id="id93" Classifier="id236" source="id20" target="id237 id22"/>
<Join id="id94" Classifier="id257" source="id53 id58" target="id306"/>
<Classifier xsi:type="CSM:MessageType" id="id167" name="checkOut_STARTC" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id168" name="checkOut_ENDC" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id171" name="isEmpty_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id172" name="isEmpty_END" kind="reply"/>

<Classifier xsi:type="CSM:MessageType" id="id175" name="displayWithPrice_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id176" name="displayWithPrice_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id179" name="getTotal_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id180" name="getTotal_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id183" name="getList_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id184" name="getList_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id187" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id188" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id191" name="getPrice_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id192" name="getPrice_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id195" name="getPrice_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id196" name="getPrice_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id199" name="getInStock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id200" name="getInStock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id203" name="setInStock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id204" name="setInStock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id207" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id208" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id211" name="getName_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id212" name="getName_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id215" name="getPrice_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id216" name="getPrice_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id219" name="getPrice_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id220" name="getPrice_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id223" name="getNewOrderNumber_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id224" name="getNewOrderNumber_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id227" name="Order_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id228" name="Order_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id231" name="Order_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id232" name="Order_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id235" name="Shipping_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id236" name="Shipping_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id239" name="start_START" kind="async"/>
<Classifier xsi:type="CSM:MessageType" id="id244" name="getShoppingCart_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id245" name="getShoppingCart_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id248" name="getClient_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id249" name="getClient_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id252" name="getOrderNumber_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id253" name="getOrderNumber_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id256" name="getClient_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id257" name="getClient_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id260" name="ShoppingCart_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id261" name="ShoppingCart_END" kind="reply"/>

<Classifier xsi:type="CSM:MessageType" id="id264" name="ShoppingCart_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id265" name="ShoppingCart_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id268" name="addProduct_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id269" name="addProduct_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id272" name="isEmpty_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id273" name="isEmpty_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id276" name="isEmpty_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id277" name="isEmpty_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id280" name="displayWithPrice_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id281" name="displayWithPrice_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id284" name="getList_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id285" name="getList_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id288" name="lock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id289" name="lock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id292" name="releaseLock_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id293" name="releaseLock_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id296" name="display_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id297" name="display_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id300" name="getShoppingCart_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id301" name="getShoppingCart_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id304" name="addPrevOrder_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id305" name="addPrevOrder_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id308" name="display_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id309" name="display_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id312" name="getCreditCard_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id313" name="getCreditCard_END" kind="reply"/>
<Classifier xsi:type="CSM:MessageType" id="id316" name="Billing_START" kind="sync"/>
<Classifier xsi:type="CSM:MessageType" id="id317" name="Billing_END" kind="reply"/>
</Scenario>
<Component id="id77" name="Client"/>
<Component id="id78" name="Server"/>
<Component id="id79" name="ShoppingCart1"/>
<Component id="id80" name="Book2"/>
<Component id="id81" name="Order"/>
<Component id="id82" name="Shipping"/>
<Component id="id83" name="Shipping1"/>
<Component id="id84" name="radsbookstore.Order@17ce4e7"/>
<Component id="id85" name="ShoppingCart2"/>
<Component id="id86" name="ShoppingCart3"/>
<Component id="id87" name="Inventory"/>
<Component id="id88" name="radsbookstore.Order@982589"/>
<Component id="id89" name="CAccount1"/>
<Component id="id90" name="radsbookstore.Billing@c88440"/>

</CSM:CSMType>

# Appendix C LQN File

## *Appendix C1Synchronous Communication Pattern*

```
G
"CSM2LQN output for file: C:/Documents and
Settings/aaronguo/Desktop/thesisCodesDeployment/thesisFinalCodes/Output/output.csm"
0.0
0
0
0.9
-1

P 0
p InfiniteProc f i
-1

T 0
t Client_id6 f Client_id6_E1  -1 InfiniteProc
t Inventory_id8 f Inventory_id8_E1  -1 InfiniteProc
t RefTask1 r RefTask1_E1  -1 InfiniteProc
t Server_id7 f Server_id7_E1  -1 InfiniteProc
-1

E 0
-1

A Client_id6
s Client_id6_A1 1.0
s Client_id6_A2 1.0
y Client_id6_A2 Server_id7_E1 1.0
s Client_id6_A3 1.0
:
;
Client_id6_A1 -> Client_id6_A2;
Client_id6_A2 -> Client_id6_A3
-1

A Inventory_id8
s Inventory_id8_A1 1.0
s Inventory_id8_A3 1.0
s display_START_id3 1.0
:
;
Inventory_id8_A1 -> display_START_id3;
display_START_id3 -> Inventory_id8_A3;
Inventory_id8_A3[ ]
-1

A RefTask1
```

```
s RefTask1_A1 1.0
z RefTask1_A1 Client_id6_E1 1.0
-1

A Server_id7
s Server_id7_A1 1.0
s Server_id7_A3 1.0
y Server_id7_A3 Inventory_id8_E1 1.0
s Server_id7_A4 1.0
s browse_STARTC_id2 1.0
:
;
Server_id7_A1 -> browse_STARTC_id2;
browse_STARTC_id2 -> Server_id7_A3;
Server_id7_A3 -> Server_id7_A4;
Server_id7_A4[ ]
-1
```

