

An Approach to Predicting Performance for Component Based Systems

by

Xiuping Wu, B.CS.

A thesis submitted to the Faculty of Graduate Studies and Research in
partial fulfillment of the requirements of the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, K1S 5B6
Canada

July 20th, 2003

©2003, Xiuping Wu

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

**An Approach to Predicting Performance for
Component Based Systems**

Submitted by Xiuping Wu, B.CS. in partial fulfillment of the
requirements of the degree of Master of Applied Science

Dr. Rafik Goubran, Chair

Department of Systems and Computer Engineering

Dr. C. Murray Woodside, Thesis Supervisor

Carleton University
July 20th, 2003

ABSTRACT

Component Based Software Engineering has emerged as a promising paradigm for software engineering. It brings higher efficiency and better quality by using reusable software components. It also offers some potential advantages for performance engineering. If a planned system involves many pre-existing components, it could be easier to model it than a system with completely new components. In this research, each software component is represented by a parameterized performance sub-model stored in a library. These sub-models extract performance sensitive attributes of the software components. By using these sub-models, system models can be built quickly for many different configurations. As well, those configurations are tied to software configurations. An approach is described to model component based software systems based on these sub-models and a system assembly model. A language and a tool have been developed to help generate system performance models. The final performance models are in the format of a layered queuing network model. A case study illustrates the use of the tool.

Acknowledgments

First of all, I would like to give my big thank you to my supervisor, Professor C. Murray Woodside. Thank you for your invaluable guidance and advice throughout my thesis research.

My second thank you goes to my husband, who always supports me and understands me. Thank you for always encouraging me. Without your support, definitely, I could not have achieved so much. For all my family, I would like to say thank you for your love and understanding during my study.

I also want to extend my thanks to people in the RADS lab who help to create the great working environment.

A very special thank you goes to Godfrey Lee, Jim Miller-Cushon and Bob Minns from Cognos for their help in understanding issues of performance modeling for software product lines.

Financial support provided by Nortel Networks Scholarship and Carleton University is greatly appreciated.

TABLE OF CONTENTS

Abstract	i
Acknowledgments	ii
Table of Contents.....	iii
Chapter 1 Introduction	1
1.1 Motivation and Objective	1
1.2 Thesis goals.....	2
1.3 Contributions.....	3
1.4 Thesis Organization	3
Chapter 2 Background.....	5
2.1 Software Performance Engineering (SPE)	5
2.1.1 Introduction to SPE	5
2.1.2 Methods for Building Software Performance Models	7
2.2 Component Based Software Engineering (CBSE)	9
2.2.1 Overview of Component Based Software Engineering (CBSE).....	9
2.2.2 Performance Modeling in CBSE	10
2.3 Layered Queuing Network (LQN) Performance Model.....	11
2.3.1 Software Bottlenecks.....	13
2.3.2 LQN Graphical Notations.	14
2.3.3 LQN Solvers and Input File Format	16
2.4 The CB-LQN Component Model.....	18
2.4.1 The Structure of CB-LQN Component Model.....	19

2.4.2	Protocol for Plugging the CB-LQN Component into the System Model	21
2.4.3	Advantages and Limitations of the CB-LQN Component Model .	21
Chapter 3 LQML: An XML Based Language for the LQN Component Model and Assembly Model.....		
	Assembly Model.....	23
3.1	Overview of XML, XML Schema and XSLT	23
3.1.1	The eXtensible Markup Language (XML).....	23
3.1.2	XML Schema (XSD) Schema.....	24
3.1.3	The eXtensible Stylesheet Language Transformations (XSLT). 25	
3.2	Overview of the XML-Based LQML Component Model	26
3.3	Apply XML to LQN Definition	28
3.3.1	Why XML Applies to LQN Definition.....	28
3.3.2	XML Schema for LQML Sub-model and Assembly Model Definition	29
3.3.2.1	New Elements Introduced to LQML	33
3.3.2.1.1	Slot.....	33
3.3.2.1.2	Phase activities.....	36
3.3.2.1.3	Task-activities	37
3.3.2.1.4	Service.....	41
3.3.2.2	LQN Core	42
3.3.2.2.1	Processor in LQN Core.....	43
3.3.2.2.2	Task in LQN Core.....	44
3.3.2.2.3	Activities in LQN Core.....	45

3.3.2.3	LQN Sub-model	46
3.3.2.4	LQN Assembly Model	48
3.4	An Example of Components Assembly	49
3.5	Approaches to Creating LQN Component Models.....	54
Chapter 4	An Approach to Component Based Performance Modeling.....	56
4.1	Design Issues.....	56
4.2	Overview of the Present Approach.....	57
4.3	LQN Assembly Model.....	58
4.3.1	An Example of an LQN Assembly Model	59
4.3.2	The Reusability and Adaptability of the Assembly Model	60
4.4	Tool – LQComposer	61
4.4.1	Overview of Tool Design.....	62
4.4.2	LQNAssemble Design.....	63
4.4.3	xml2LQN Design	66
4.4.3.1	Template Algorithm in xml2LQN Stylesheet.....	72
4.5	Tool Validation	80
4.5.1	Validation of LQNAssemble.....	80
4.5.2	Validation of xml2LQN	84
4.5.3	Validation Against the Combined Tool –LQComposer.....	94
4.5.4	Conclusions.....	94
Chapter 5	Industrial Case Study.....	96
5.1	A Conceptual Performance Model for a Management Information System (MIS)	96

5.2	The Component Based Approach to Model the MIS	99
5.2.1	The Assembly Model for the MIS	100
5.2.2	The Application Server Component Model.....	101
5.2.3	Resulting Model from LQComposer	102
5.3	Performance Results for the Base Case.....	105
5.4	Performance Results with Multithreading.....	107
5.5	Performance Results with Replicated Application Nodes.....	114
5.6	Scaling Limits	118
Chapter 6 Conclusions.....		122
6.1	Conclusions	122
6.2	Limitations	123
6.3	Future Research.....	123
Reference		124
Appendix A XSD Schema for LQML		129
A.1	XSD Schema for LQN Core (lqn-core.xsd)	129
A.2	XSD Schema for LQN Sub-model (lqn-sub.xsd)	136
A.3	XSD Schema for LQN Assembly Model (lqn.xsd)	137
Appendix B Some Input and Output Documents for LQN Models.....		139
B.1	XML Document for SingleMod (SingleMod.xml)	139
B.2	XML Document for NestedMod (NestedMod.xml).....	141
B.3	XML Document for Flattened NestedMod (NestedMod_flt.xml).....	143
B.4	Output of Transforming SimpleAbl (SimpleAbl.lqn)	145
B.5	XML Document for a Database Model (par-db.xml).....	146

B.6	Output of Transforming par-db.xml (par-db.lqn).....	152
B.7	XML Document for an LQN Model that Has Task Activities (task-act.xml)	154
B.8	Output Model from Transforming task-act.xml (task-act.lqn)	156
Appendix C	Source Files for LQN Models in the Case Study.....	157
C.1	XML Document for MISAssemble (MISAssemble.xml).....	157
C.2	XML Document for AppComp (AppComp.xml).....	160
C.3	Resulting Model in LQN Language (MISAssemble.lqn)	163

Chapter 1 Introduction

1.1 Motivation and objective

Early performance predictions are crucial to software system development. By building predictive performance models at design time, it helps to identify any potential design problems and this can be used as feedback to software system designers. It lowers the development cost as well as the runtime risks. In order to accomplish this, building predictive performance models is the key.

In recent years, Component Based Software Engineering (CBSE) has drawn more and more interest in both academic and industrial communities. However, much more work is focusing on functional aspects with relatively less reported on non-functional aspects such as performance. One difficulty is that a software component could be deployed in different environments. The need to consider its potential execution environments and incorporate them into the system performance modeling makes the problem more difficult. Sherif, Yacoub et. al. [44] proposed a way to characterize a software component, in which they mentioned non-functional attributes, without details about how to describe them and use them. Sitaraman et. al. [33] argued that performance specification of software components and assemblies is a basic problem that must be solved to enable software engineers to assemble systems from components. It would be very helpful if performance sub-models could be built and the performance attributes of each software component could be described in the sub-models. Then a system model can be built by assembling these sub-models. Therefore, the system performance prediction can be done by solving these system models. It is desirable if there are tools or methodologies that can automate these processes. This thesis work has been motivated by this thought.

Very often, when a system is planned, in order to predict the performance, the model has to be built from scratch even though it may have some pre-existing components. For large and complex systems, this is error-prone and tedious work. If the sub-models can be reused, it should be much easier to build system-level models. By using these sub-models, system models can be built more quickly for many different configurations which are tied to software configurations. By solving these system models, system capacity and its delay can then be predicted. Meanwhile, using these models, the performance sensitive attributes

can be studied. By changing some values of the parameters in the model, some parameters that system performance is sensitive to as well as those that are non-sensitive can be identified. As a result, instead of worrying about all the parameters, maybe only a few of them need to be studied. This obviously simplifies the system performance analysis and helps to address the important points.

The development of component based software systems can be generalized [43] as shown in the left side of Figure 1-1. Using the same specification as for the component-based software, the approach in this thesis is to assemble performance sub-models for these software components into a system-level performance model, using an automated tool for model assembly, as illustrated on the right side of the Figure.

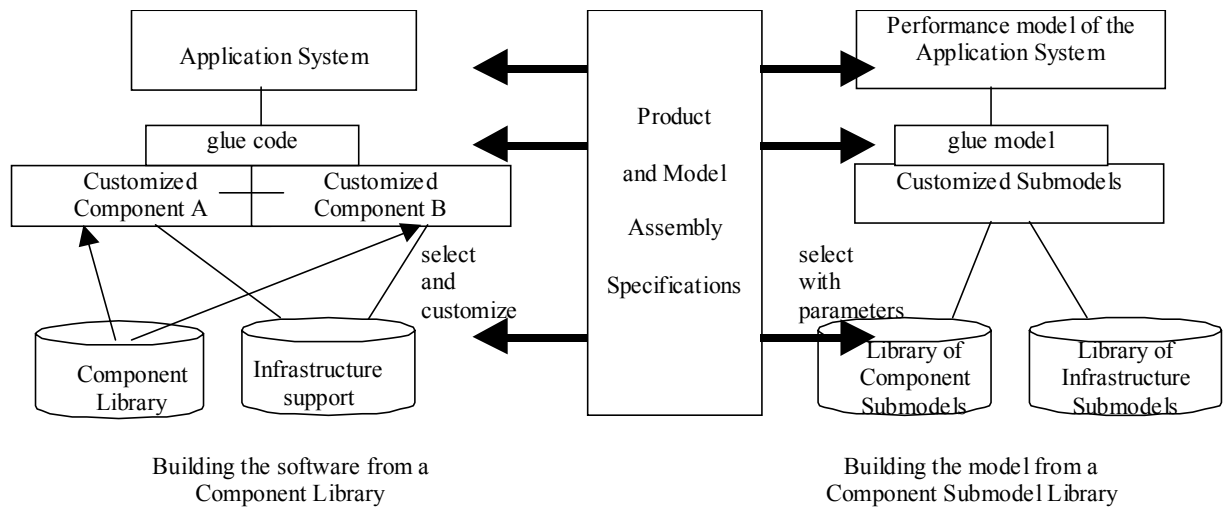


Figure 1-1 Component Based Software Development

1.2 Thesis goals

The goal of this research is to take some steps towards the model building system in Figure 1-1, concentrating on the right side of the Figure. An XML (eXtensible Markup Language) based language called LQML has been designed for the layered queuing models, that describes the use of components, and the component sub-models. A tool called LQComposer has been developed to assemble the component sub-models together with a

system model. This substantially extends a previous tool. In this work, the LQN models and its sub-models will adopt XML language since XML can better express the models and also many tools support development such as XML editors and parsers. So if an LQN model uses XML language, then this model can be displayed in any tools that understand XML. In order to express the constraints that the LQN model has, an XML schema for the sub-model and system model definition will be given. Currently, LQN model solvers can only accept the previous LQN language which is in plain text format. So the transformation has to be done to transform XML format to the LQN previous plain text format.

1.3 Contributions

This thesis makes the following contributions:

- An XML schema for LQN model and sub-model description as well as the language for composing performance sub-models have been proposed and developed.
- An improved version of component sub-model has been proposed and a tool (LQComposer) for generating system-level performance models from sub-models automatically has been developed.
- An industrial case study has been conducted including model assembly and analysis of some software performance problems.

1.4 Thesis Organization

This thesis is organized as follows. The first chapter gives the motivation and objective of this research project. The second chapter gives some background of this research which includes overview of Software Performance Engineering (SPE), Component Based Software Engineering, Layered Queuing Network models and the CB-LQN component model. Chapter 3 first gives some brief introductions to XML and its related tools. Then it introduces the LQML component model. After that, it describes how XML language can be applied to LQN definition. It then elaborates the schema of the proposed LQML language for LQN component model and assembly model. Chapter 4 describes the present approach

to model component based software systems. This chapter also describes the tool LQComposer which generates performance models automatically from the sub-models. Chapter 5 presents an industrial case study. In this chapter, it describes how to apply the tool LQComposer in this study. Later, some results as well as performance analysis are presented. Chapter 6 gives the conclusions for this thesis and it also points out some future work.

Chapter 2 Background

This chapter covers some background for Software Performance Engineering, Component Based Software Engineering, Layered Queuing Network (LQN) performance models, software bottlenecks and the previous CB-LQN component model. The proposed LQML performance sub-model is based on this one. The terms component model and sub-model are equivalent and used interchangeably in the whole thesis.

2.1 Software Performance Engineering (SPE)

This section provides some introduction to software performance engineering (SPE) and different approaches to SPE.

2.1.1 Introduction to SPE

SPE is a systematic approach that provides quantitative assessment for the emerging software systems so that their performance objective can be achieved [34]. Performance usually refers to system response time seen by the end user. SPE addresses the performance issues such as bottlenecks, system delay, system capacity as well as system scalability in the entire life cycle of software development. The reason that performance issues are becoming so important is that performance failures may delay the time to market and it may result in losing customers. There are different approaches towards software performance engineering.

- Performance Measurement. In this approach, system performance metrics are gained based on the experiment or operational system. Although it provides more accurate results and it may also be able to identify some performance problems such as bottlenecks, it still has some drawbacks. For performance testing based on experiments such as prototyping, it could be time-consuming to develop the prototype and collect data from lots of tests. Further, it may not be able to give the feedback to system design if the system has to meet a tight deadline. So this probably is a “fix-it-later” approach [34]. Meanwhile, it maybe costly to set up the experiments and do the tests. For performance evaluation based on the operational

system, this really falls in the “fix-it-later” approach. The performance problems detected at this stage are usually hard to fix if they are originated from the system design and most often they are.

- Performance Tuning. In this approach, performance improvement is done on an existing system. The tuning is often performed by an expert who is knowledgeable of the system architecture and design. However, the improvement could be very limited and it won't be able to handle the performance problems if they are fundamental in the software design. This is also the “fix-it-later” approach.
- Early prediction through performance models. In the above two approaches, performance problems are detected late in the development process which may then become very hard to fix. It is not easy as fixing coding bugs [34]. Sometimes it may involve very intensive implementation changes. The reason for this is that performance defects often exist in the software architecture or design which means performance problems are introduced in the early stages. As consequences, performance failures may delay the time to market and it may result in losing customers. Hence, early performance prediction is very important especially for performance-critical software systems. It provides cost-effective ways to avoid performance failures. The ability to predict performance at very early stage when changes can be easily made is desirable. The approach to early prediction is by building predictive performance models. By analyzing these models, the performance characteristics can be explored so that the emerging software system can be evaluated to decide whether the performance objective can be met. The results of these performance models can also be used as feedback to software architects or designers so they make any necessary changes to the architecture or design. On the other hand, the performance models are very helpful in that it can predict the results of many alternatives that the system architecture and design may have. Therefore, by studying these results, the appropriate architecture and design can be determined.

2.1.2 Methods for Building Software Performance Models

This section provides a quick overview of two main methods for building software performance models that are not used directly in this research. They are Queuing Network models and Stochastic Petri Nets. The following sections will describe them very briefly.

- Queuing Network Models

Queuing Network models consist of a number of servers and customers (which are also known as tokens or jobs). Customers make requests to the servers. Depending on the status of the server, the requests may be queued if the server is busy or get processed if the server is idle. There is a special kind of server called delay server or infinite server where no queue happens. Customers get serviced immediately. There are three kinds of Queuing Network models classified as open, closed and mix. In an open queuing network, customers can enter or leave the network. There is no fixed population. In a closed queuing network, customers circulate the servers; therefore, the population is fixed. In the mixed queuing network, some customers may leave or enter the network, whereas some circulate in the network. In a queuing network model, customers are grouped based on their statistical behavior. As a result, there may be multiple classes of customers in one network. The main parameters for a queuing network model are mean service time per request at per server and the visit ratio that a customer makes to the server. However, its performance is also relevant to other parameters such as the scheduling policy and the queue size of the server. The queuing network models can be solved analytically or by simulation. The results of interests are throughput and mean response time.

Although traditional Queuing Network models are powerful in solving some performance issues, they are not able to capture software contention easily. For software systems, Queuing Network models model the software as pure customers and the devices as pure servers. They can not model an intermediate software server that accepts requests from other software processes and may also send requests to other processes. They also cannot incorporate the case where a job has phases of execution

where an early reply occurs. Parallelism that involves forks and joins is hard to be modeled in Queuing Network models, too.

A paper of Daniel A. Menasce [23] discusses a technique of two-level iterative queuing modeling of software contention. In this paper, two queuing networks are considered: one for software resources and the other one for hardware resources. Although this approach is simple and straightforward, it is hard to apply to the large and complex software systems where multiple layers of software servers may be involved.

The Layered Queuing Network (LQN) models, which were previously called Stochastic Rendez-Vous Networks (SRVN) models, extend the traditional queuing network models by allowing for an arbitrary number of software layers which may act as clients or servers at different layers [30] [40]. The LQN models have been used in this thesis research and will be introduced in section 2.3.

- Stochastic Petri Nets

Petri nets were introduced as a model of computation for concurrent systems by Carl Adam Petri in 1962. A detailed introduction can be found in the book [1]. A Stochastic Petri Net model is a high-level model which generates a stochastic process. The stochastic process is then analyzed via Markov Chain technique. The performance measures are then obtained from the steady state probabilities. The main components in a Petri net are places and transitions. Places represent states or resources in the system while transitions model the activities. Places are graphically represented as circles and transitions are graphically represented as bars. Places and transitions are connected via a set of directed arcs. Places may contain tokens which are graphically represented as dots inside the circle. Tokens move through the network (from place to place) according to certain rules. In Stochastic Petri Nets, a transition is enabled if its input places have one or more tokens. Each transition has a firing time which is an exponentially distributed random event. An enabled transition has the probability of firing but it can only fire after an exponentially distributed amount of time elapsed. In

addition, only one enabled transition can be fired at any time. When a transition fires, a token is removed from the input places and added to the output places. The current state of the model is determined by the number of tokens in each place and this is also called a marking. These markings (states) generate the underlying Markov Chain models. The performance evaluation is achieved by solving these Markov Chain models.

Although the Stochastic Petri Net models are useful for performance modeling of computer systems that exhibit concurrency, synchronization and randomness, they suffer from state space explosions which make performance computation very hard.

2.2 Component Based Software Engineering (CBSE)

This section gives some background of CBSE. It also reviews some research work of performance engineering in this area.

2.2.1 Overview of Component Based Software Engineering (CBSE)

CBSE has emerged as a promising paradigm for software engineering. It brings higher efficiency and better quality by exploiting reusability. In the traditional approach, when a software system is going to be developed, the implementation has to be done from scratch. While CBSE emphasize on building system by reusing high quality configurable software components. This not only reduces its development cost and time-to-market but also ensures higher reliability and better maintainability. The development process of CBSE is quite different from the traditional software engineering approach. In the paper of [3], it has been generalized as having the following steps: Component Requirement Analysis, Component Development, Component Certification, Component Customization, System Architecture Design, System Integration, System Testing and System Maintenance.

There are two central parts in CBSE: components and architecture. Components are the basic building blocks while architecture describes how components are assembled into an application system. However, there is no unified definition for software components. Clemens Szyperski and William T. Council gave their own definitions respectively in the

books of [36] [14]. In general, the following characteristics are included in a software component:

- 1) It is an independent, compositional and deployable unit.
- 2) It has clearly defined and documented interfaces interacting with other components.
- 3) It has certain functionalities.
- 4) It may have explicit context dependencies such as operating system or other software components, etc.

Interests in components and architectures are now present in both academic and industrial community. Today, a business application can become so large and complex that some of the components must be developed separately. These components must be adaptable to the integrated system [25]. All these interests and demands lead to defining architecture standards as well as the development of component technologies. Among these are 1) Common Object Request Broker Architecture (CORBA) from OMG. 2) Java Beans and Enterprise Java Beans (EJB) from Sun Microsystems and 3) Component Object Model (COM) and Distributed COM from Microsoft. A comparison of these technologies can be found in [3]. The component based approach is now widely used in Software Product Line [2] [6].

2.2.2 Performance Modeling in CBSE

Research in CBSE has mainly focused on functional aspects with relatively little reported on non-functional aspects. Performance prediction remains as one of the key challenges in this area. One of the difficulties is that a software component may execute in different environments and this makes it hard to know the performance properties in advance. On the other hand, each component technology has a different infrastructure and implementation. As consequences, they may exhibit different performance characteristics which makes performance predictions harder. Some existing tools such as C. U. Smith's *SPE.ED* [35] are of limited use in these situations. M. Sitaraman in his paper [33] agrees

that software components should have performance specifications as well as functional specifications. Due to the fact that components have parameters, it is very hard to give its performance specifications in general. Classical techniques and notations for performance analysis do not work. He pinpoints that performance specifications are very basic problems which must be solved before components can be assembled in order that the resulting product can meet the specified performance goals. He also argues that performance specifications are necessary so that clients can decide which components will meet their goal. He suggests that performance specifications should include execution time and memory capacity requirements.

In the paper [4] the authors proposed an empirical approach to predict the performance of the assembled system by benchmarking and profiling. A model is then built based on the observations and this model is used as a performance predictor for a class of applications which are based on the specific component technology. Although the authors applied this approach to predict the performance of the StockOnline which is a test application and demonstrated the success, it is still time-consuming to make the observations from benchmarking and profiling.

2.3 Layered Queuing Network (LQN) Performance Model

This thesis work adopts the Layered Queuing Network (LQN) modeling technique to model component based systems. LQN models [39] extend the traditional queuing network models by considering both software and hardware contention, and the impact of layers on service time. The structure of an LQN model resembles the software architecture of the system. It is expressed as a set of objects called “tasks” offering services (like methods) called “entries”; entries of one task make requests to entries of others at lower layers. The modeling of sequential executions, parallel executions (AND Forks and Joins), alternative executions (OR Forks and Joins) as well as repetitive executions is accommodated by “activities”. Activities are the smallest unit of computation. They can accept requests from entries or send requests to entries. A task that does not receive any requests but only sends requests to lower layers is called a “reference task”. A reference task is the load generator of the system. Tasks are executed on processors which represent physical resources such as CPUs and disks. There are three kinds of interactions between tasks.

1. Synchronous message. In this case, sender sends the message and is blocked, waiting for the reply.
2. Asynchronous messages. Sender sends the message and then continues doing its work, no blocking in this case.
3. Forwarding messages. The first sender sends a synchronous message. But the receiver does not reply directly, it does partial processing and then forwards the messages to a third party. The third party will reply back to the original sender. There may be multiple intermediate receivers that forward the request further. The last receiver is expected to send the reply back to the original sender.

In synchronous type of interactions, the receiver (including the intermediate receivers in case of forwarding) may have two phases of execution. The first phase consists of the activities that occur between its accepting the synchronous request and replying (or forwarding) it back. The second phase consists of the activities that occur after the reply. The introduced second phase can release the sender from its blocking state earlier which promotes some concurrency between the sender and the receiver. This was also termed as an “aggressive” reply as in the paper [11].

These three kinds of interactions are illustrated in the following UML sequence diagrams.

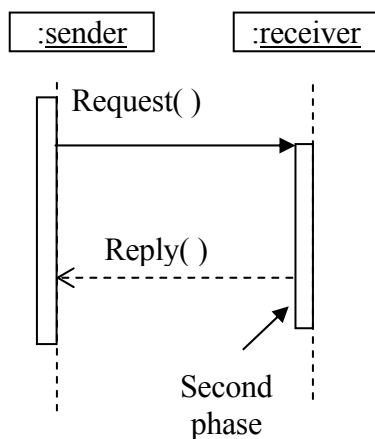


Figure 2-1a A Synchronous Interaction

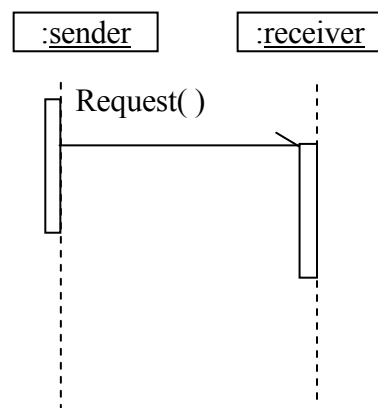


Figure 2-1b An Asynchronous Interaction

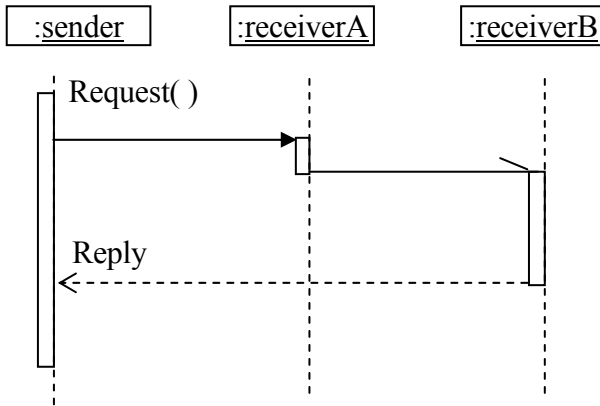


Figure 2-1c A Forwarding Interaction

The LQN modeling technique was formulated not only for the one level client-server interaction systems but also those that have multiple levels [12]. Therefore, an LQN can model the intermediate software servers which are very common in the large and distributed systems. It enables the detection of software bottlenecks (which will be introduced in the next section) as well as hardware bottlenecks. LQN models are suitable for many important classes of systems, including those that use Remote Procedure Calls (RPC) and distributed systems such as distributed database systems, telecommunication systems and agent systems [17] [37] [31].

The performance measures obtained by solving the LQN models are detailed at different levels which include service times of each entry, throughput and utilization per phase of each entry and each task, the utilization and waiting time of each processor and some values of variance and bounds. For this thesis work, the metrics of interests are throughput and utilization of each task and each processor.

2.3.1 Software Bottlenecks

A bottleneck is a single point of contention that limits the overall system performance [24]. In performance analysis, hardware bottlenecks are well understood in the conventional queuing network models of computer systems (e.g. [20]). They usually occur

at a CPU or a disk or other devices. However, in the systems that have multiple levels of client-servers, performance can also be constrained by software tasks, especially those that act as intermediate servers or “active servers” as they are called in [21]. The software task that is fully utilized while the resources it uses are underutilized, and thus constrains the total system performance, is termed a software bottleneck [24]. Software bottlenecking is quite different from hardware bottlenecking in that, when a software task is highly utilized, it will “push back” on its clients which makes them appear to be saturated, too. In those systems that are deeply layered, software blocking spreads upwards and may affect a large part of the system. At the same time, the resources that these tasks use are underutilized. A detailed discussion of software bottlenecking can be found in [24].

A typical example of software bottlenecking is the case where a single-threaded task is waiting for the I/O operation. Another example is shown in [21] where the model has two levels of servers. The active server, which has the roles of both client to its lower layer and server to its upper layer, could become the software bottleneck in the system if it is not multithreaded.

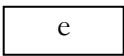
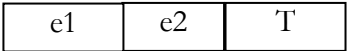
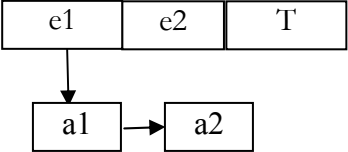


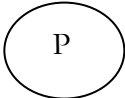
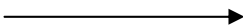

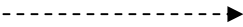
LQN models generalize the traditional queuing network models by capturing both hardware contention and software contention and the impact of layers. They can be applied to detect both hardware bottlenecks and software bottlenecks for performance analysis.

The guideline for eliminating software bottlenecks is by multithreading or cloning the software task that causes performance constraints [24]. By doing so, the utilization of the underlying resources can be enhanced and the overall system performance can be improved.

2.3.2 LQN Graphical Notations

Some graphical notations in LQN models are shown in table 2-1 below.

Table 2-1 Some Graphical Notations in LQN models

Name	Graphical Notation	Description
Entry		Each entry has a unique name in the model.
Task that has entries		The right side rectangle represents Task. The rest are for entries. Task T has two entries e1 and e2.
Task that has entries and activities		Activity a1 is the starting activity associated with entry e1.
Activity sequence		OR-Fork OR-Join
		And-Fork And-Join
Processor		Host on which the service is executed.
Synchronous call		Send and wait for the reply.
Asynchronous call		Send and continues, no reply.
Forwarding call		The call is forwarded to a third party by the receiver.

An example of LQN model is shown in Figure 2-1. This example is for a Web Server as described in the paper [9]. In this example, clients send requests to the web server. On the web server, the listener accepts these requests and based on the types of the requests, they are forwarded to the corresponding services in the server pool. These services need to access the disks. After the results have been worked out, they are sent back to the clients. The parameters such as CPU demands and number of interactions between entries are not shown in this model.

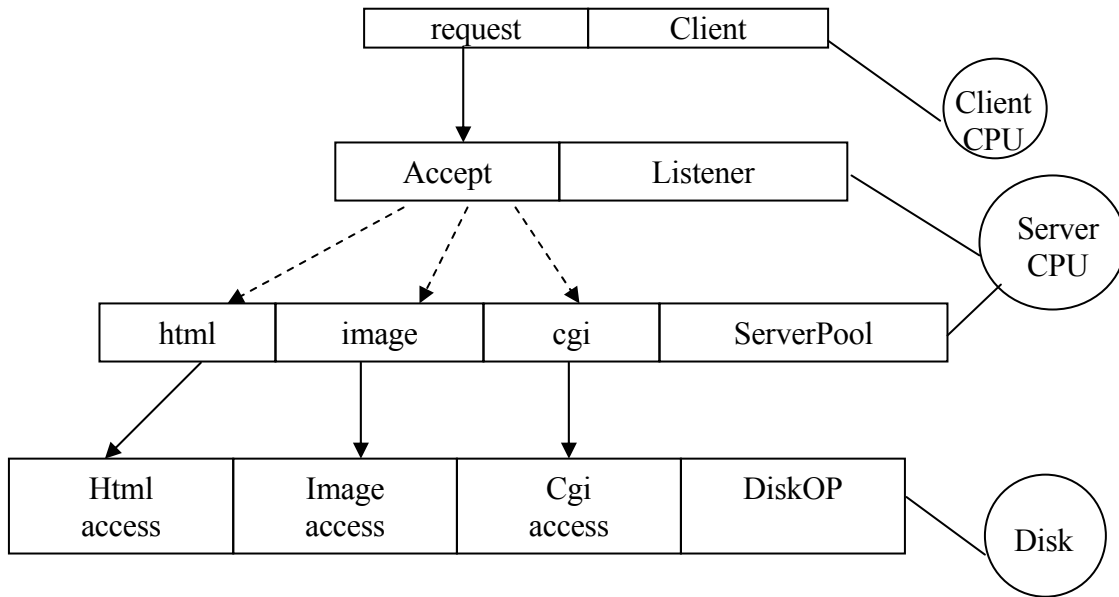


Figure 2-1 An LQN model for a Web Server

2.3.3 LQN Solvers and Input File Format

For LQN modeling analysis, performance metrics are obtained by solving the LQN models. There are two solvers available. One is called LQNS which uses analytical mean-value queuing approximations to solve the queues at all entries [13]. The other one is a simulator called ParaSRVN which uses the ParaSol simulation environment [7]. There is another tool that supports experiment instrumentation called Spex [15]. This tool can execute parameterized experiments using LQNS or ParaSRVN.

The LQNS and ParaSRVN program accept the same input format. The input file is a text file which provides model parameters for the solvers and has the following format in the exact order described below.

1. G section. This is the general information section that declares some control parameters for model solvers. It starts with a 'G' followed by the parameters of convergence criterion, iteration limit, print interval and under-relaxation. It ends with a '-1'.

2. P section. This section describes the parameters of all the processors in the model. It begins with a 'P' followed by the number of processors. For each processor, it begins with a 'p' followed by its id and a flag of scheduling discipline, 'f' for FIFO (First In First Out), 'r' for random, 'p' for preemptive, 'h' for hol (head of line) or non-pre-emptive and 's' for processor sharing. Then following this is its multiplicity (if applicable) and number of replications (if applicable). This section ends with a '-1'.
3. T section. This section describes the parameters for all the tasks in the model. It begins with a 'T' followed by the number of tasks. Each task starts with a 't' followed by its id, flag of reference ('r' for reference task and 'n' for non-reference task); followed by list of entries in this task, then '-1', then the processor id and its multiplicity. This section ends with a '-1'.
4. E section. This section describes the parameters for all the entries in the model. It starts with an 'E' followed by the number of entries. Each entry may have more than one description. The starting letter 's' denotes its service demands, while 'c' denotes the service coefficient variation. If it has an arrival rate, it will be declared by an 'a' followed by its named and the arrival rate. If it makes requests to other entries, it will have a description denoted by a 'y' (for Synchronous) or 'z' (for Asynchronous) or 'F' (for forwarding). If it has activities, it will be declared as an 'A' followed by the starting activity name. This section ends with a '-1'.
5. A section. This is the activity section. Each task that has activities will have an activity section. Each section begins with 'A' followed by the task id. For each activity, its host demand must be declared which begins with an 's' then followed by the activity name and the demand. The declaration of the request it makes has the same format as an entry. If the call is deterministic, it is denoted by an 'f' followed by the destination name and the number of calls. Then separated by a ':', the declarations of activity connections are followed. Some examples are illustrated as follows.

$A1 \rightarrow A2$ means that activity A1 is followed by activity A2 in sequence.

$A1 \rightarrow A2 \& A3$ This is an AND fork which means that activity A1 is followed by both A2 and A3.

$A1 \rightarrow (0.4)A2 + (0.6)A3$ This is an OR fork which means that activity A1 is followed by either A2 or A3. The numbers in the parenthesis mean the probability of the choice.

$A1 \rightarrow 3.2 * A2, A3$ This means that activity A1 is followed by the repetitive activity of A2 which repeats 3.2 times on average. Activity A3 is executed after this loop.

$A1 + A2 \rightarrow A3$ This is an OR join which means that either A1 or A2 is followed by A3.

$A1 \& A2 \rightarrow A3$ This is an AND join which means that both A1 and A2 are followed by A3.

$A3[E3]$ This shows the reply activity which means that activity A3 replies back to entry E3.

Each activity section ends with a '-1'.

2.4 The CB-LQN Component Model

The CB-LQN component model [22] was developed by David McMullan when he worked at Carleton University. "A component is a pre-constructed LQN sub-model that can be plugged into another LQN model" [22]. It can be any subsystem. Its place in the system is represented by a pseudo task. Its input interface is represented by the entries defined in the component. The output interface is also represented by entries but these entries are defined in the harness which will be introduced in the later section. An LQN component is different from an LQN model in that a component has an interface section.

By knowing and using the interface, a component can be used to substitute a single task in the LQN model. This is very useful in the following case. When some details of a component are not clear, a single task with appropriate services can be used temporarily. When the details are obtained, a component could be constructed accordingly and the single task can be replaced with this component. This is often the case in the early planning and design phase.

2.4.1 The Structure of CB-LQN Component Model

A graphical example of this component model is shown in Figure 2-2 below. This is a much simplified component model of an application server in an Information System. The controller task interprets requests. The ReportGen task creates and edits reports on an SQL database, and the ResultsCache task stores report data for reuse, to assist the assembly of complex reports and the presentation of the same data from different points of view, or at different levels of detail. The harness portion represents the services that the component must obtain from the system in order to function.

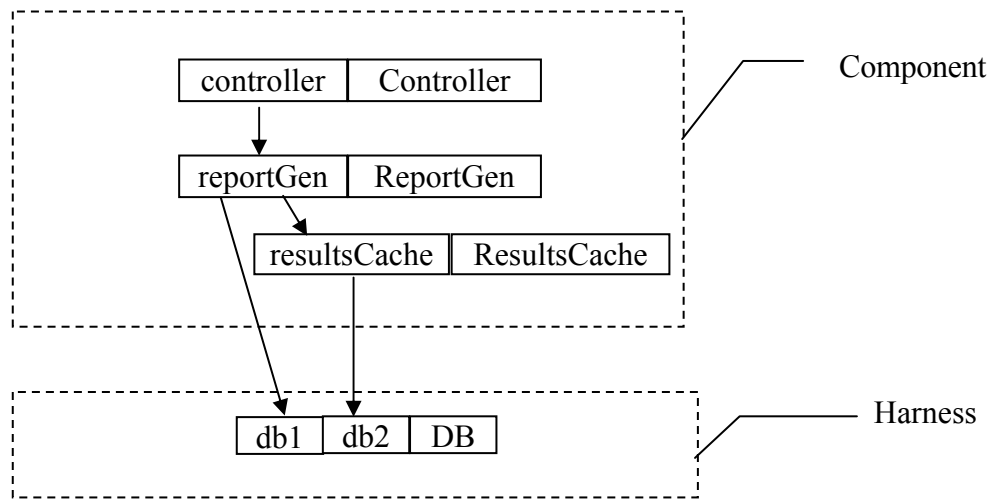


Figure 2-2 A graphical example of the CB-LQN component model

Figure 2-2 defines a class of components which can be instantiated one or more times within a model. The concept of component class is very similar to the concept of class in

Object-Oriented programming language. A component is an instance of component class with appropriate parameters passed in. A component class can have different instantiations in different situations with different parameters. The definition of a component class includes the following five sections.

1. Component Declaration.

This section has only one single line which is similar to the declaration of a method of a class. It has the component name and its associated list of variables. The variables in the component class may have default values. These variables may involve the following parameters: CPU service demands, service request parameters between elements in the component and threading levels.

2. Interface

This section defines the interface of the component. It includes the service entries that the component provides to outside elements and the request entries that the component requires from outside as well as the processors that can be replaced by outside processors in the system model.

3. General information

This is the same as that appears in the normal LQN definition [29]. It defines some parameters for the LQNS solvers [13].

4. Component definition

This is the body of component. It has the same format as a normal LQN definition which involves sections of processors, tasks and entries except that it may have variables prefixed by '\$' sign.

5. Environment harness.

This harness is a kind of template for the environment needed by the component, including driver tasks to provide inputs and stub tasks to provide service. Some of these elements may be optionally incorporated, with the component sub-model, in the model.

2.4.2 Protocol for Plugging the CB-LQN Component into the System Model

The component is plugged into the model by substituting the corresponding single task. This is done through the binding section. This section shows how a particular component is connected into a system. There is a separate binding section for each task that needs to be replaced by the corresponding component. A new instance of the component class will be created to replace it. Each binding section starts with a statement which includes the single task that is going to be replaced, the component class name, and the list of instantiation parameters.

2.4.3 Advantages and Limitations of the CB-LQN Component Model

The CB-LQN component model is very useful when a single task in the system model needs to be replaced by the component. By substituting the single task with its component, more details can be added into the system model and this brings benefits to system performance modeling. The component models are reusable and thus make system performance modeling easier if some components are pre-existing. Since current LQNS solvers cannot solve a model containing binding sections, a tool has been developed by David McMullan which is a pre-processor. This tool pre-processes an LQN system model which contains binding sections of components. The result of this pre-processor is a normal LQN model which can be solved by LQNS solvers. Some advantages of this tool and the component models have been explored in the paper [41].

However, although this approach seems feasible to bind performance sub-models for component based systems, it is very awkward to put into practical use. The assembly model needs to add some extra tasks in order for the binding sections to work properly. The binding sections are quite complicated due to the naming issues. Before the component is bound, the user has to be very clear about the new names of the bound request entries,

service entries and processors. The names of each element in the component instance have been changed by prefixing the replaced single task name. The harness substitution is more confusing and hard to use. It has to be matched in the system model; therefore, the user has to know exactly what the harness section in the component model is, which is beyond the component interface.

In this research, the CB-LQN component model was used to model the component based software systems. It has been found that it is easy for substituting but hard for composing. Therefore, in order to build performance models for component based software systems, the component model has to be improved so that not only does it reflect the performance attributes of software components, but also that it can be easily composed. In the following chapter, the improved version of LQN component model will be introduced.

Chapter 3 LQML: An XML Based Language for the LQN Component Model and Assembly Model

This section first gives some overview of the XML language and its related technology. Then it introduces the LQML which is an XML based language for LQN component model and the assembly model. Later it gives the details of the XML schema for these models definition.

The definition of this new version LQN component model and assembly model is in XML format. The reason of adopting XML format is that XML language is a standard with standard tools. XML is so flexible that any tags needed to describe the model can be added.

3.1 Overview of XML, XML Schema and XSLT

This section gives the overview of XML language, XML schema and XSLT stylesheet.