## *Appendix C2Asynchronous Communication Pattern*

```
G
"CSM2LQN output for file: C:/Documents and
Settings/aaronguo/Desktop/thesisCodesDeployment/thesisFinalCodes/Output/4.2 asynchronous
communication.csm"
0.0
0
0
0.9
-1

P 0
p InfiniteProc f i
-1

T 0
t Client_id4 f Client_id4_E1  -1 InfiniteProc
t Inventory_id6 f Inventory_id6_E1  -1 InfiniteProc
t RefTask3 r RefTask3_E1  -1 InfiniteProc
t Server_id5 f Server_id5_E1  -1 InfiniteProc
-1

E 0
-1

A Client_id4
s Client_id4_A1 1.0
s Client_id4_A2 1.0
z Client_id4_A2 Server_id5_E1 1.0
:
;
Client_id4_A1 -> Client_id4_A2
-1
```

```
A Inventory_id6
s Inventory_id6_A1 1.0
s Inventory_id6_A3 1.0
s display_START_id3 1.0
:
;
Inventory_id6_A1 -> display_START_id3;
display_START_id3 -> Inventory_id6_A3
-1

A RefTask3
s RefTask3_A1 1.0
z RefTask3_A1 Client_id4_E1 1.0
-1

A Server_id5
s Server_id5_A1 1.0
s Server_id5_A3 1.0
z Server_id5_A3 Inventory_id6_E1 1.0
s browse_STARTC_id2 1.0
:
;
Server_id5_A1 -> browse_STARTC_id2;
browse_STARTC_id2 -> Server_id5_A3
-1
```

## *Appendix C3 Forwarding Communication Pattern*

```
G
"CSM2LQN output for file: C:/Documents and
Settings/aaronguo/Desktop/thesisCodesDeployment/thesisFinalCodes/Output/4.3 forwarding
communication.csm"
0.0
0
0
0.9
-1

P 0
p InfiniteProc f i
-1

T 0
t Client_id5 f Client_id5_E1  -1 InfiniteProc
t Inventory_id7 f Inventory_id7_E1  -1 InfiniteProc
t RefTask4 r RefTask4_E1  -1 InfiniteProc
t Server_id6 f Server_id6_E1  -1 InfiniteProc
-1

E 0
-1
```

```
A Client_id5
s Client_id5_A1 1.0
s Client_id5_A2 1.0
y Client_id5_A2 Server_id6_E1 1.0
s Client_id5_A4 1.0
s getName_START_id4 1.0
:
;
Client_id5_A1 -> Client_id5_A2;
Client_id5_A2 -> getName_START_id4;
getName_START_id4 -> Client_id5_A4
-1

A Inventory_id7
s Inventory_id7_A1 1.0
s Inventory_id7_A3 1.0
s display_START_id3 1.0
:
;
Inventory_id7_A1 -> display_START_id3;
display_START_id3 -> Inventory_id7_A3;
Inventory_id7_A3[ ]
-1

A RefTask4
s RefTask4_A1 1.0
z RefTask4_A1 Client_id5_E1 1.0
-1

A Server_id6
s Server_id6_A1 1.0
s Server_id6_A3 1.0
y Server_id6_A3 Inventory_id7_E1 1.0
s browse_STARTC_id2 1.0
:
;
Server_id6_A1 -> browse_STARTC_id2;
browse_STARTC_id2 -> Server_id6_A3;
Server_id6_A3[ ]
-1
```