3.1.1 The eXtensible Markup Language (XML)

XML has been introduced by the World Wide Web Consortium (W3C) as a way to describe structured data. It is a text-format based, platform-independent markup language that is derived from SGML (Standard Generalized Markup Language). Since it is text-format, it is human readable and editable. It is similar to HTML but unlike HTML, one can define his/her own set of tags. The big advantage of XML documents is the separation of syntax and semantics. The content of an XML document is independent from its rendering. And because of this, XML is now playing an increasingly important role in data exchanging over the web.

An XML document is mainly composed of elements which are enclosed by start tags and end tags. Elements are nested and they can have attributes that are assigned values in the start tags. The whole document is contained by a distinguished root element which appears at the very beginning of the document [19].

An example of an XML document to describe a book in a catalog would be as follows.


```
<book>
  <title>XSLT Programmer's Reference</title>
  <author>
    <FirstName>Michael</FirstName>
    <LastName>Kay</LastName>
  </author>
  <ISBN>0-201-70485-4</ISBN>
  <price unit="US$">54.99</price>
</book>
```

The root element in this example is book. The element price has an attribute named unit which has the value of "US\$".

An XML document is said to be well-formed if its logical structure obeys certain formatting rules which enables a standard XML parser to parse it. For instance, a start tag must have a matched end tag. Tags must be strictly nested if the element has sub-elements. An XML document is said to be valid if it conforms to a pre-defined DTD (Document Type Definition) or to an XML schema which specifies the rules governing the structure of an XML document. In this thesis work, the validation of an XML document is performed against the XML schema which will be introduced in the next section.

3.1.2 XML Schema (XSD Schema)

The W3C Schema is a W3C Recommendation which intends to describe and constrain the content of a set of similar XML documents. It constrains the allowed structure of the XML documents with precision. It also imposes constraints on the datatypes that are permitted at individual locations within that structure [42]. The Schema Definition Language XSD is also in the form of an XML language. It is much more powerful than the previous DTD (Document Type Definition) in terms of datotyping and constraining the

document content. In an XML document, the content of the elements can be nested. The XML schema specifies content models for element types which describe the frequencies and the orders that the elements can appear in the content of the element type. It can also specify the possible attributes and their data ranges for an element type. Elements that have neither child elements nor possess attributes are said to have simple types, whereas elements having child elements or attributes are said to have complex types. Some of the elements have attributes and attributes always have simple types.

The relationship between an XML schema and its valid XML documents is similar to that of a class and its instances in the Object-Oriented programming language. Therefore, an XML schema can be used to validate a set of XML documents. Having a pre-defined schema, it enables the XSLT (eXtensible Stylesheet Language Transformations) transformation rules to be applied to a set of XML documents. The XSLT will be introduced in the next section.

3.1.3 The eXtensible Stylesheet Language Transformations (XSLT)

XSLT is a high-level declarative language that transforms an XML document into other kinds of text-based document such as HTML, XML or plain text document [18]. It manipulates the structure of the tree representation of the document. The transformation is expressed as a set of rules for transforming a source tree into a result tree [5]. The result tree is separate from the source tree. They can have completely different structures. The elements from the source tree can be re-ordered and filtered into the result tree. In addition, the result tree can add some extra structure or elements if necessary. The transformation is achieved by using templates that are associated with patterns. A pattern is matched against all elements in the source tree. A template is instantiated to create part of the result tree [5].

A transformation is defined in XSLT by a “stylesheet” which consists of a set of template rules. These rules are based on what should be generated in the result document if certain patterns occur in the source document. Therefore, a template rule includes two parts: a pattern and a template, which have been described in the previous paragraph. A

stylesheet can be used to transform a class of documents that conform to the same schema.

XSLT stylesheets describe the transformation rules that are associated with the specific class of XML source documents. The XSLT processor then applies the stylesheet to the XML document to generate the result document. This is illustrated in Figure 3-1.

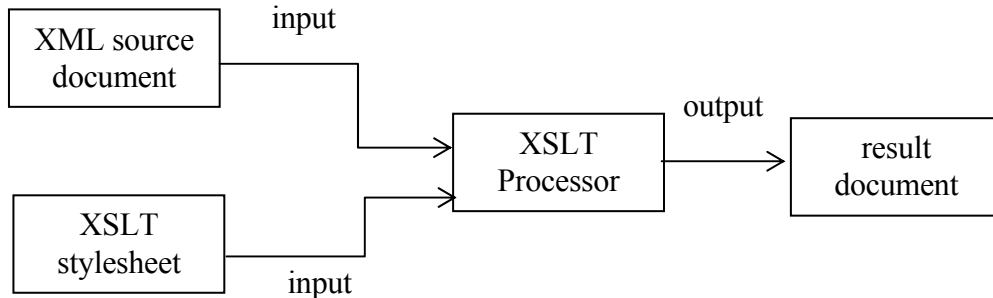


Figure 3-1 XSLT Transformation

There are several free XSLT processors available.

- Xalan. This is an open source XSLT processor from the Apache organization. This thesis uses this processor to generate text file documents. This processor can be downloaded from <http://xml.apache.org/>. It has Java and C++ versions. This thesis uses Xalan-Java. The command to invoke it is as follows:

```
java org.apache.xalan.xslt.Process -in <XML source document> -xsl <stylesheet>
```
- MSXML3. This processor is from Microsoft and can be downloaded at <http://msxml.com/msxml3.html>. It enables a stylesheet to run within Internet Explorer. Normally, it suits the case where the output is an HTML document.
- Saxon. This is another open source XSLT processor developed by Michael Kay as a Java application. It is downloadable from <http://saxon.sourceforge.net/>.

3.2 Overview of the XML-Based LQML Component Model

The goals of this new version component model are as follows.

- Making it more flexible for instantiating multiple instances of a single component class.
- Making it more flexible for tailoring the component with parameters.

Figure 3-2 below shows a graphical example of the application server component model which was presented before in Figure 2-2 in Chapter 2.

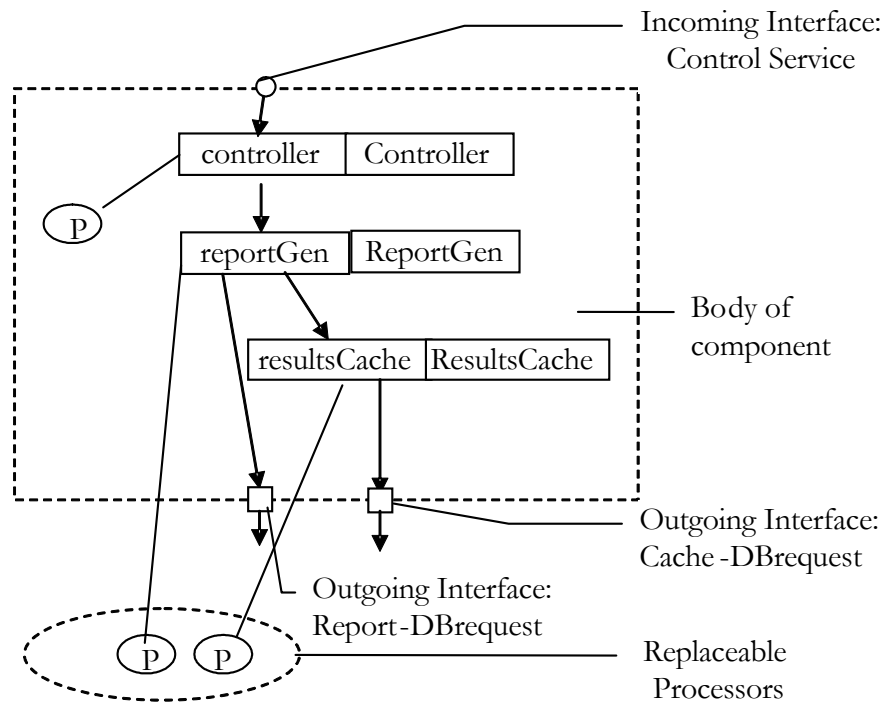


Figure 3-2 An LQML Component Model Example

The LQML component model has removed the harness part and classified the interfaces as incoming interfaces and outgoing interfaces. It also separates the replaceable processors from these interfaces. A replaceable processor is a kind of service serving a request for hosting tasks. Some processors are not replaceable and are shown inside the component body in Figure 3-2. The most outstanding difference is that in this version, the interfaces are separate elements other than those defined in the component body. In the CB-LQN model, these interfaces are actually the entries of some tasks in the component or in the

harness section. This is one of the reasons that make the binding section confusing and difficult to use in the CB-LQN model. However, in an LQML model, the incoming and outgoing interfaces are the extra and add-on parts of the component. The incoming interfaces are connected to the service entries in the component. The outgoing interfaces are connected from the request entries in the component. These interfaces have names which are used for specifying composition with other components. Meanwhile, since the definition of the component model has adopted XML format, there is an attribute for describing the interface named ‘description’. This may be very useful when composing several components since it helps users to understand what the specific interface means.

The syntax of the complete component definition language is described using XML schema which is attached in Appendix A. The details of the schema are described in the following section.

3.3 Apply XML to LQN Definition

This section describes the structure of the LQN model and the XML schema for LQN sub-model and assembly model definition.

3.3.1 Why XML Applies to LQN Definition?

XML is useful for expressing structured data. An LQN model can be viewed as having the following structure shown in Figure 3-3. This diagram uses UML notations.

In Figure 3-3, an LQN model is organized hierarchically, beginning with the processors. Each processor hosts several tasks that have entries serving different requests. A task may include activities that describe execution sequence or parallelism. Each entry has either one to two phases of execution, or a set of activities.

A set of activities that is associated with one entry should be defined as part of the entry. However, it is possible for a set of activities to involve two or more entries (for instance, to join flows arriving at two entries) and such activities should be defined in the task level. The two phases of execution in an entry can also be defined as two activities that happen one after another.

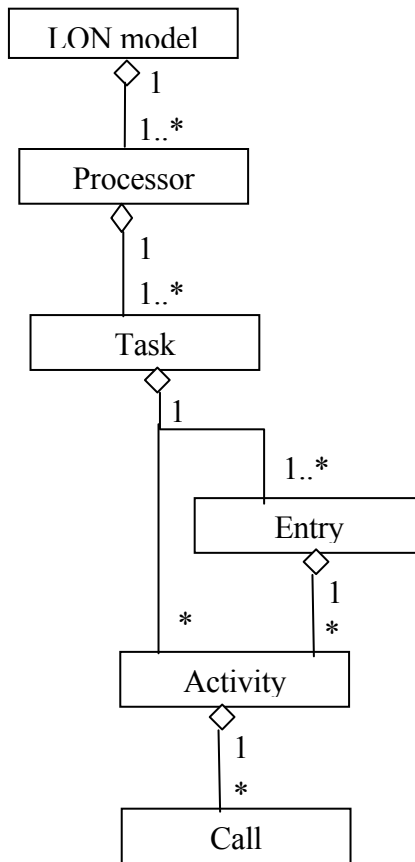


Figure 3-3 Structure of LQN model

Figure 3-3 shows that an LQN model is a well nested-structured model and can be fit into XML description exactly. Since there are many XML tools available now and some of them are free, it helps users to better understand the LQN model and have ways to edit and verify the syntax of the model. Some tools are so powerful that they can load the tags of the XML schema. The tool XMLWriter is such an example which has a trial version downloadable from <http://xmlwriter.net/>. Therefore, to define an LQN model a user will only need to drag and drop the appropriate tags and then fill in the parameters.

3.3.2 XML Schema for LQML Sub-model and Assembly Model Definition

In this new version, an LQN model is regarded as an assembly from sub-models. Therefore, it is also called LQN Assembly Model in this thesis. The syntax is described

using an XML schema. The detailed schema of these can be found in Appendix A. This section will elaborate the elements and attributes defined for the model and sub-model.

The assembly model and the sub-model share some common components in definition which are termed as LQN Core whose definitions include processors, tasks, entries, activities and the connections of inner sub-models. The term of “slot” has been introduced in order to be able to define nested models. It will be detailed in the next section.

Both assembly model and sub-model are capable of assembling components. They share the kernel of a model in common. However, a sub-model must have interfaces and is defined mainly for assembling in other models. The ability of assembly enables a sub-model to be defined with inner sub-model contained. Its resulting model is still a sub-model that has the same interfaces. Meanwhile, the assembly model must have run controls and is defined mainly for assembling components. Its resulting model is a system model ready for the solvers to solve.

The relationship between LQML sub-model and lqn-core is illustrated in Figure 3-4a while LQN assembly model and lqn-core is in Figure 3-4b. The top elements in lqn-core are illustrated in Figure 3-5.

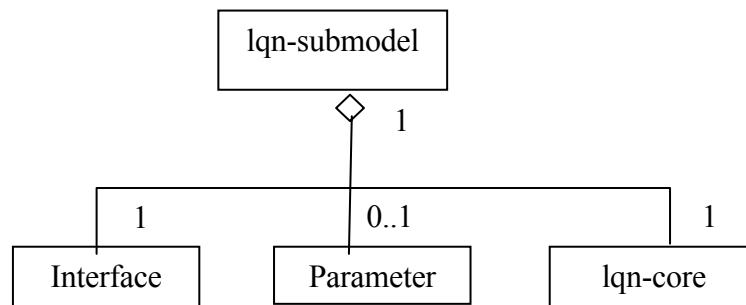


Figure 3-4a LQN sub-model definition structure

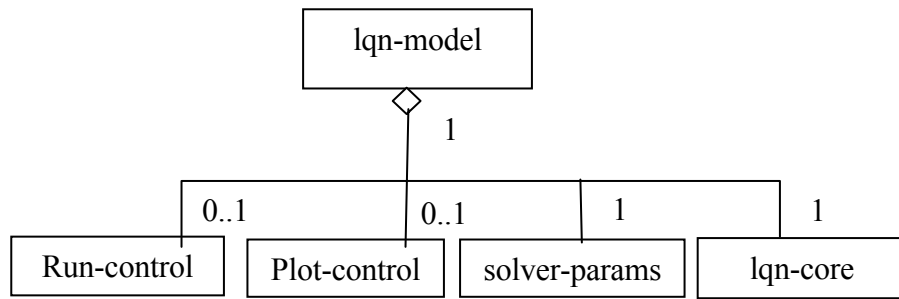


Figure 3-4b LQN assembly model definition structure

In the definitions of lqn-model and lqn-submodel, there is not an actual element of lqn-core. But instead, all the elements that are defined in lqn-core are included in lqn-model and lqn-submodel. The file lqn-core.xsd (for lqn-core definition) is included in both lqn-sub.xsd (for lqn-submodel definition) and lqn-model.xsd (for lqn-model definition).

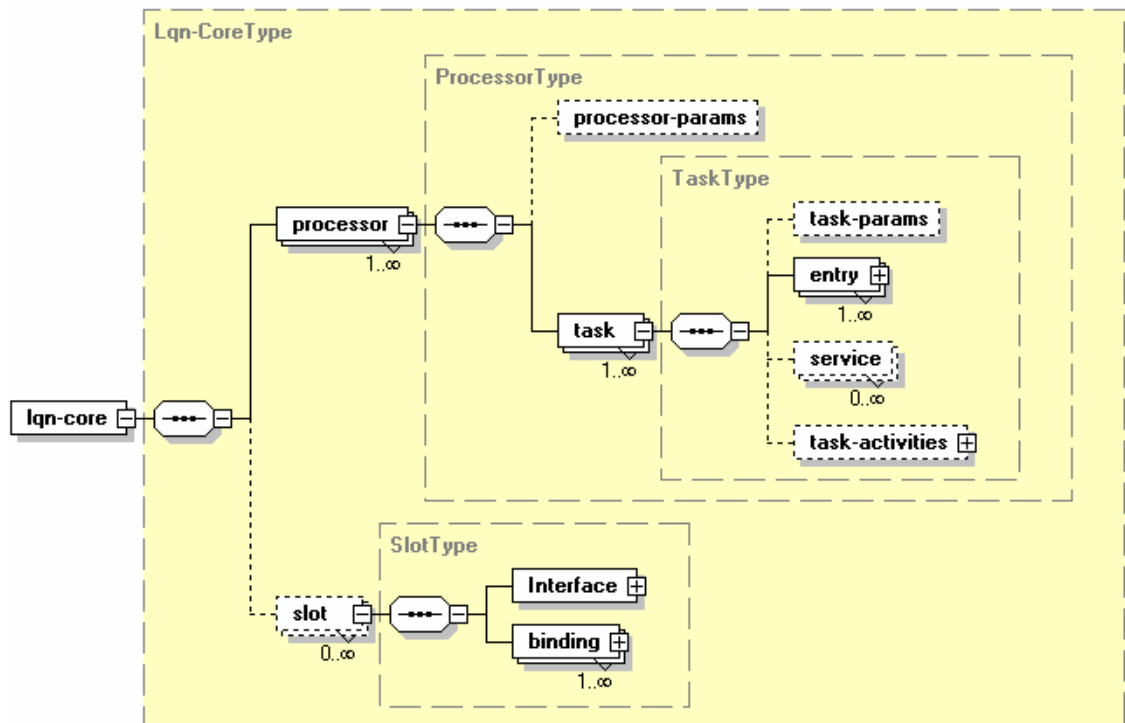

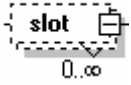





Figure 3-5 LQN Core definition structure

The figures that illustrate the schema structures at different levels in this chapter are generated by the XMLSpy Schema editor. The meanings of some graphical notations that are present in these figures are explained in the following Table 3-1.

Table 3-1 The Graphical Notations in the Schema Diagrams Generated by XMLSpy

Graphical Notation	Description
	<p>Element lqn-model contains several child elements.</p> <p>The solid rectangle means element lqn-model is a required element.</p>
	<p>The dashed rectangle means element slot is an optional element.</p> <p>Numbers 0..∞ represent the cardinality of the element.</p>
	<p>Alternatives. Only one of them will appear as a child.</p>
	<p>The element has child elements which have not been expanded.</p>
	<p>The three short lines in the upper left corner means that the element pre has neither child elements nor attributes.</p>

These figures only show the elements that are defined in the schema whereas attributes cannot be presented graphically using XMLSpy. The XML based language definition has introduced some new elements to the models that have not appeared in the previous LQN language. These new elements will be introduced in the next section and the other elements in the XML schema definitions are detailed in the section that follows.

3.3.2.1 New Elements Introduced to LQML

Four new elements have been introduced to the XML-based model: slot, phase activities, task-activities and service.

3.3.2.1.1 Slot

The concept of “slot” is introduced as a sort of “middleware” that connects an inner component to the rest of the model. A slot is actually a placeholder for an inner component that will be plugged into it. It describes the outer interfaces to the rest of the model while its bindings describe how its ports are connected to the interfaces of the inner component. By using slots, a nested component model can be defined. Hence, a component could be built from smaller ones to a larger one. An example of this is illustrated in Figure 3-6.

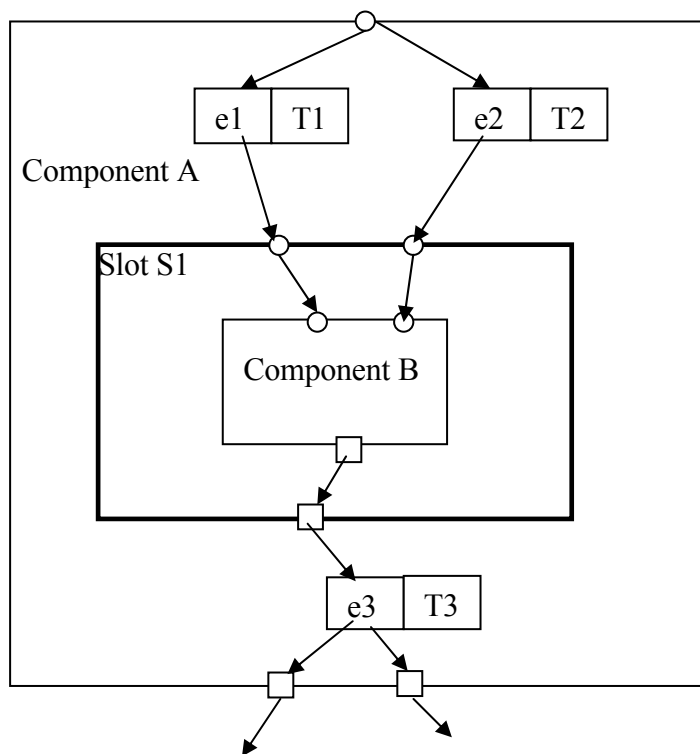


Figure 3-6 An example of a nested component

In this example, component A is the outer component and component B is the inner one that will be plugged into. The thick box represents a slot defined in component A. The slot is named S1. The circles represent incoming interfaces while rectangles are for outgoing interfaces. In this example, the inner component is actually bound to a slot. The advantages of introducing slots are as follows.

1. Slots have ports that connect to the outer model and the inner component. In order to build the outer model, modeler only needs to know the interfaces of the inner component and how these interfaces are connected to the slot.
2. Each slot has a name, and this name can be used as the prefix for all the elements of the inner component instances. Modelers do not need to worry about the naming issues to avoid any conflict that may arise. This is simpler than the previous CB-LQN component model.

The element slot is defined as one of the top elements of lqn-core. The elements that a slot contains are shown in Figure 3-7.

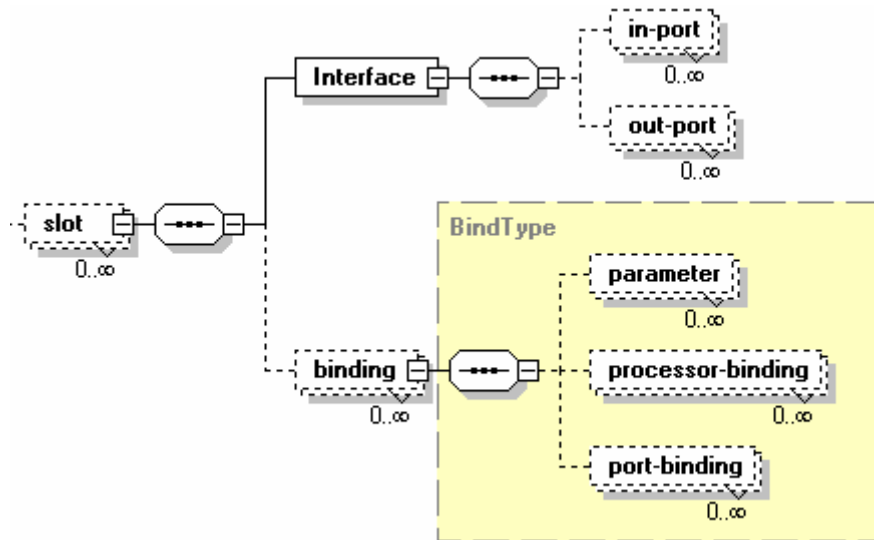


Figure 3-7 Elements defined within a slot

A slot has interfaces which are classified as in-port and out-port. Its binding is for inner component instantiation and customization. It describes how the inner component is connected to its ports. This includes three parts: parameter is used to give instantiation values for the inner sub-model parameters; processor-binding is for processor re-configuration which describes how the replaceable processors of the inner sub-model are replaced by the model processors if applicable; port-binding describes how the ports of the slots are connected to interfaces of the inner sub-models. The attributes of these elements are as listed in Table 3-2.

Table 3-2 Attributes of elements defined within a slot

Name of the Element	Attributes	Description of Attributes
Slot	id	An unique id of the slot
	bind-target	The name of the inner sub-model that is bound to the slot
	replic-num	The number of replicated sub-models needed
in-port	name	Name of the port
	connect-from	Which elements in the model connect to this port
	description	Descriptions of the port
out-port	name	Name of the port
	connect-to	Which elements in the model that this port connects to
	description	Descriptions of the port
parameter	name	The name of the parameter in the inner sub-model
	value	The redefined instantiation value of the parameter

Table 3-2 Attributes of elements defined within a slot (Contd.)

Name of the Element	Attributes	Description of Attributes
processor-binding	source	The name of the replaceable processor in the inner sub-model
	target	The name of the processor in the model that will replace the inner one
port-binding	source	The name of the port of inner sub-model
	target	The name of the port of the slot that will be connected to the inner sub-model

3.3.2.1.2 Phase activities

In the previous LQN model definition, activities and phases are defined as two separate and different elements as detailed in section 2.3.3. Activities belong to a task while phases are defined in an entry. Execution in the first phase is performed before a synchronous reply while second phase execution is done after the reply. The second phase execution introduces some concurrent operations between the sender and the receiver. As a matter of fact, phases can also be regarded as activities that occur one after another. Phase one is followed by phase two. The separation of activities and phase concept sometimes causes confusion; actually, there is no real distinction between phases and activities. In this XML based definition, the two phases are defined as Phase1 Activity and Phase2 Activity. These two phase activities are executed in sequence. Also the activities now belong to the entry instead of task except those that are originated from different entries join at some point, in which case the activities will be defined in the task-activities. The task-activities will be introduced in the following section.

3.3.2.1.3 Task-activities

The element task-activities is defined as one of the top elements within a task. It is used to describe activities that cannot be assigned to a single entry. As mentioned in section 3.3.1, there may be the case where activities originating from different entries need to synchronize. If this is the case, then all the activities have to be defined within the element task-activities. The elements that it contains are illustrated in Figure 3-8 below, without showing the details of element precedence. The switch sign means alternatives.

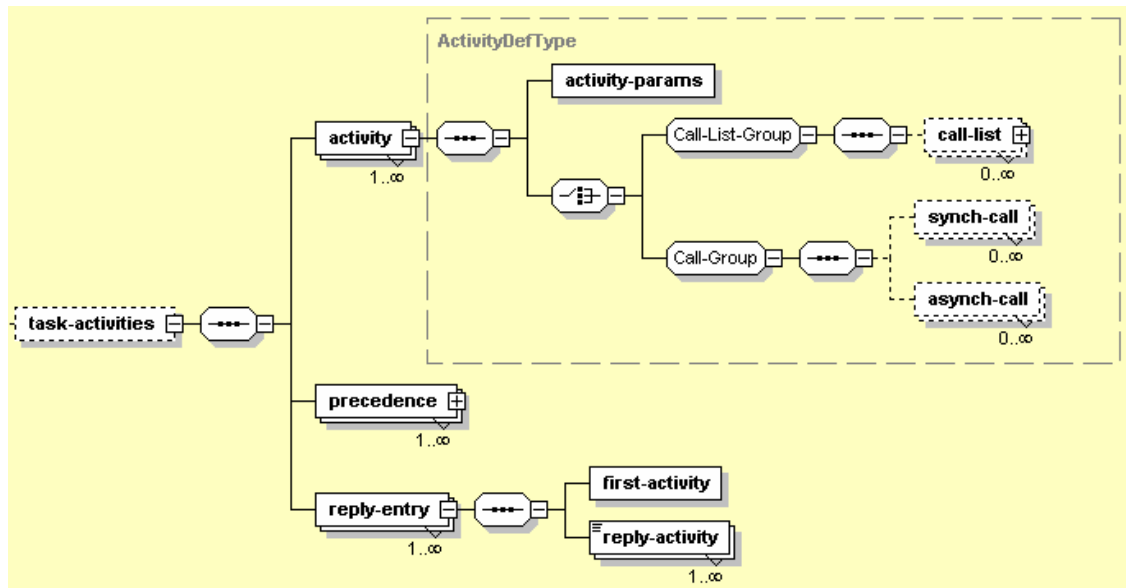


Figure 3-8 Elements within task-activities

The top elements within the task-activities are activity, precedence, and reply-entry.

The element activity defines all the activities within this task, including the activities parameters and calls that they have made. This element will be elaborated in detail later in section 3.3.2.2.3.

The element reply-entry defines a set of entries that are expecting replies from the activities. The element first-activity refers to this entry's start activity. The element reply-

activity refers to the activity that replies back to the entry. An entry may have more than one reply activities.

The element precedence defines the precedence relationships among the activities in this task. Its definition structure is shown in Figure 3-9.

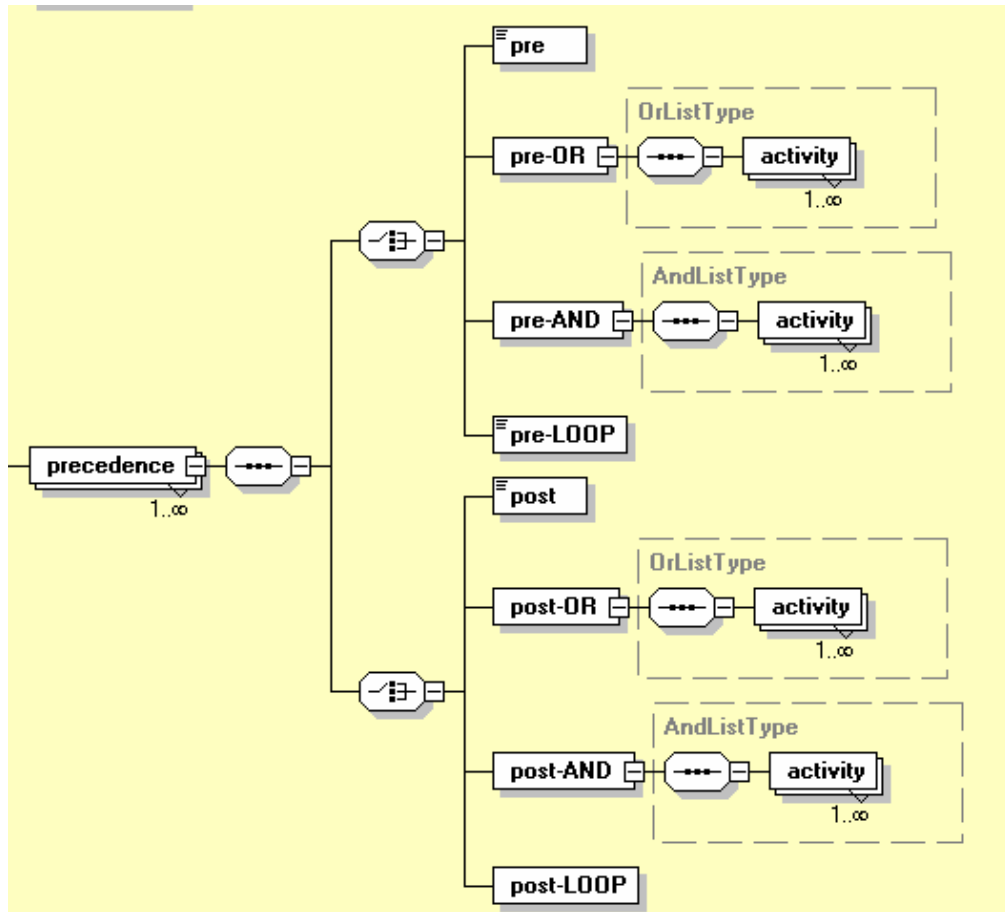


Figure 3-9 Elements within precedence

The expression of a precedence is split into two parts. Some examples are illustrated as follows.

Previously, $A1 \rightarrow A2$ means that activity A1 is followed by activity A2 in sequence. Now it has the following format.

<precedence>

<pre>A1</pre>

<post>A2</post>

</precedence>

Previously, $A1 \rightarrow A2 \& A3$ is an AND fork which means that activity A1 is followed by both A2 and A3. Now it has the following format.

<precedence>

<pre>A1</pre>

<post-AND>

<activity name="A2"/>

<activity name="A3"/>

</post-AND>

</precedence>

Previously, $A1 \rightarrow (0.4)A2 + (0.6)A3$ is an OR fork which means that activity A1 is followed by either A2 or A3. The numbers in the parenthesis mean the probability of the choice. Now it has the following format.

<precedence>

<pre>A1</pre>

<post-OR>

<activity name="A2" prob="0.4"/>


```
<activity name="A3" prob="0.6"/>
```

```
</post-OR>
```

```
</precedence>
```

Previously, $A1 \rightarrow 3.2 * A2, A3$ means that activity A1 is followed by the repetitive activity of A2 which repeats 3.2 times on average. Activity A3 is executed after this loop. Now it has the following format.

```
<precedence>
```

```
<pre-LOOP>A1</pre-LOOP>
```

```
<post-LOOP head="A2" count="3.2" end="A3"/>
```

```
</precedence>
```

Previously, $A1 + A2 \rightarrow A3$ is an OR join which means that either A1 or A2 is followed by A3. Now it has the following format.

```
<precedence>
```

```
<pre-OR>
```

```
<activity name="A1"/>
```

```
<activity name="A2"/>
```

```
</pre-OR>
```

```
<post>A3</post>
```

```
</precedence>
```

Previously, A1&A2->A3 is an AND join which means that both A1 and A2 are followed by A3. Now it has the following format.

```
<precedence>  
  
  <pre-AND>  
  
    <activity name="A1"/>  
  
    <activity name="A2"/>  
  
  </pre-AND>  
  
  <post>A3</post>  
  
</precedence>
```

Previously, A3[E3] shows the reply activity which means that activity A3 replies back to entry E3. Now it has the following format in task-activities definition, where activity A1 is supposed to be the start activity of entry E3.

```
<reply-entry name="E3">  
  
  <first-activity name="A1"/>  
  
  <reply-activity>A3</reply-activity>  
  
</reply-entry>
```

3.3.2.1.4 Service

The element “service” describes a self-contained outgoing interface of a task. Currently this feature has not been used for performance modeling. It has been introduced to support extensions, such as fault-tolerant LQNs [8]. Calls can be directed from an entry to the service point, which then targets the calls to different entries of servers at lower layers,

depending on their failure states. To be useful, calls should be able to be defined from a service to entries. Currently, this has not been implemented.

The element service is defined as one of the elements of task. It has an attribute of name which is the id of the service point.

3.3.2.2 LQN Core

LQN Core is the kernel of LQN assembly models and sub-models which consists of the definitions of all elements of the previous LQN language model, and the new elements which have been introduced in section 3.3.2.1. The main elements that are included in LQN Core are shown in Figure 3-10. This diagram has not been fully expanded to include all the elements due to the space limitation. However, each of the elements will be detailed in the rest of this section.

The LQN Core has two top elements: processor and slot. Slot has been elaborated in the previous section 3.3.2.1.1. Therefore, the rest of this section will focus on processor definition.

LQN Core is also described by Figure 3-3.

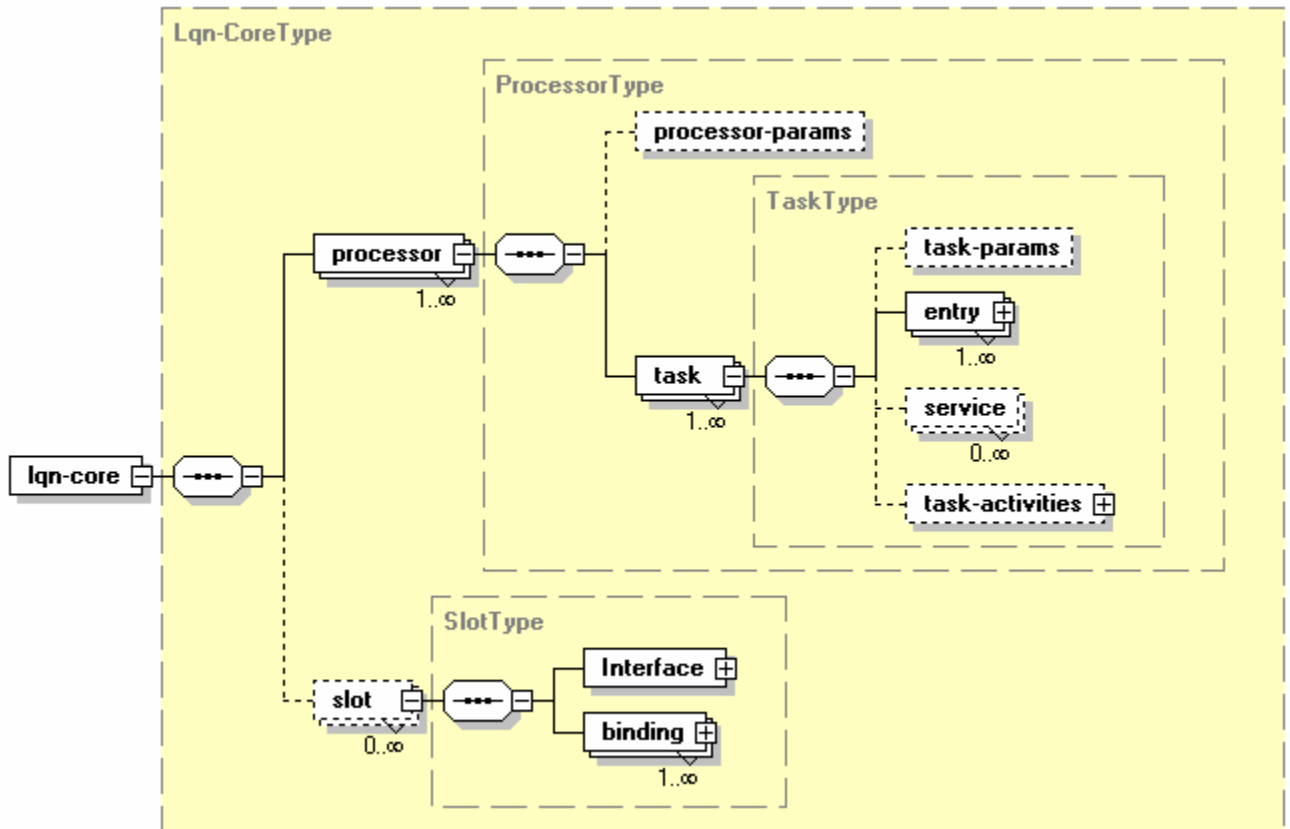


Figure 3-10 Elements within LQN Core

3.3.2.2.1 Processor in LQN Core

The processor definition encapsulates the structure of tasks which in turn contains the structure of entries.

A processor has an attribute of name which is the id of the processor. It has two top elements: processor-params and task.

The element processor-params has attributes that are used to specify the parameters that were defined in the previous LQN language. These attributes are named as multiplicity, speed-factor, scheduling, quantum and replication respectively. The meaning of these attributes has been described in section 2.3.3.

3.3.2.2.2 Task in LQN Core

The element task has four top elements: task-params, entry, service and task-activities.

1. **task-params.** It defines the parameters of a task which are the same as those in the previous LQN language. It has those attributes that are termed as mult for multiplicity, replication, scheduling, activity-graph, think-time and priority respectively. The attributes of mult and replication have default values of 1. The value of scheduling has the same meaning as for processor parameters except that ‘r’ refers to reference task and ‘n’ refers to non-reference task. The task is regarded as a non-reference task by default. The attribute of activity-graph indicates whether this task has activities other than phase activities. It has two values, “YES” or “NO”. Its value is considered as “NO” by default.
2. **entry.** An entry has three top elements: entry-params, forwarding, and entry-activities. These are illustrated in Figure 3-11 below.

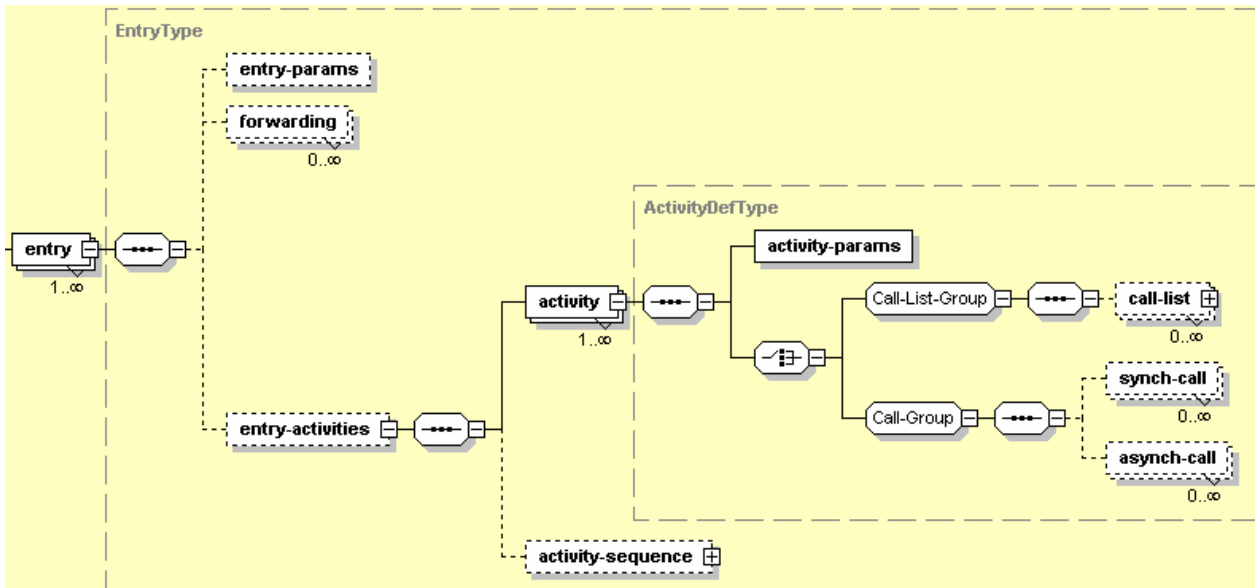


Figure 3-11 Elements defined within an entry

The element entry-params defines the parameters for an entry. These parameters are priority and open-arrival-rate.

The element `entry-activities` defines those activities that have no synchronization with other activities from other entries. The definition includes the activity's parameters, calls and activity sequence (if applicable). The element activity will be detailed in the next section.

The calls that an entry has made are classified as forwarding calls, synchronous calls or asynchronous calls. The attributes of synchronous and asynchronous calls will be introduced in the next section.

The element `forwarding` has such attributes as `dest` for destination and `probability` (of forwarding).

3. **service**. This has already been elaborated in section 3.3.2.1.4.
4. **task-activities**. Again, this has been elaborated in section 3.3.2.1.3.