## *Appendix C4 Synchronous pattern with nested interaction*

```
G
"CSM2LQN output for file: C:/Documents and
Settings/aaronguo/Desktop/thesisCodesDeployment/thesisFinalCodes/Output/4.4 synchronous pattern
with nested interaction.csm"
0.0
0
0
0.9
-1
```

```
P 0
p InfiniteProc f i
-1

T 0
t Book1_id11 f Book1_id11_E1  -1 InfiniteProc
t Client_id8 f Client_id8_E1  -1 InfiniteProc
t Inventory_id10 f Inventory_id10_E1  -1 InfiniteProc
t RefTask5 r RefTask5_E1  -1 InfiniteProc
t Server_id9 f Server_id9_E1  -1 InfiniteProc
-1

E 0
-1

A Book1_id11
s Book1_id11_A1 1.0
s Book1_id11_A3 1.0
s getName_START_id4 1.0
:
;
Book1_id11_A1 -> getName_START_id4;
getName_START_id4 -> Book1_id11_A3;
Book1_id11_A3[ ]
-1

A Client_id8
s Client_id8_A1 1.0
s Client_id8_A2 1.0
y Client_id8_A2 Server_id9_E1 1.0
s Client_id8_A3 1.0
:
;
Client_id8_A1 -> Client_id8_A2;
Client_id8_A2 -> Client_id8_A3
-1

A Inventory_id10
s Inventory_id10_A1 1.0
s Inventory_id10_A3 1.0
y Inventory_id10_A3 Book1_id11_E1 1.0
s Inventory_id10_A4 1.0
s display_START_id3 1.0
:
;
Inventory_id10_A1 -> display_START_id3;
display_START_id3 -> Inventory_id10_A3;
Inventory_id10_A3 -> Inventory_id10_A4;
Inventory_id10_A4[ ]
-1
```

```
A RefTask5
s RefTask5_A1 1.0
z RefTask5_A1 Client_id8_E1 1.0
-1

A Server_id9
s Server_id9_A1 1.0
s Server_id9_A3 1.0
y Server_id9_A3 Inventory_id10_E1 1.0
s Server_id9_A4 1.0
s browse_STARTC_id2 1.0
:
;
Server_id9_A1 -> browse_STARTC_id2;
browse_STARTC_id2 -> Server_id9_A3;
Server_id9_A3 -> Server_id9_A4;
Server_id9_A4[ ]
-1
```

## *Appendix C5 Synchronous Communication Pattern with Fork*

```
G
"CSM2LQN output for file: C:/Documents and
Settings/aaronguo/Desktop/thesisCodesDeployment/thesisFinalCodes/Output/4.5 synchronous
communication with fork.csm"
0.0
0
0
0.9
-1

P 0
p InfiniteProc f i
-1

T 0
t Book_id10 f Book_id10_E1  -1 InfiniteProc
t Client_id7 f Client_id7_E1  -1 InfiniteProc
t Inventory_id9 f Inventory_id9_E1  -1 InfiniteProc
t RefTask6 r RefTask6_E1  -1 InfiniteProc
t Server_id8 f Server_id8_E1  -1 InfiniteProc
-1

E 0
-1

A Book_id10
s Book_id10_A1 1.0
s Book_id10_A3 1.0
s displayBook_STARTC_id4 1.0
:
;
```

```
Book_id10_A1 -> displayBook_STARTC_id4;
displayBook_STARTC_id4 -> Book_id10_A3;
Book_id10_A3[ ]
-1

A Client_id7
s Client_id7_A1 1.0
s Client_id7_A2 1.0
y Client_id7_A2 Server_id8_E1 1.0
s Client_id7_A3 1.0
:
;
Client_id7_A1 -> Client_id7_A2;
Client_id7_A2 -> Client_id7_A3
-1

A Inventory_id9
s Inventory_id9_A1 1.0
s Inventory_id9_A3 1.0
s display_START_id3 1.0
:
;
Inventory_id9_A1 -> display_START_id3;
display_START_id3 -> Inventory_id9_A3
-1

A RefTask6
s RefTask6_A1 1.0
z RefTask6_A1 Client_id7_E1 1.0
-1

A Server_id8
s Server_id8_A1 1.0
s Server_id8_A4 1.0
z Server_id8_A4 Inventory_id9_E1 1.0
s Server_id8_A6 1.0
y Server_id8_A6 Book_id10_E1 1.0
s Server_id8_A7 1.0
s Server_id8_ANDbr3 1.0
s Server_id8_ANDbr5 1.0
s browse_STARTC_id2 1.0
:
;
Server_id8_A1 -> browse_STARTC_id2;
Server_id8_ANDbr3 -> Server_id8_A4;
Server_id8_ANDbr5 -> Server_id8_A6;
Server_id8_A6 -> Server_id8_A7;
Server_id8_A7[ ]
-1
```