3.3.2.2.3 Activities in LQN Core

An activity contains two top elements: `activity-params` and either `call-list` or `stochastic calls`.

Since the previous phases and activities are now unified as activities, the previous parameters of an entry such as mean host demands, coefficient of variation and think time are now defined by `activity-params` within the element activity. The call-order of an activity call is categorized as `STOCHASTIC`, `DETERMINISTIC` or `LIST`. This property is defined as an attribute named `call-order` of `activity-params`.

The element `synch-call` refers to the synchronous calls that an entry makes to its destination while `asynch-call` is for asynchronous calls. They both have such attributes as `dest` for destination, `calls-mean` for the mean number of calls, `calls-cvsq` for the coefficient variance of the mean number of calls, `fanin` and `fanout`. `Fanin` and `fanout` are special attributes of models with replication, which will not be discussed here.

The element call-list contains two elements synchronous-call and asynchronous-call both of which have three attributes. The attribute dest refers to the call destination, fanout for number of calls fanning out and fanin for number of fanin calls.

Activity-sequence describes the sequences of activities that all originated from the same entry. No joining with activities originated from other entries occurs in this case. The elements that are defined within the activity-sequence are shown in Figure 3-12.

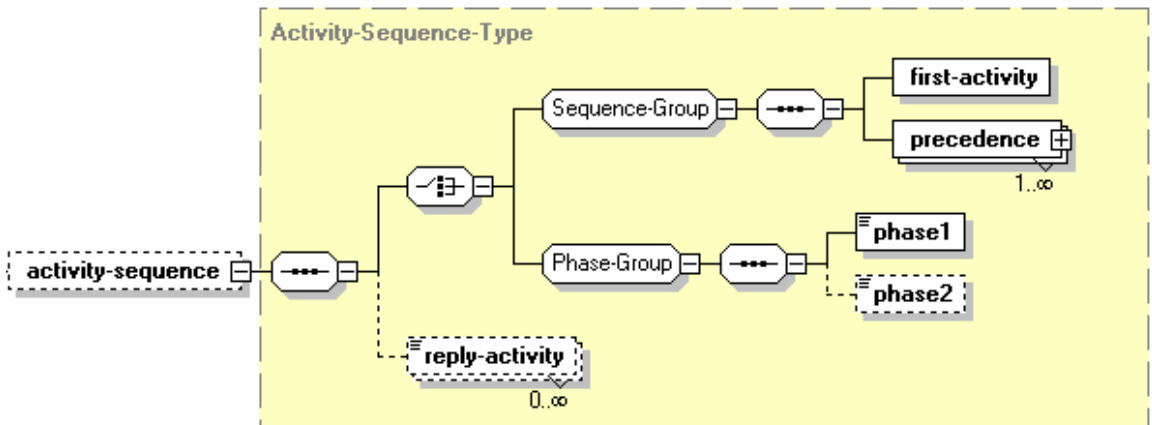


Figure 3-12 Elements within activity-sequence in an entry

The activity-sequence within an entry has two alternatives: phase activities or other kinds of activities. Phase activities are those executions in the two phases of an entry as defined in the previous LQN language. These activities are called phase1 and phase2. The other activity-sequence containing first-activity and precedence is exactly the same as the one defined in the task-activities which has been detailed in the previous section. The element reply-activity refers to the activity that replies back to the entry. For phase activities, phase1 is always the replying activity.

3.3.2.3 LQN Sub-model

An LQN sub-model definition consists of Interface, Parameter and all other elements defined in lqn-core. This section will focus on its Interface and Parameter parts.

1. Interface

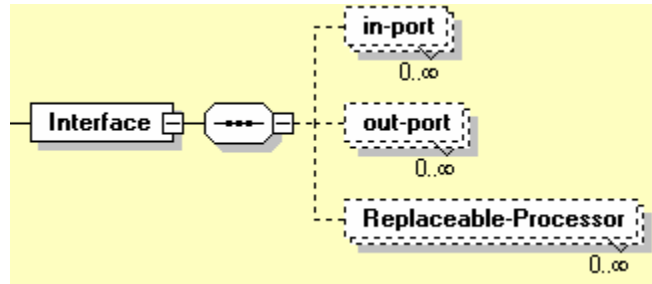


Figure 3-13 Elements within Interface

This is the section that exhibits information about the component sub-model to the outside. As shown in Figure 3-13, an interface contains three elements: in-port, out-port and Replaceable-Processor. For an in-port, it has the attributes of name, connect-to and description. The connect-to attribute specifies the destination entries in the component that this port is connected to. The attribute of description is for describing any other information associated with this port. For an out-port, it has the similar attributes except that the attribute of connect-to is replaced by connect-from. This attribute specifies the entries in the component that connect to this port. The element of Replaceable-Processor defines the processors in the component that could be substituted by outside processors in the outer component or system model.

2. Parameter

This part defines the parameters that a sub-model might have. These parameters are used to characterize its different workloads when it executes in different environments. The parameters include execution demands, service request parameters as well as configuration parameters such as threading levels. The parameters have default values. When the sub-model is instantiated in the outer model which will be elaborated later, the instantiation parameters can redefine these parameters such as execution demands and configuration parameters, etc. The parameter has only one top element named para. This element can present for an arbitrary number of times. It has two attributes:

name and default. The attribute name specifies the variable name which begins with a ‘\$’ sign. The attribute default specifies the default value.

The names of the variable parameters in the model begins with a dollar sign ‘\$’ followed by a string for the name. For instance, ‘\$multi’ is a variable parameter if it appears in the sub-model.

3.3.2.4 LQN Assembly Model

An LQN assembly model consists of run-control, plot-control, solver-params and all other elements in lqn-core. The elements of run-control and plot-control are used for experiment instrumentation purpose. They are used in generating Spex input file format which can be found in [15].

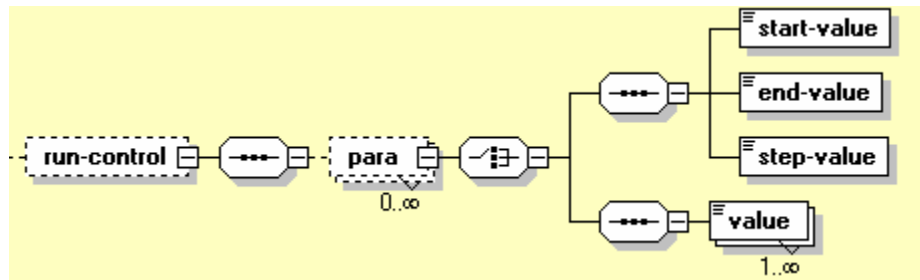


Figure 3-14 Elements within run-control

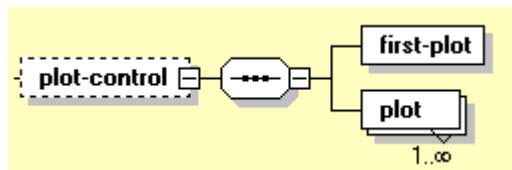


Figure 3-15 Elements within plot-control

The element run-control has two choices. The control values of the parameters can be either a loop or enumeration. The loop is in the format of for loop; it has start-value, end-value and the step-value.

The element plot-control is used to generate reported values for the specified parameters. It contains two elements. The element first-plot defines the parameter whose value can vary while element plot defines those parameters that need recording the values according to the particular value of first-plot.

The element solver-params is used to define the control parameters for LQNS or ParaSRVN, defined in the General information section which has been described in section 2.3.3. Those parameters are defined as attributes for solver-params; they are named as comment, conv_val for convergence value, it_limit for iteration limit, print_int for print interval, underrelax_coeff for coefficient of under relaxation.

The other elements in lqn-core have been described in the previous section 3.3.2.2.

3.4 An Example of Components Assembly

As mentioned before, both LQN sub-model and assembly model are capable of assembling components. The way that they compose sub-models is the same; the composition is accomplished by bindings. The bindings are defined within the slot. Thus, assembly is the process of defining the bindings. This section will describe an example that shows how to define a nested sub-model. This example was originated from the report [22].

A “flat” sub-model which is called SingleMod is shown in Figure 3-16 and the nested model which contains SingleMod called NestedMod is shown in Figure 3-17.

The XML files for these two sub-models are attached in Appendix B.1 and B.2

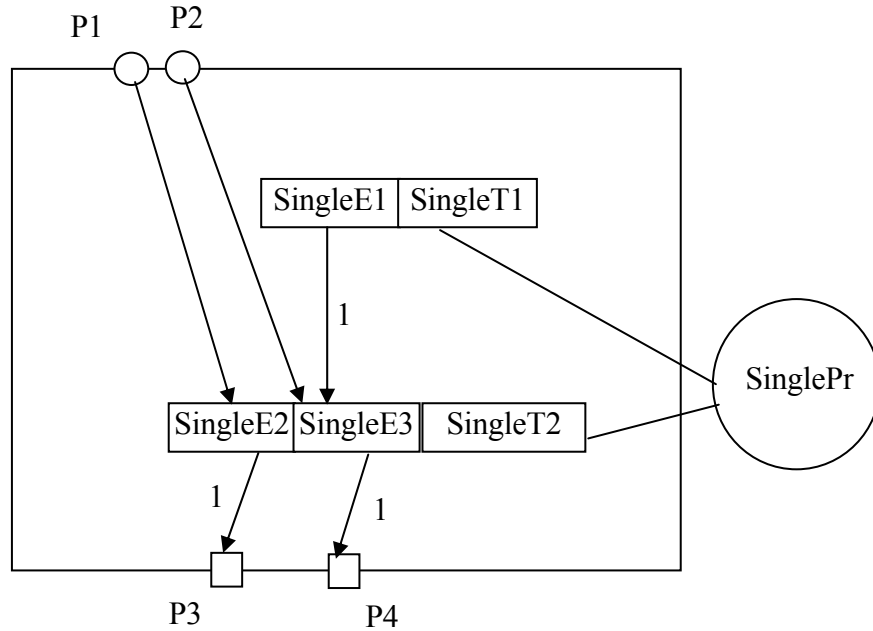


Figure 3-16 A “flat” LQN sub-model - SingleMod

In Figure 3-16, task SingleT1 represents a set of user tasks that are part of the component (e.g. administrator). It is a reference task that does not need to be called and generates workload for the component and the rest of the system. This sub-model has two incoming ports shown by circles and two outgoing ports shown by rectangles. The numbers on the message arrows indicate the mean number of calls that an entry makes to its destination. The connection from the incoming ports to the inside entries have no numbers attached. They are determined by the outside callers. The processor SinglePr is a replaceable processor.

The interface and parameter section for this sub-model is listed below while the full text can be found in Appendix B.1

List 3-1 Interface of the example SingleMod

<Interface>

<in-port name="p1" connect-to="SingleE2" description="read data from database"/>

```

<in-port name="p2" connect-to="SingleE3" description="update data to database"/>
<out-port name="p3" connect-from="SingleE2" description="read request to file
sever"/>
<out-port name="p4" connect-from="SingleE3" description="update request to file
server"/>
<Replaceable-Processor name="SinglePr"/>
</Interface>
<Parameter>
<para name="$SingleT2_mult " default="1"/>
<para name="$SingleE2_demand" default="1"/>
</Parameter>

```

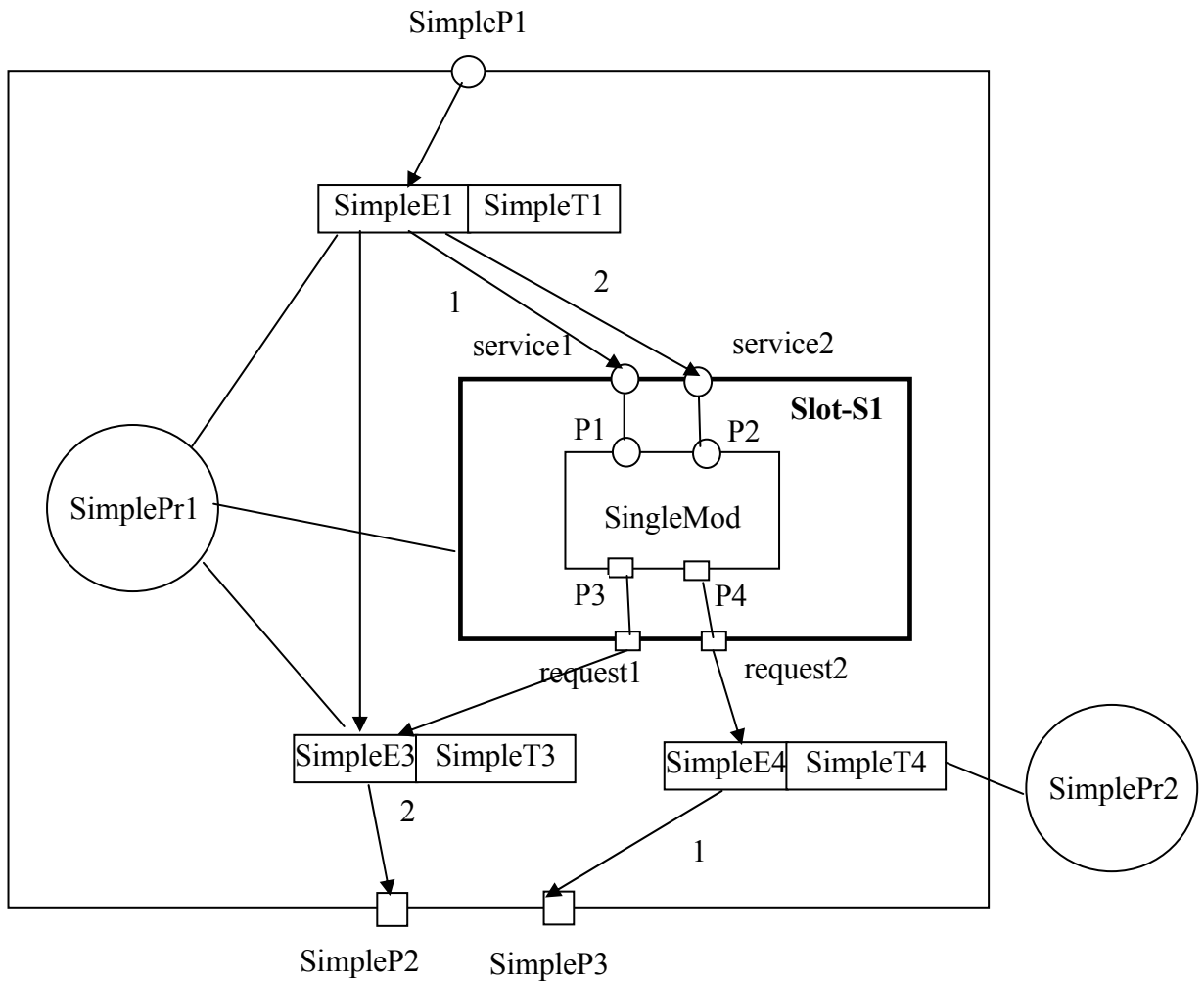


Figure 3-17 A nested LQN sub-model - NestedMod

In Figure 3-17, the thick box represents a slot defined in the sub-model which will plug sub-model SingleMod into NestedMod. This slot, which is named S1 has four ports which are connected to the interfaces of SingleMod as shown in the figure. These ports are named service1, service2, request1 and request2 respectively. The slot definition is listed below. The bold lines are the binding section. In this example, the processor SinglePr in the inner component SingleMod will be replaced by SimplePr1 in the sub-model of NestedMod. The tags that start with <!-- and end with --> are used for comments in XML file.

List 3-2 The Slot in NestedMod

```

<slot id="S1" bind-target="SingleMod">
  <Interface>
    <in-port name="service1" connect-from="SimpleE1"/>
    <in-port name="service2" connect-from="SimpleE1"/>
    <out-port name="request1" connect-to="SimpleE3"/>
    <out-port name="request2" connect-to="SimpleE4"/>
  </Interface>
  <binding>
    <!--parameter assignment here for SingleMod-->
    <parameter name="$SingleT2_mult" value="4"/>
    <!--the rest parameters are defined as parameters for NestedMod -->
    <processor-binding source="SinglePr" target="SimplePr1"/>
    <!--source refers to elements in the inner component -->
    <!--target refers to elements in the slot -->
    <port-binding source="p1" target="service1"/>
    <port-binding source="p2" target="service2"/>
    <port-binding source="p3" target="request1"/>
    <port-binding source="p4" target="request2"/>
  </binding>
</slot>

```

After binding, the resulting sub-model is shown in Figure 3-18 below.

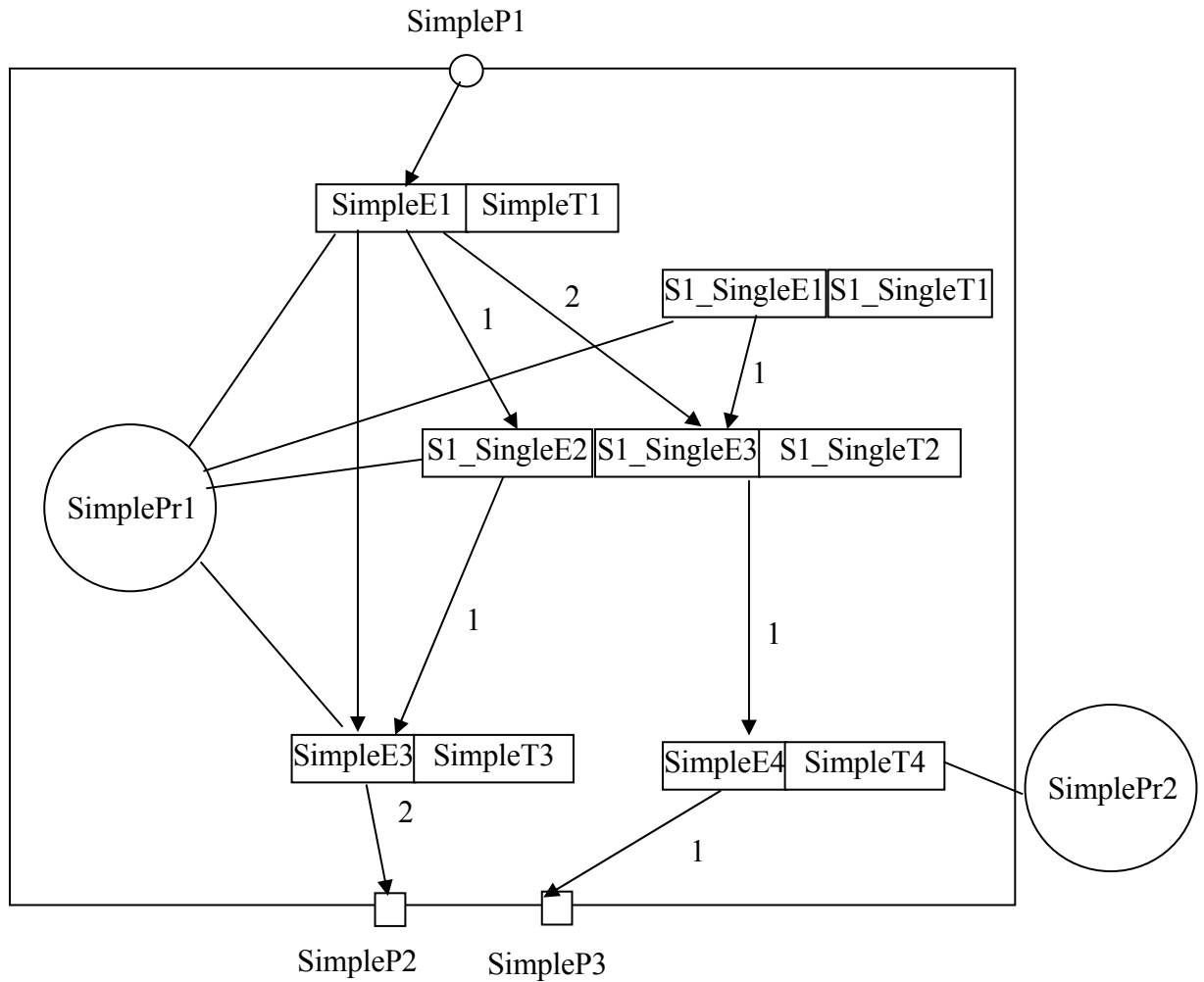


Figure 3-18 The Resulting Sub-model of NestedMod

As indicated in Figure 3-18, the slot and the interfaces have now disappeared. The names of the tasks in SingleMod are now prefixed by the slot name S1. They are now executed in processor SimplePr1 of the NestedMod. The interactions between the tasks are now overwritten by the specified values in sub-model SingleMod. For instance, now the mean number of calls from S1_SingleE3 to SimpleE4 is 1, which is specified in SingleMod from SimpleE3 to its outgoing port P4 that directs the calls to SimpleE4.

The resulting model of LQN assembly model that contains sub-models is very similar to those out of LQN sub-models except that there are no interfaces in the resulting model; it has run control and other solver parameters.

This assembly process has been automated by a tool LQComposer which will be introduced in detail in Chapter 4.

3.5 Approaches to Creating LQN Component Models

An LQN component that corresponds to a software component or subsystem can be derived in several ways. Basically, the approach to derive an LQN component model or an LQN model is the same. It can be derived from analysis of scenarios, or design specifications [27] [28], or from traces [16], or from requirement analysis [26].

An example of deriving LQN models from scenario analysis is shown in Figure 3-19. From the UML sequence diagram on the left, an LQN model can be derived as shown on the right.

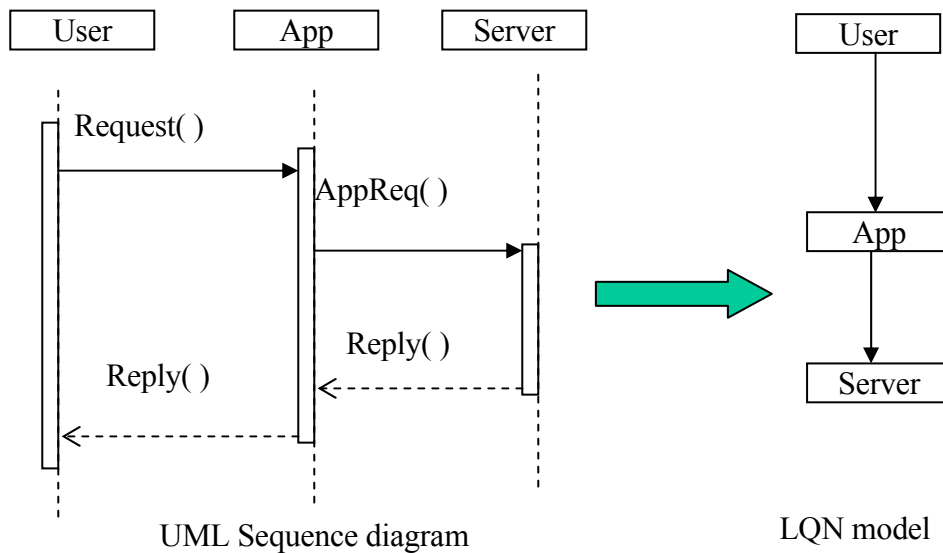


Figure 3-19 Deriving LQN Models from Scenario Analysis

The execution demands can be obtained from experience, or from budgeted values [32]. The performance attributes of the software component can be verified using a testbed

which provides drivers, load generator and operational stubs as described in the Layered System Generator (LSG) [38].

Chapter 4 An Approach to Component Based Performance Modeling

This chapter first gives some brief overview of the design issues about the component based performance modeling. Then it presents the present approach and describes the process and tools for assembling component sub-models into a system model, as on the right side of Figure 1-1. The assembly model specifies how to compose the sub-models for software components, and their parameters. The assembly can occur at both system-level model and sub-model as mentioned in Chapter 3. The prediction of the planned system is achieved by solving the system-level model. The results from these models can be used as feedback to the software architect or designers. This chapter will focus on system-level assembly.

4.1 Design Issues

For performance modeling of component based software systems, one difficulty is that components may run in different environments. The need to “describe” performance attributes of the software components incorporating its runtime environments presents a big challenge to performance prediction of the component based systems. This thesis work has considered this issue by using parameterized performance sub-models. The parameterization reflects the different behavior of the component in different runtime environments. Meanwhile, software components are configurable and this is described by sub-model parameters and replaceable processors. This thesis proposes an XML based language to describe the performance sub-models for software components. The system model is obtained by composing the sub-models. The language has been designed to enable sub-model composition. It also takes into account the fact that a software component may be built nestedly. Therefore, the language supports nested definition in the component sub-models. It also specifies how the component sub-models can be bound to different servers and processors.

The XML language is suitable to describe structured data. For LQN models, this research has taken a different view, containment relationships between processors, tasks, entries and activities as described in section 3.3 in Chapter 3. This containment relationship

better describes the structured data in LQN models compared to the previous LQN language. It treats the previous activities and phases as both activities in entries which eliminates the confusion between activities and phases.

The sub-model parameters can be passed through outer sub-model to the inner sub-model.

4.2 Overview of the Present Approach

The present approach is illustrated in the following diagram Figure 4-1. From the documentation of the planned system, such as product specification, software architecture or any high-level designs, the performance sub-models are selected from the library. Based on these, a system assembly model is derived which consists of the customization and binding information of the sub-models. This assembly model along with the appropriate information of the sub-models is then input to the tool LQComposer. This tool generates the normal LQN models which can be solved by existing solvers such as LQNS and ParaSRVN. The results from LQN models can then be fed into the design documentation. Therefore, the design can be evaluated.

In this work, a tool called LQComposer has been developed to automate the process of sub-models assembly. This tool can generate system-level models from sub-models automatically.

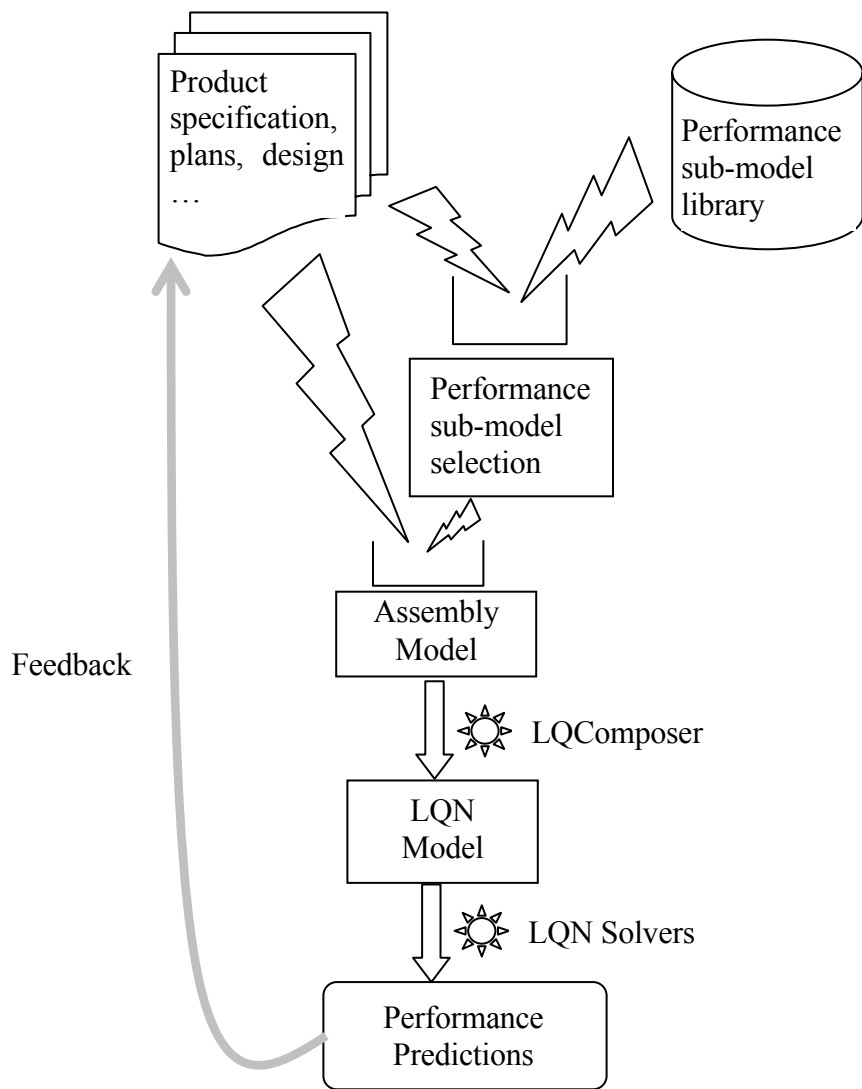


Figure 4-1 The Present Approach to Component Based Performance Prediction

4.3 LQN Assembly Model

This assembly model defines the system-level model as an assembly of components plus “glue” that defines the context and behavior. It adopts XML representation of LQN models that describes the LQN elements such as tasks, entries and activities. It differs from a normal LQN model in that it has slots that identify which components can be “glued”.

4.3.1 An Example of an LQN Assembly Model

The system assembly model can be derived from the software architecture of the planned system. An example of this is shown in Figure 4-2 which is an LQN assembly model for a three-tier E-business application system.

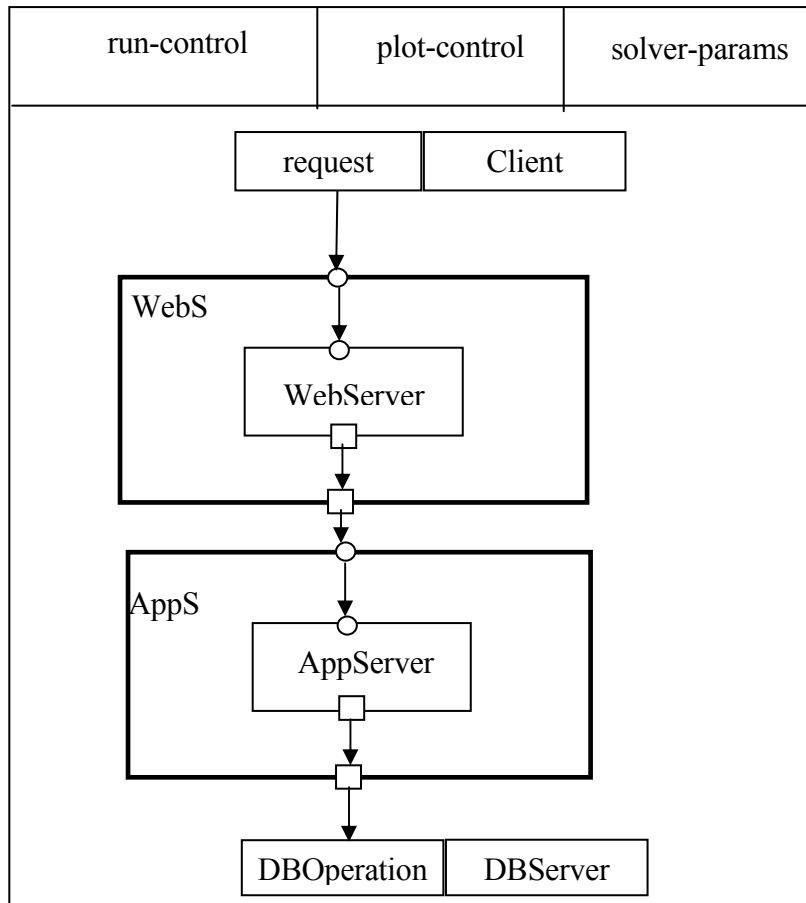


Figure 4-2 An Assembly Model for an E-Business Application System

In this model, clients send requests to the web server which does some computing and then makes requests to the application server. The application server does some processing and accesses some operations from the database server. The web server and the application server are modeled as replaceable subsystems. They will be substituted with the suitable sub-models from a component model library. In this assembly model, the two boxes with thick lines represent slots. Slot WebS is the placeholder for the web server component and AppS is for the application server component. These component models could have very

complex internals and may have nested component models inside. The bindings that will glue the components are described within the slots. As mentioned in Chapter 3, the bindings contain the information about how the interfaces of the slots are connected to the interfaces of the components that will be plugged in. The circles in this diagram represent the incoming interfaces of the slots and the components. The squares stand for the outgoing interfaces. The client and database server in this assembly model are modeled as one single task although they may be sub-models too. This assembly model also contains a section of runs control parameters which are divided into run-control, plot-control and solver-params that have been described in Chapter 3.

4.3.2 The Reusability and Adaptability of the Assembly Model

Having an assembly model is useful in the case of developing similar products as is the case in product lines. A software product line is a set of software intensive systems that shares a set of common features and satisfies the specific needs of a certain market segment or mission. These software systems are developed from a set of core assets in a prescribed way [6]. The core assets include the software architecture, the reusable software components, documentation of requirement analysis, specifications, performance model, test plans and etc. Software product lines are a form of component-based development. Each product is generated by selecting applicable components from the base of the core assets and these components are then tailored according to the specific needs through the mechanisms such as parameterization or inheritance. The assembly model of the product is a performance model in this case. By taking appropriate parameterized component sub-models, the particular system model for one product can be then generated. Since the architecture of the product line is reusable and configurable, the assembly model derived from the architecture is therefore reusable for the family of the products.

On the other hand, the assembly model is adaptable. Figure 4-2 shows an example of assembly model for an E-business application. In this system, the application server has only one outgoing interface which directs requests to the database server. In the case that the application server needs to have two or more outgoing interfaces, this model can be

adapted by making a little change to the slot AppS and the bindings for AppS. Meanwhile, the database server task can serve two or more requests by having two or more entries.

4.4 Tool – LQComposer

This section describes the design and the implementation of the tool LQComposer which assembles sub-models to system-level models. The resulting model is in the previous text format LQN language and ready for LQN solvers to solve. This tool is also capable of expanding a sub-model which contains nested sub-models to a “flat” sub-model. But the resulting model is still in XML language and contains elements that are specific for a sub-model (e.g. interfaces).

The workflow of this tool is shown in the following picture Figure 4-3.

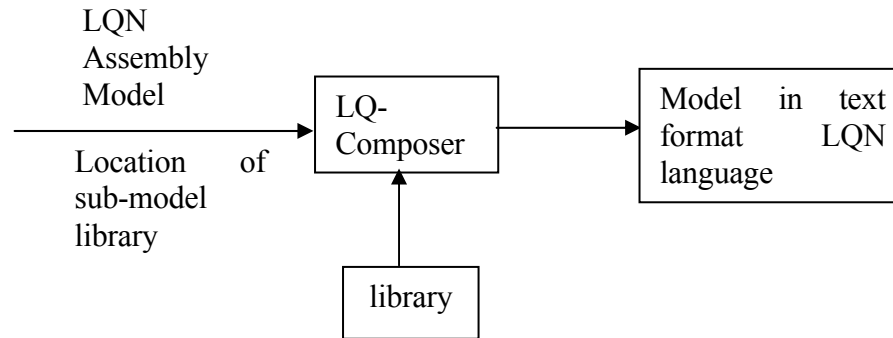


Figure 4-3 Workflow of LQComposer

The input to this tool is the LQN assembly model and the location of the sub-model library. The output is a model in the previous LQN language. This tool will generate task instances and their parameters guided by the binding sections in the assembly model. After running this tool, the interfaces of each component sub-model will disappear in the system model. And the connections of components will be overwritten by the actual interactions that are specified by parameterization or by the sub-model definition itself.

4.4.1 Overview of Tool Design

The representation of component models and assembly models are now all in an XML language, but the model solvers that exist in RADS lab such as LQNS and ParaSRVN only take the previous text format LQN language. Therefore, the transformation must be done between the XML language and the previous LQN language. This is divided into two steps. The first step is to transform the assembly model which contains slots and sub-model interfaces to a “flat” model that has no interfaces or slots but is still in XML language. This output XML document is still valid against the LQML schema. The second step is to transform this “flat” model in XML language to a model in the previous LQN language. The first step is accomplished by taking advantage of JAXP (Java API for XML Processing) and DOM (Document Object Model) which will be introduced in the next section. The second step is done through XSLT transformations. The first one is a Java program named LQNAssemble. The second one is a XSLT stylesheet named xml2LQN. These two programs form the tool LQComposer. The LQNAssemble can work for either system-level assembly or a sub-model level assembly. However, xml2LQN can only work for a system level model that has no interfaces and no undefined components. This is because there are no equivalent elements for sub-model interface or variable parameters in the previous LQN language.

These two programs can be invoked separately. The command to invoke the first program is as follows (which is a batch command):

```
LQNAssemble [-lib <the library directory>] <assembly file>
```

The command to invoke the second one is as follows (which is also a batch command that invokes the XSLT engine and xml2LQN stylesheet):

```
xml2LQN <XML model> [output model]
```

4.4.2 LQNAssemble Design

This Java program transforms an LQN assembly model in XML language, which contains slots which embed component models, to a “flat” LQN model that has not only removed the interfaces and slots but also incorporated elements in the body of component models and directed calls to the appropriate destination. The output model is still in XML representation. The transformation is illustrated in section 3.4 in Chapter 3 as an example of component assembly in which a sub-model contains a nested sub-model and the result model is a flat sub-model. The input to this program is an assembly model in XML language and an optional directory for searching for sub-models. The output is the flattened XML model named as the assembly model file suffixed by “_flt”.

This program is developed using Java API (application programming interface) for XML Processing (JAXP) and the Document Object Model (DOM). The DOM is an API that defines the logical structure of objects contained in the document and the way the document is accessed and manipulated. Thus, it enables programmers to add, remove or change the elements or create a document. In DOM, a document has a logical structure that is very much like a hierarchical tree which consists of nodes that can be manipulated. The DOM level 2 core specification has a Java language binding. The JAXP provides an API specifically for processing XML documents. Its reference implementation uses Crimson which is available from <http://xml.apache.org/crimson/> as its default XML parser and Xalan (available from <http://xml.apache.org/xalan-j/index.html>) as its default XSLT engine. These are the implementations of the specified APIs. By using these two APIs, an XML document can be parsed and transformed.

The algorithm of this program is as follows.

1. Read in an XML assembly file from the command line and the optional library directory.
2. Parse this XML file into a DOM document.
3. Check if the document contains any elements of type “slot”.

- 3.1 If it does not, exit the program.
- 3.2 If it does, record all slots' information into a Java vector in which each element represents one slot. (A Java class of Slot is developed to record the attributes and elements contained in a slot.)
- 3.3 For each slot that is an element in a Java vector, repeat the following.
 - 3.3.1 Locate the XML file that contains the sub-model. Parse this file into another document. Obtain the slot id which will be the prefix for the names of processors, tasks and entries in the sub-model. Obtain the replaceable processors in the sub-model.
 - 3.3.2 Rename each entry in the sub-model by prefixing the slot id.
 - 3.3.3 Rename each task in the sub-model by prefixing the slot id.
 - 3.3.4 Instantiate parameters for the sub-model. Locate the elements whose attributes' values start with '\$'s. For each located parameter, substitute its value with the specified value that is stored in the object Slot recorded in the vector in step 3.2.
 - 3.3.5 Direct all calls in the sub-model to the proper destination. If the original call destination is within the sub-model then prefix the destination by the slot id. Otherwise, the calls should be directed to the system model (or the outer model). This is done by checking the out-port that is bound to a slot. Check the attributes "connect-to" of the slot which record where the calls should be directed. Direct the calls to the proper destination in the system model.
 - 3.3.6 Direct calls made in the system model that are connected to the in-port of the slot. The destination should be within the sub-model. Direct the calls to the proper destination by checking the attributes

of the slot and the attributes of the in-port of the sub-model. The destination should be prefixed by the slot id.

3.3.7 Obtain all the processors in the sub-model. Rename those that are not replaceable by prefixing the slot id. For each non-replaceable processor, do the following.

3.3.7.1 Make a deep clone of this processor node. (The deep clone will clone all of its descendants too.) Import this cloned processor node to the assembly document. Make this cloned node as one of the children of the root node in the assembly document. Remove this processor node along with its descendants from the sub-model document.

3.3.8 For each replaceable processor, do the following.

3.3.8.1 Obtain the processor name in the system model that will replace this sub-model processor. This can be achieved by checking the binding elements that are recorded in the slot.

3.3.8.2 If this replaceable processor has children, for each child, do the following.

3.3.8.2.1 If the child is not named as “processor-params”, make a deep clone of this child node. Import this cloned child to the assembly document. Make it as a child of the target processor that will replace the sub-model processor.

3.3.8.3 Remove this replaceable processor node along with its descendants from the sub-model document.

3.4 Remove each slot node along with its descendants from the assembly model.

4.4.3 xml2LQN Design

This is a XSLT stylesheet that transforms a “flat” LQN model in XML language to the previous LQN language which is in plain text format. The details of the previous language have been described in section 2.3.3. It includes five sections: General information section which is denoted by a letter ‘G’, Processor section which is denoted by a letter ‘P’, Task section which is denoted by a letter ‘T’, Entry section which is denoted by a letter ‘E’ and possibly several Activity sections which are denoted by a letter ‘A’. Activity sections only appear when tasks have activities. There is one activity section for each task that has activities executions. An example of an LQN model in the previous language is listed below. This example has no activities. The # tag denotes comments which are used to summarize the syntax. These comments are excerpted from the documentation which is available online from <http://www.sce.carleton.ca/rads/lqn/lqn-documentation/reserv-templ.lqn>. The figure 4-5 is the corresponding graphical representation.