## *Appendix C6 Nested Synchronous Communication Pattern with Fork and Join*

```
G
"CSM2LQN output for file: C:/Documents and
Settings/aaronguo/Desktop/thesisCodesDeployment/thesisFinalCodes/Output/4.6 synchronous
communication with fork and join.csm"
0.0
0
0
0.9
-1

P 0
p InfiniteProc f i
-1

T 0
t Book_id11 f Book_id11_E1  -1 InfiniteProc
t Client_id8 f Client_id8_E1  -1 InfiniteProc
t Inventory_id10 f Inventory_id10_E1  -1 InfiniteProc
t RefTask7 r RefTask7_E1  -1 InfiniteProc
t Server_id9 f Server_id9_E1  -1 InfiniteProc
-1

E 0
-1

A Book_id11
s Book_id11_A1 1.0
s Book_id11_A3 1.0
s displayBook_STARTC_id4 1.0
:
;
Book_id11_A1 -> displayBook_STARTC_id4;
displayBook_STARTC_id4 -> Book_id11_A3
-1

A Client_id8
s Client_id8_A1 1.0
s Client_id8_A2 1.0
y Client_id8_A2 Server_id9_E1 1.0
:
;
Client_id8_A1 -> Client_id8_A2
-1

A Inventory_id10
s Inventory_id10_A1 1.0
s Inventory_id10_A3 1.0
s display_START_id3 1.0
:
```

```
;
Inventory_id10_A1 -> display_START_id3;
display_START_id3 -> Inventory_id10_A3
-1

A RefTask7
s RefTask7_A1 1.0
y RefTask7_A1 Client_id8_E1 1.0
-1

A Server_id9
s Server_id9_A1 1.0
s Server_id9_A4 1.0
y Server_id9_A4 Inventory_id10_E1 1.0
s Server_id9_A6 1.0
y Server_id9_A6 Book_id11_E1 1.0
s Server_id9_ANDbr3 1.0
s Server_id9_ANDbr5 1.0
s browse_STARTC_id2 1.0
:
;
Server_id9_A1 -> browse_STARTC_id2;
Server_id9_ANDbr3 -> Server_id9_A4;
Server_id9_ANDbr5 -> Server_id9_A6
-1
```

## *Appendix C7 Asynchronous Communication Pattern with Fork*

```
G
"CSM2LQN output for file: C:/Documents and
Settings/aaronguo/Desktop/thesisCodesDeployment/thesisFinalCodes/Output/4.7 asynchronous
Communication Pattern with Fork.csm"
0.0
0
0
0.9
-1

P 0
p InfiniteProc f i
-1

T 0
t Book_id8 f Book_id8_E1  -1 InfiniteProc
t Client_id5 f Client_id5_E1  -1 InfiniteProc
t Inventory_id7 f Inventory_id7_E1  -1 InfiniteProc
t RefTask8 r RefTask8_E1  -1 InfiniteProc
t Server_id6 f Server_id6_E1  -1 InfiniteProc
-1

E 0
-1
```

```
A Book_id8
s Book_id8_A1 1.0
s Book_id8_A3 1.0
s displayBook_START_id4 1.0
:
;
Book_id8_A1 -> displayBook_START_id4;
displayBook_START_id4 -> Book_id8_A3
-1

A Client_id5
s Client_id5_A1 1.0
s Client_id5_A2 1.0
z Client_id5_A2 Server_id6_E1 1.0
s Client_id5_ANDbr3 1.0
s Client_id5_ANDbr4 1.0
:
;
Client_id5_A1 -> Client_id5_A2
-1

A Inventory_id7
s Inventory_id7_A1 1.0
s Inventory_id7_A3 1.0
s Inventory_id7_A4 1.0
z Inventory_id7_A4 Book_id8_E1 1.0
s display_START_id3 1.0
:
;
Inventory_id7_A1 -> display_START_id3;
display_START_id3 -> Inventory_id7_A3;
Inventory_id7_A3 -> Inventory_id7_A4
-1

A RefTask8
s RefTask8_A1 1.0
z RefTask8_A1 Client_id5_E1 1.0
-1

A Server_id6
s Server_id6_A1 1.0
s Server_id6_A3 1.0
z Server_id6_A3 Inventory_id7_E1 1.0
s browse_STARTC_id2 1.0
:
;
Server_id6_A1 -> browse_STARTC_id2;
browse_STARTC_id2 -> Server_id6_A3
-1
```