List 4-1 An LQN model in previous language

```

G      #General information section
"This is a test case" #comments of the model
0.00001      #convergence criterion
50           #iteration limit
1           #print interval
0.8         #under-relaxation
-1          #end of General section
# Processor Information
#(the zero is necessary;
#it may also give the number of processors)
P 0
#SYNTAX:
# p ProcessorName SchedDiscipline [multiplicity, default = 1]

```

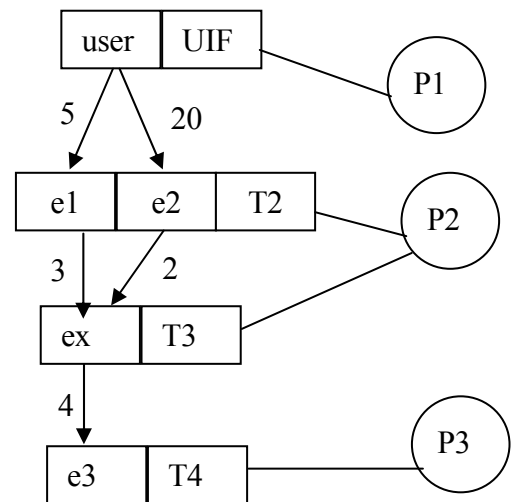


Figure 4-5 The LQN graphical model

```

#SchedDiscipline = f fifo|r random|p preemptive|h hol or non-pre-empt|s proc-sharing
#multiplicity = m value (multiprocessor)|i (infinite)
p P1 f          #Processor P1 (FIFO)
p P2 f          #Processor P2 (FIFO)
p P3 f          #Processor P3 (FIFO)
-1             #end of Processor section
# Task Information: (the zero is necessary; it may also give the number of tasks)
T 0
#SYNTAX: t TaskName RefFlag EntryList -1 ProcessorName [multiplicity]
#TaskName is any string, globally unique among tasks
#RefFlag = r (reference or user task)|n (other)
#multiplicity = m value (multithreaded)|i (infinite)
t UIF r user -1 P1 m 10
t T2 n e1 e2 -1 P2
t T3 n ex -1 P2
t T4 n e3 -1 P3 m 10
-1
#Entry Information: (the zero is necessary; it may also give the total number of entries)
E 0
# SYNTAX-FORM-A: Token EntryName Value1 [Value2] [Value3] -1
#           EntryName is a string, globally unique over all entries
#           Values are for phase 1 and 2 (phase 1 is before the reply)
#           Tokens indicate the significance of the Value:
#           s HostServiceDemand for EntryName
#           c HostServiceCoefficientofVariation
#           f PhaseTypeFlag
# SYNTAX-FORM-B: Token FromEntry ToEntry Value1 [Value2] [Value3] -1
#           Tokens indicate the Value Definitions:
#           y SynchronousCalls (no. of rendezvous)
#           F ProbForwarding (forward to ToEntry rather than replying)
#           z AsynchronousCalls (no. of send-no-reply messages)

```

```

#           o Fanout (for replicated servers)
#           i FanIn (for replicated servers)
# This example only shows use of host demands and synchronous requests
s user 1.0 -1
y user e1 5 -1
y user e2 20 -1
s e1 0.04 -1
y e1 ex 3 -1
s e2 0.2 -1
y e2 ex 2 -1
s ex 0.9 -1
y ex e3 4 -1
s e3 0.05 -1
-1

```

An XML language for this model is listed below in List 4-2.

List 4-2 An XML representation for the LQN model in List 4-1

```

<lqn-model name="test" description="a LQN model in xml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn.xsd">
  <solver-params comment="This is a test case" conv_val="0.00001"
it_limit="50" print_int="1" underrelax_coeff="0.8"/>
  <processor name="P1">
    <processor-params multiplicity="1" replication="1"/>
    <task name="UIF">
      <task-params mult="10" scheduling="ref"/>
      <entry name="user">
        <entry-activities>
          <activity name="user_ph1">
            <activity-params host-demand-mean="1.0"/>
            <synch-call dest="e1" calls-mean="5"/>
            <synch-call dest="e2" calls-mean="20"/>
          </activity>

```

```

    <activity-sequence type="PH1PH2">
      <phase1>user_ph1</phase1>
    </activity-sequence>
  </entry-activities>
</entry>
</task>
</processor>
<processor name="P2">
  <processor-params multiplicity="1" scheduling="fcfs" replication="1"/>
  <task name="T2">
    <task-params mult="1" activity-graph="NO"/>
    <entry name="e1">
      <entry-activities>
        <activity name="e1_ph1">
          <activity-params host-demand-mean="0.04"/>
          <synch-call dest="ex" calls-mean="3"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>e1_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
    <entry name="e2">
      <entry-activities>
        <activity name="e2_ph1">
          <activity-params host-demand-mean="0.2"/>
          <synch-call dest="ex" calls-mean="2"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>e2_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
  <task name="T3">
    <task-params mult="1" activity-graph="NO"/>
    <entry name="ex">

```

```

    <entry-activities>
      <activity name="ex_ph1">
        <activity-params host-demand-mean="0.9"/>
        <synch-call dest="e3" calls-mean="4"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>ex_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
</processor>
<processor name="P3">
  <processor-params multiplicity="1" scheduling="fcfs" replication="1"/>
  <task name="T4">
    <task-params mult="10" activity-graph="NO"/>
    <entry name="e3">
      <entry-activities>
        <activity name="e3_ph1">
          <activity-params host-demand-mean="0.05"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>e3_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
</processor>
</lqn-model>

```

The model in XML language has the structure as detailed in Chapter 3. The stylesheet `xml2LQN.xsl` will take the XML document listed in List 4-2 as input, and transform it to the file as listed in List 4-1. The main flow of transformation is explained in Figure 4-6. The transformation involves six templates and five of them will generate the five sections in the previous text format LQN language. There is a main template that controls the order of invoking the five other templates that generate the elements for each section. For

generating each section except the activity section, it works in the following order: outputting starting tags (e.g. ‘G’ for general information section), invoking the particular template that will generate the particular section, and outputting the ending tags (‘-1’) for the section. The procedure of generating the activity sections is exceptional in that not every LQN model has activity executions. The ActivityTemplate will check if there are any activities involved in the model. Depending on this, it may or may not generate an activity section for the particular task. The transformation algorithm in each template will be described in the next section.

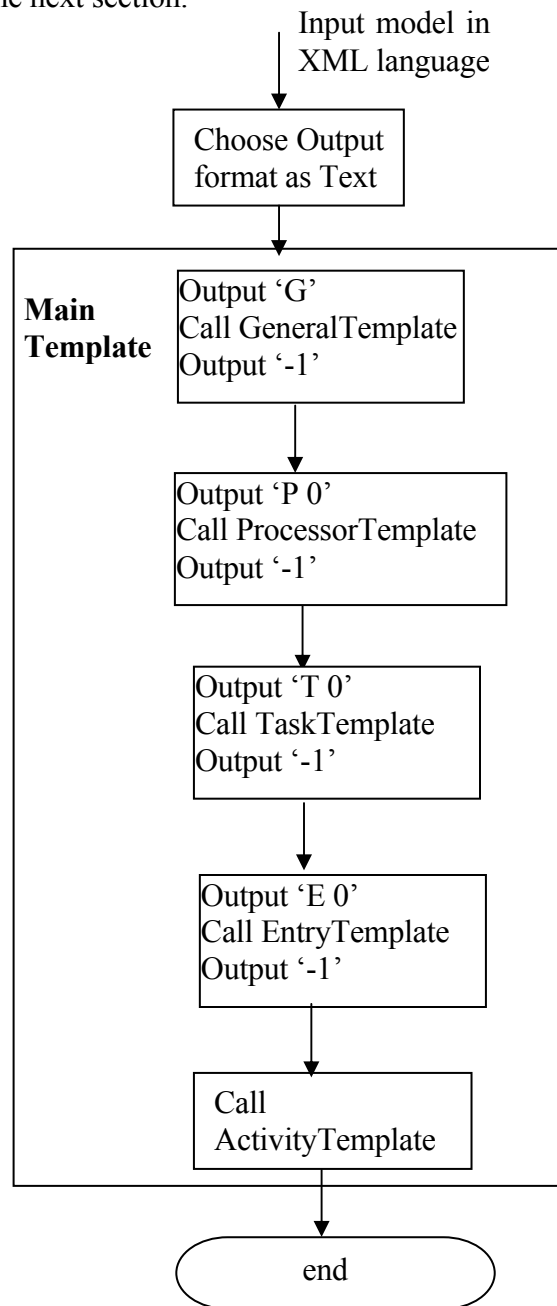


Figure 4-6 The workflow of xml2LQN

4.4.3.1 Template Algorithm in xml2LQN Stylesheet

In the descriptions of the transformation rules below, a space character is used to separate the adjacent elements in one line of the output.

- **GeneralTemplate.** This template generates the General information section. It checks the attribute node of /lqn-model/solver-params which is supposed to contain all the solver parameters that are presented in the General information section in the previous LQN language. These parameters are now given as attributes of the element solver-params which is the one of the top elements of lqn-model. The transformation algorithm is as follows.

- 1 Locate the node of /lqn-model/solver-params/.
- 2 Output a double quotation mark (“”).
- 3 Output the value of attribute comment.
- 4 Output another double quotation mark (“”).
- 5 Output the value of attribute conv_val plus a new line symbol.
- 6 Output the value of attribute it_limit plus a new line symbol.
- 7 Output the value of attribute print_int plus a new line symbol.
- 8 Output the value of attribute underrelax_coeff plus a new line symbol.

- **ProcessorTemplate.** This template generates the previous Processor information section. The transformation algorithm is as follows.

For each processor node

- 1 Retrieve the value of the attribute name. Output ‘p’ plus the name.
- 2 Retrieve the value of scheduling policy.

2.1 If the value equals 'fcs' then output 'f'.

2.2 If the value equals 'ps' then output 's'.

2.3 If the value equals 'pp' then output 'p'.

2.4 If the value equals 'r' then output 'r'.

2.5 If the value equals 'h' then output 'h'.

2.6 If none of these is satisfied, output 'f'.

3 Retrieve the value of quantum. If it is greater than zero, then output that value.

4 Retrieve the value of multiplicity.

4.1 If the value is greater than 1, then output 'm' plus the value.

4.2 If the value is 'i', then output 'i'.

5 Retrieve the value of replication. If the value is greater than 1, then output 'r' plus the value.

6 Retrieve the value of Speed-factor. If it is greater than 0, then output 'R' plus the value.

7 Output a new line symbol.

- **TaskTemplate.** This template generates task information section. The transformation algorithm is as follows.

For each task node

1 Retrieve the value of attribute name. Output 't' plus the name.

2 Retrieve the value of attribute scheduling of element task-params.

2.1 If the value equals 'ref', output 'r'.

2.2 If the value equals 'fcfs', output 'f'.

2.3 Otherwise, output that value. Output 'n' by default.

3 For each entry inside this node,

3.1 Retrieve the value of attribute name. Output the name.

4 Output '-1'

5 Retrieve the name of the processor which is the parent node of the task. Output the name.

6 Retrieve the value of attribute priority. If it is greater than zero, output that value.

7 Retrieve the value of attribute think-time. If it is greater than zero, output 'z' plus that value.

8 Retrieve the values of attributes of task-params which is the child node of task.

8.1 Retrieve the value of attribute mult.

8.1.1 If it is greater than 1, output 'm' plus the value.

8.1.2 If it equals 'i', output 'i'.

8.2 Retrieve the value of attribute replication. If it is greater than 1, output 'r' plus the value.

9 Output a new line symbol.

- **EntryTemplate.** This template generates the Entry information section. Its transformation algorithm is as follows.

For each Entry node

- 1 Check the node of entry-params which is the child of entry node.
 - 1.1 Retrieve the value of attribute open-arrival-rate. If it is greater than 0, then retrieve the name of the entry and output 'a' plus the name plus the arrival rate.
 - 1.2 Retrieve the value of attribute priority. If it is greater than 0, then retrieve the name of the entry and output 'P' plus the name plus the priority value.
- 2 For each forwarding node which is the child of entry (if it has forwarding calls), retrieve the destination and the probability. Output 'F' plus the entry name plus the destination plus the probability and '-1'.
- 3 Locate the node of entry-activities which is the child of entry.
 - 3.1 Retrieve the value of attribute type of element activity-sequence which is the child of entry-activities node. If the value equals 'PH1PH2',
 - 3.1.1 Retrieve service demands for both phases, output 's' plus the entry name plus the demands plus '-1'.
 - 3.1.2 Retrieve think time of the entry. If it is greater than 0, output 'Z' plus entry name plus the think time plus '-1'.
 - 3.1.3 Retrieve the host demand coefficient of variation, if it is greater than 0, output 'c' plus the entry name plus the value plus '-1'.
 - 3.1.4 Retrieve the max-service-time, if it is greater than 0, output 'M' plus the entry name plus the value plus '-1'.
 - 3.1.5 Retrieve call-order, if it equals 'DETERMINISTIC', output 'f' plus the entry name plus '1' for deterministic phase plus '-1'.

3.2 For each node of synch-call which is the child of activity node that is the child of entry-activities node.

3.2.1 Retrieve the value of call destination and mean number of calls, output 'y' plus the entry name plus the destination plus the mean number of calls plus '-1'.

3.2.2 Retrieve the value of fanout, if it is greater than 1, output 'o' plus the entry name plus the destination plus the value plus '-1'.

3.2.3 Retrieve the value of fanin, if it is greater than 1, output 'i' plus the entry name plus the destination plus the value plus '-1'.

3.3 For each node of asynch-call which is the child of activity node that is the child of entry-activities node.

3.3.1 Retrieve the value of call destination and mean number of calls, output 'z' plus the entry name plus the destination plus the mean number of calls plus '-1'.

3.3.2 Retrieve the value of fanout, if it is greater than 1, output 'o' plus the entry name plus the destination plus the value plus '-1'.

3.3.3 Retrieve the value of fanin, if it is greater than 1, output 'i' plus the entry name plus the destination plus the value plus '-1'.

3.4 Retrieve the value of attribute type of element activity-sequence which is the child of entry-activities node. If the value equals 'GRAPH', retrieve the name of the first activity and output 'A' plus the entry name plus the activity name.

4 Output a new line symbol.

5 For each node of /lqn-model/processor/task/task-activities/reply-entry (if they do exist, which refers to the case where activities from different entries in a task synchronize at some point).

5.1 Retrieve the entry name of reply-entry. Output 'A' plus the entry name plus the name of first-activity which is the child of reply-entry.

5.2 Output a new line symbol.

- **ActivityTemplate.** This template generates one activity section for each task. The transformation algorithm is as follows.

For each task, retrieve the value of attribute Activity-Graph. If the value equals 'YES'.

1 Output 'A' plus the task name

2 For each activity node which is the child of entry-activities node that is the child of entry,

2.1 Retrieve the value of attribute call-order of element activity-params which is the child of activity node. If it equals 'DETERMINISTIC', output 'f' plus the activity name plus '1'.

2.2 Retrieve the value of host-demand-mean of activity-params, output 's' plus the activity name plus the value.

2.3 Retrieve the value of host-demand-cvsq of activity-params, output 'c' plus the activity name plus the value.

2.4 For each synch-call which is the child of activity node, retrieve the value of the attributes calls-mean and dest, then output 'y' plus the activity name plus the destination name plus the value of calls-mean.

- 2.5 For each asynch-call which is the child of activity node, retrieve the value of the attributes calls-mean and dest, then output 'z' plus the activity name plus the destination name plus the value of calls-mean.
- 3 For each precedence that is the child of activity-sequence that is the child of entry-activities node,
 - 3.1 If it has child pre or pre-Loop, output the activity name plus '->'.
 - 3.2 If it has child pre-AND,
 - 3.2.1 For each activity that is the child of pre-AND,
 - 3.2.1.1 If current node is not the last child of pre-AND, output the activity name plus '&'
 - 3.2.1.2 If current node is the last one, output the activity name plus '->'.
 - 3.3 If it has the child of pre-OR,
 - 3.3.1 For each activity that is the child of pre-OR,
 - 3.3.1.1 If current node is not the last child of pre-OR, output the activity name plus '+'
 - 3.3.1.2 If current node is the last one, output the activity name plus '->'.
 - 3.4 If it has the child of post, output the activity name plus ';'.
 - 3.5 If it has the child of post-AND,
 - 3.5.1 For each activity that is the child of post-AND,
 - 3.5.1.1 If current node is not the last child of post-AND, output the activity name plus '&'.

- 3.5.2 If current node is the last one, output the activity name plus ‘;’.
- 3.6 If it has the child of post-OR,
 - 3.6.1 For each activity that is the child of post-OR,
 - 3.6.1.1 If current node is not the last child of post-OR, retrieve the value of prob, then output ‘(‘ plus the value plus the ‘)’ plus the activity name plus ‘+’.
 - 3.6.1.2 If current node is not the last child of post-OR, retrieve the value of prob, then output ‘(‘ plus the value plus the ‘)’ plus the activity name plus ‘;’.
- 3.7 If it has the child of post-LOOP, retrieve the value of loop count, the value of loop head and the value of the ending activity. Output the loop count plus ‘*’ plus the loop head plus ‘,’ plus the ending activity plus ‘;’
- 4 Output a new line symbol.
- 5 For each reply-activity which is the child of activity-sequence which is the child of entry-activities node,
 - 5.1 Retrieve the reply-activity, output the reply-activity plus ‘[‘ plus the entry name plus ‘]’.
 - 5.2 If current node is the last child, output ‘-1’. Otherwise, output ‘;’.
- 6 If the node of task-activities exists,
 - 6.1 For each activity that is the child of task-activities, obtain the parameters and calls similar to the steps 1.1.2.1 to 1.1.2.5 listed above. Obtain the precedence relationships among activities similar to the steps 1.1.3.1 to 1.1.3.7 listed above.

6.2 Output each reply activity that is the child of reply-entry which is the child of task-activities. This is similar to the steps 1.3.1 and 1.3.2 listed above.

4.5 Tool Validation

The validation of the tool LQComposer was again divided into two steps: validation against LQNASsemble and xml2LQN.

4.5.1 Validation of LQNASsemble

The validation of LQNASsemble was done against two cases. One case is that a system-level assembly model is taken as input. The other case is that a subsystem-level assembly model is taken as input. In the first case, the output should be a system model in XML language and ready for xml2LQN to transform into LQN language. In the second case, the output is still a component model but with the inner components embedded and extended. In both cases, the slots will disappear as well as the interfaces of the sub-models that are bound to the slot.

The first case adopts a system model that is very similar to Figure 3-17 except that there are no interfaces of the outer model and the task SimpleT1 is now a user task that generates load to the system. The sub-model is the same as shown in Figure 3-16 with its interface listed in List 3-1. The XML document for this assembly model named SimpleAbl.xml is listed in List 4-3 below. The slot and bindings are in bold font.

List 4-3 SimpleAbl.xml- an example of assembly model in XML language

```
<lqn-model name="SimpleAbl" description="an xml version of a Simple
Assembly model" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn.xsd">
  <solver-params comment="This is a test case" conv_val="0.00001"
it_limit="50" print_int="1" underrelax_coeff="0.8"/>
  <processor name="SimplePr1">
    <processor-params multiplicity="1"/>
    <task name="SimpleT1">
      <task-params mult="3" scheduling="ref"/>
      <entry name="SimpleE1">
        <entry-activities>
          <activity name="SimpleE1_ph1">
```

```

        <activity-params host-demand-mean="1"/>
        <synch-call dest="S1.service1" calls-mean="1"/>
        <synch-call dest="S1.service2" calls-mean="2"/>
        <synch-call dest="SimpleE3" calls-mean="1"/>
    </activity>
    <activity-sequence type="PH1PH2">
        <phase1>SimpleE1_ph1</phase1>
    </activity-sequence>
</entry-activities>
</entry>
</task>
<task name="SimpleT3">
    <task-params mult="2" activity-graph="NO"/>
    <entry name="SimpleE3">
        <entry-activities>
            <activity name="SimpleE3_ph1">
                <activity-params host-demand-mean="1.5"/>
            </activity>
            <activity-sequence type="PH1PH2">
                <phase1>SimpleE3_ph1</phase1>
            </activity-sequence>
        </entry-activities>
    </entry>
</task>
</processor>
<processor name="SimplePr2">
    <task name="SimpleT4">
        <task-params activity-graph="NO"/>
        <entry name="SimpleE4">
            <entry-activities>
                <activity name="SimpleE4_ph1">
                    <activity-params host-demand-mean="5.0"/>
                </activity>
                <activity-sequence type="PH1PH2">
                    <phase1>SimpleE4_ph1</phase1>
                </activity-sequence>
            </entry-activities>
        </entry>
    </task>
</processor>
<slot id="S1" bind-target="SingleMod">
    <Interface>
        <in-port name="service1" connect-from="SimpleE1"/>
        <in-port name="service2" connect-from="SimpleE1"/>
        <out-port name="request1" connect-to="SimpleE3"/>
        <out-port name="request2" connect-to="SimpleE4"/>
    </Interface>
    <binding>
        <!--parameter assignment here for SingleMod-->
        <parameter name="$SingleT2_mult" value="4"/>
        <parameter name="$SingleE2_demand" value="5.5"/>
        <processor-binding source="SinglePr" target="SimplePr1"/>
        <!--source refers to elements in the inner component -->
        <!--target refers to elements in the slot -->
        <port-binding source="p1" target="service1"/>
        <port-binding source="p2" target="service2"/>
    </binding>
</slot>

```

```

    <port-binding source="p3" target="request1"/>
    <port-binding source="p4" target="request2"/>
  </binding>
</slot>
</lqn-model>

```

Invoke the assembly tool from the command line:

```
LQNASsemble SimpleAbl
```

The input file extension name (.xml) is not needed in the command line. The output file is SimpleAbl_ft.xml and its content is displayed in List 4-4 below.

List 4-4 The output SimpleAbl_ft.xml from SimpleAbl.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<lqn-model name="SimpleAbl" description="an xml version of a Simple
Assembly model" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn.xsd">
  <solver-params comment="This is a test case" conv_val="0.00001"
it_limit="50" print_int="1" underrelax_coeff="0.8"/>
  <processor name="SimplePr1">
    <processor-params multiplicity="1"/>
    <task name="SimpleT1">
      <task-params mult="3" scheduling="ref"/>
      <entry name="SimpleE1">
        <entry-activities>
          <activity name="SimpleE1_ph1">
            <activity-params host-demand-mean="1"/>
            <synch-call dest="S1_SingleE2" calls-mean="1"/>
            <synch-call dest="S1_SingleE3" calls-mean="2"/>
            <synch-call dest="SimpleE3" calls-mean="1"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>SimpleE1_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
    <task name="SimpleT3">
      <task-params mult="2" activity-graph="NO"/>
      <entry name="SimpleE3">
        <entry-activities>
          <activity name="SimpleE3_ph1">
            <activity-params host-demand-mean="1.5"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>SimpleE3_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
  </processor>
</lqn-model>

```

```

    </entry>
  </task>

<task name="S1_SingleT1">
  <task-params mult="1" activity-graph="NO" scheduling="ref"/>
  <entry name="S1_SingleE1">
    <entry-activities>
      <activity name="SingleE1_ph1">
        <activity-params host-demand-mean="1.0"/>
        <synch-call dest="S1_SingleE3" calls-mean="1"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>SingleE1_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
<task name="S1_SingleT2">
  <task-params mult="4" activity-graph="NO"/>
  <entry name="S1_SingleE2">
    <entry-activities>
      <activity name="SingleE2_ph1">
        <activity-params host-demand-mean="5.5"/>
        <synch-call dest="SimpleE3" calls-mean="1"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>SingleE2_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
  <entry name="S1_SingleE3">
    <entry-activities>
      <activity name="SingleE3_ph1">
        <activity-params host-demand-mean="1"/>
        <synch-call dest="SimpleE4" calls-mean="1"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>SingleE3_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
</processor>
<processor name="SimplePr2">
  <task name="SimpleT4">
    <task-params activity-graph="NO"/>
    <entry name="SimpleE4">
      <entry-activities>
        <activity name="SimpleE4_ph1">
          <activity-params host-demand-mean="5.0"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>SimpleE4_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
</processor>

```

```
</task>
</processor>

</lqn-model>
```

In the output, there are no slots or interfaces. The elements of the sub-model have been plugged into the system model. The tasks and entries have been renamed by prefixing the slot id “S1” in this case (e.g. S1_SingleT1). The calls in the system model and sub-model have been correctly directed. (e.g. the calls that are in italic font have been directed. SimpleE1 is connected to S1_SingleE2 and S1_SingleE3 instead of to S1.service1 and S2.service2) The output model is very similar to the graphical one in Figure 3-18 except for no interfaces.

The following test case is adopted from the example in Figure 3-17 and the inner sub-model shown in Figure 3-16. The XML document is very similar to the previous assembly model except that it has interfaces. The slot and bindings are listed in List 3-2. The input and output document are listed in Appendix B.2 and B.3 Again, after running the assembling tool, the interfaces of the inner component have disappeared. The slot has also disappeared.

These output XML models were validated against the schema lqn.xsd and lqn-sub.xsd and the results indicate they are all valid.

All the output files were compared to those that were created manually from the sub-models and assembly models. The comparison indicates they are the same.

4.5.2 Validation of xml2LQN

The validation of this tool involves all the syntactic elements that are defined in the LQN model in XML language. These elements include:

- Solver parameters: comment on the model, convergence value, maximum number of iterations, print interval and under-relaxation coefficient.

- Processor parameters: scheduling type, multiplicity, quantum, speed factor and replication number.
- Task parameters: task scheduling type, think time, priority, multiplicity and replication number.
- Entry parameters: arrival rate, priority, coefficient of variation, max-service-time, phase type (Deterministic or Stochastic), fan-in, fan-out, service time, synchronous calls, asynchronous calls, forwarding calls, think time and possible starting activity.
- Activity parameters: service time, coefficient of variation, phase type, think time, synchronous calls and asynchronous calls.
- Activity Precedence relationship: sequential, OR-fork, OR-join, And-fork, And-join and loop.

The command to invoke this tool is as follows:

```
xml2LQN <xml document> [output model]
```

If there is no specified output model name, the default one will have the same file name as the XML file except that the extension name is different. The output model has a default extension name lqn.

The output models in LQN language were compared to those that were manually created. The comparisons indicate that they are exactly the same. Below some examples are given that demonstrate this.

The first case adopts the example with the graphical representation shown in Figure 4-5 and the source document is in List 4-2. The source document is named as list4-2.xml. In this example there are no activities involved. Run the following command.

```
xml2LQN List4-2
```

The output is List4-2.lqn which is listed below. The output is exactly the same as in List 4-1 except that the comments are removed because they are ignored in the XML document list4-2.xml.

List 4-5 Output model List4-2.lqn from transforming List4-2.xml

```
#This is the output from xml2LQN
G
"This is a test case"
0.00001
50
1
0.8
-1
P 0
p P1 f
p P2 f
p P3 f
-1
T 0
t UIF r user -1 P1 m 10
t T2 n e1 e2 -1 P2
t T3 n ex -1 P2
t T4 n e3 -1 P3 m 10
-1
E 0
s user 1.0 -1
y user e1 5 -1
y user e2 20 -1
s e1 0.04 -1
y e1 ex 3 -1
s e2 0.2 -1
y e2 ex 2 -1
s ex 0.9 -1
y ex e3 4 -1
s e3 0.05 -1
-1
```

The second test case has involved lots of activities: sequential, OR-fork, AND-fork, AND-join and looping activities. The original model in LQN language is listed in the following List 4-6 below, which can also be found at <http://www.sce.carleton.ca/rads/lqn/lqn-documentation/>. The model in XML language is listed in List 4-7 and the file is named activity-test.xml. Run the following command.

```
xml2LQN activity-test activity
```

The output activity.lqn is listed in List 4-8.

List 4-6 The original LQN model activity-test in LQN language

```
#This template documents the use of activities in depth
G "Activity template" 1e-06 50 5 0.9 -1

P 0
  p UserP f i
  p ServerP s #processor sharing at the server
  p Disk1P f
  p Disk2P f
-1

T 0
  t User r user -1 UserP z 50 m 50
  t Server n server -1 ServerP m 4 #4 threads with activities
  t BigLoop n bigLoop -1 ServerP i
    #pseudo-task for a complex loop pattern
  t Disk1 n disk1read disk1write -1 Disk1P
  t Disk2 n disk2read disk2write -1 Disk2P
-1

E 0
  s user 1.0 -1
  f user 1 -1
  y user server 1 -1 #one request to the server per cycle
  A server serverStart #entry server is defined by
  #activities, with the first one being serverStart
  A bigLoop bigLoopStart
  s disk1read 0.04 -1 #operation time of this entry
  s disk1write 0.04 -1
  s disk2read 0.03 -1
  s disk2write 0.03 -1
-1

#Optional sections for definition of activities
# One section for each task that has activities, beginning A TaskName
# list of activity parameters, using the syntax for entry parameters,
# but with just one value and no terminator -1
# : (separator), then a section for precedence among activities
# Syntax for precedence:
# a1 -> a2 for sequence
# a1 -> a2 & a3 ... AND-fork (any number)
# a1 & a2 ... -> a3 AND join
# a1 & a2 ... -> a3 & a4 ... AND join followed by AND fork
# a1 -> (prob2)a2 + (prob3)a3 ... OR fork (any number, with
#probabilities)
# a1 -> meanCount*a2,a3....for a repeated activity a2, followed by a3
# (notice that activities that follow a2 are inside the loop)
# a6[entryName] indicates that after a6, a reply will be sent
# to entryName
```



```

A Server
s serverStart 0.0
  #every activity that is used must have a host demand
s seqInit 0.3
s parInit 0.1
s parA 0.05
y parA disk1read 1.3 #average of 1.3 read operations
s parB 0.08
y parB disk2read 2.1
s parReply 0.01
s loopOperation 0.1
y loopOperation disk1read 0.7
s loop2 0
f bigLoopDriver 1 #exactly one call operation (deterministic)
y bigLoopDriver bigLoop 1
  #trigger the pseudo-task for the complex loop
s seqReply 0.005
s loopEnd 0
:
serverStart -> (0.4)seqInit + (0.6)parInit;
parInit -> parA & parB;
parA & parB -> parReply;
parReply[server]; #reply for the parallel branch
seqInit -> 3.5* loopOperation, loopEnd;
loopOperation -> loop2; #this activity is also in the loop
loopEnd -> 1.2* bigLoopDriver, seqReply;
  #big loop is executed avge 1.2times
seqReply[server] #reply for the sequential branch
-1
A BigLoop #activities for the loop pseudo-task
# (a loop pseudo-task is needed if there is a fork-join within a loop)
s first 0.01 #execute
f second 1 #deterministic sequence in this activity
y second disk1write 1 #exactly one file write on this branch
y third disk2write 1 #average of one write on this branch
s fourth 0.13 #execute only
:
bigLoopStart -> first;
first -> second & third;
second & third -> fourth;
fourth[bigLoop]
#generate the reply from the pseudo task, ending the loop op
-1

```

List 4-7 The LQN model activity-test in XML language

```

<lqn-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn.xsd">
  <solver-params comment="Test case of activity-templ"
conv_val="0.000001" it_limit="50" print_int="5" underrelax_coeff="0.9"/>
  <processor name="UserP">
    <processor-params multiplicity="i"/>

```

```

<task name="User">
  <task-params mult="50" scheduling="ref" activity-graph="NO" think-
time="50"/>
  <entry name="user">
    <entry-activities>
      <activity name="user_ph1">
        <activity-params host-demand-mean="1.0"
          call-order="DETERMINISTIC"/>
        <synch-call dest="server" calls-mean="1"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>user_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
</processor>
<processor name="ServerP">
  <processor-params scheduling="ps"/>
  <task name="Server">
    <task-params mult="4" activity-graph="YES"/>
    <entry name="server">
      <entry-activities>
        <activity name="serverStart">
          <activity-params host-demand-mean="0.0"/>
        </activity>
        <activity name="seqInit">
          <activity-params host-demand-mean="0.3"/>
        </activity>
        <activity name="parInit">
          <activity-params host-demand-mean="0.1"/>
        </activity>
        <activity name="parA">
          <activity-params host-demand-mean="0.05"/>
          <synch-call dest="disk1read" calls-mean="1.3"/>
        </activity>
        <activity name="parB">
          <activity-params host-demand-mean="0.08"/>
          <synch-call dest="disk2read" calls-mean="2.1"/>
        </activity>
        <activity name="parReply">
          <activity-params host-demand-mean="0.01"/>
        </activity>
        <activity name="loopOperation">
          <activity-params host-demand-mean="0.1"/>
          <synch-call dest="disk1read" calls-mean="0.7"/>
        </activity>
        <activity name="loop2">
          <activity-params host-demand-mean="0"/>
        </activity>
        <activity name="bigLoopDriver">
          <activity-params host-demand-mean="0.0"
            call-order="DETERMINISTIC"/>
          <synch-call dest="bigLoop" calls-mean="1"/>
        </activity>
        <activity name="seqReply">

```

```

    <activity-params host-demand-mean="0.005"/>
  </activity>
  <activity name="loopEnd">
    <activity-params host-demand-mean="0"/>
  </activity>
  <activity-sequence type="GRAPH">
    <first-activity name="serverStart"/>
    <precedence>
      <pre>serverStart</pre>
      <post-OR>
        <activity name="seqInit" prob="0.4"/>
        <activity name="parInit" prob="0.6"/>
      </post-OR>
    </precedence>
    <precedence>
      <pre>parInit</pre>
      <post-AND>
        <activity name="parA"/>
        <activity name="parB"/>
      </post-AND>
    </precedence>
    <precedence>
      <pre-AND>
        <activity name="parA"/>
        <activity name="parB"/>
      </pre-AND>
      <post>parReply</post>
    </precedence>
    <precedence>
      <pre-LOOP>seqInit</pre-LOOP>
      <post-LOOP head="loopOperation" count="3.5" end="loopEnd"/>
    </precedence>
    <precedence>
      <pre>loopOperation</pre>
      <post>loop2</post>
    </precedence>
    <precedence>
      <pre-LOOP>loopEnd</pre-LOOP>
      <post-LOOP head="bigLoopDriver" count="1.2" end="seqReply"/>
    </precedence>
    <reply-activity>seqReply</reply-activity>
    <reply-activity>parReply</reply-activity>
  </activity-sequence>
</entry-activities>
</entry>
</task>
<task name="BigLoop">
  <task-params mult="i" activity-graph="YES"/>
  <entry name="bigLoop">
    <entry-activities>
      <activity name="first">
        <activity-params host-demand-mean="0.01"/>
      </activity>
      <activity name="second">
        <activity-params host-demand-mean="0.0"
          call-order="DETERMINISTIC"/>

```

```

    <synch-call dest="disk1write" calls-mean="1"/>
  </activity>
  <activity name="third">
    <activity-params host-demand-mean="0.0"/>
    <synch-call dest="disk2write" calls-mean="1"/>
  </activity>
  <activity name="fourth">
    <activity-params host-demand-mean="0.13"/>
  </activity>
  <activity-sequence type="GRAPH">
    <first-activity name="bigLoopStart"/>
    <precedence>
      <pre>bigLoopStart</pre>
      <post>first</post>
    </precedence>
    <precedence>
      <pre>first</pre>
      <post-AND>
        <activity name="second"/>
        <activity name="third"/>
      </post-AND>
    </precedence>
    <precedence>
      <pre-AND>
        <activity name="second"/>
        <activity name="third"/>
      </pre-AND>
      <post>fourth</post>
    </precedence>
    <reply-activity>fourth</reply-activity>
  </activity-sequence>
</entry-activities>
</entry>
</task>
</processor>
<processor name="Disk1P">
  <task name="Disk1">
    <task-params activity-graph="NO"/>
    <entry name="disk1read">
      <entry-activities>
        <activity name="disk1read_ph1">
          <activity-params host-demand-mean="0.04"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>disk1read_ph1</phase1>
          <reply-activity>disk1read_ph1</reply-activity>
        </activity-sequence>
      </entry-activities>
    </entry>
    <entry name="disk1write">
      <entry-activities>
        <activity name="disk1write_ph1">
          <activity-params host-demand-mean="0.04"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>disk1write_ph1</phase1>

```

```

        <reply-activity>disk1write_ph1</reply-activity>
    </activity-sequence>
</entry-activities>
</entry>
</task>
</processor>
<processor name="Disk2P">
    <task name="Disk2">
        <task-params activity-graph="NO"/>
        <entry name="disk2read">
            <entry-activities>
                <activity name="disk2read_ph1">
                    <activity-params host-demand-mean="0.03"/>
                </activity>
                <activity-sequence type="PH1PH2">
                    <phase1>disk2read_ph1</phase1>
                    <reply-activity>disk2read_ph1</reply-activity>
                </activity-sequence>
            </entry-activities>
        </entry>
        <entry name="disk2write">
            <entry-activities>
                <activity name="disk2write_ph1">
                    <activity-params host-demand-mean="0.03"/>
                </activity>
                <activity-sequence type="PH1PH2">
                    <phase1>disk2write_ph1</phase1>
                    <reply-activity>disk2write_ph1</reply-activity>
                </activity-sequence>
            </entry-activities>
        </entry>
    </task>
</processor>
</lqn-model>

```

List 4-8 The output model activity.lqn

```

#This is the output from xml2LQN
G
"Test case of activity-templ"
0.000001
50
5
0.9
-1
P 0
p UserP f i
p ServerP s
p Disk1P f
p Disk2P f
-1
T 0
t User r user -1 UserP z 50 m 50
t Server n server -1 ServerP m 4

```

```

t BigLoop n bigLoop -1 ServerP i
t Disk1 n disk1read disk1write -1 Disk1P
t Disk2 n disk2read disk2write -1 Disk2P
-1
E 0
s user 1.0 -1
f user 1 -1
y user server 1 -1
A server serverStart
A bigLoop bigLoopStart
s disk1read 0.04 -1
s disk1write 0.04 -1
s disk2read 0.03 -1
s disk2write 0.03 -1
-1
A Server
s serverStart 0.0
s seqInit 0.3
s parInit 0.1
s parA 0.05
y parA disk1read 1.3
s parB 0.08
y parB disk2read 2.1
s parReply 0.01
s loopOperation 0.1
y loopOperation disk1read 0.7
s loop2 0
f bigLoopDriver 1
s bigLoopDriver 0.0
y bigLoopDriver bigLoop 1
s seqReply 0.005
s loopEnd 0
:
serverStart -> (0.4)seqInit+(0.6)parInit;
parInit -> parA&parB;
parA&parB->parReply;
seqInit -> 3.5*loopOperation,loopEnd;
loopOperation -> loop2;
loopEnd -> 1.2*bigLoopDriver,seqReply;
seqReply[server];
parReply[server]
-1
A BigLoop
s first 0.01
f second 1
s second 0.0
y second disk1write 1
s third 0.0
y third disk2write 1
s fourth 0.13
:
bigLoopStart -> first;
first -> second&third;
second&third->fourth;
fourth[bigLoop]
-1

```

Again, the output in List 4-8 is exactly the same as in List 4-6 except the comments are removed in List 4-8. There are two other test cases that were experimented but not listed here. One case that involves OR-fork and OR-Join is listed in par-db.xml in Appendix B.5. The output is named par-db.lqn in Appendix B.6. The other test case involves task activities where activities from different entries in a task join at a point. The model in XML language named as task-act.xml is attached in Appendix B.7. The output model is named task-act.lqn listed in Appendix B.8. Both examples were originated from those in Greg Franks's Ph.D. thesis [10].

4.5.3 Validation Against the Combined Tool –LQComposer

The LQComposer is composed of the LQNAssemble and xml2LQN. It takes the assembly model in XML language as the input and the output is a model in the previous LQN language. This output model is ready to be solved by LQN solvers provided that the XML documents are all correct in syntax and semantics.

The validation here takes the assembly model in List 4-3. The command to invoke this is as follows.

```
LQComposer SimpleAbl
```

There will be an intermediate file called SimpleAbl_ft.xml which is the output from the LQNAssemble.

The final output file is SimpleAbl.lqn which is attached in Appendix B.4. It is the same as the one that was created manually.

4.5.4 Conclusions

The tools have been tested against many cases and the results show that they work properly. The second tool xml2LQN has been tested against all the syntactic elements in the XML schema and the output models were compared to the original ones. They are exactly the same. Currently, the output model can be solved by LQNS and ParaSRVN. The

features of run-control and plot-control which are used in SPEX have not been implemented in this thesis work.

Chapter 5 Industrial Case Study

This section presents a case study of a Management Information System, based loosely on a commercial software product. A conceptual model is introduced and the component-based approach is applied. The performance results and analysis are presented later in this chapter.

5.1 A Conceptual Performance Model for a Management Information System (MIS)

The purpose of this study is to examine feasibility and scalability issues for a component based Management Information System (MIS). This system is mainly used to manage and monitor data that are collected and stored in an organization. It presents many kinds of reports and analysis for many different purposes. This system is a typical three-tiered E-Business system. Clients send requests to the web server. The web server does some processing and then sends requests to the application server. The application server is responsible for executing the specific business logic and it needs to access data from a database back-end. After the results have been computed, they are sent back to the clients through the web server. In this system, there may be several application nodes. Therefore, there is a scheduler which schedules and dispatches the requests using round-robin policy. An application node includes a variable number of report servers. Each report server has 2 separate processes. There are two main types of requests in this system, namely reporting request and viewing request respectively.

Based on the preliminary architecture of the software system, a conceptual model has been built. This model is shown in Figure 5-1 below. In this model, there is one set of users sending two types of requests, Reporting Service requests and Viewing Service requests. The scenario paths of these two services have been separated, using different entries in the tasks, labeled RS and VS within the entry names. Task DSP is for dispatching requests. Entry DispRS dispatches reporting service requests while DispVS dispatches viewing requests. There are two types of threads serving the reporting service. One of them serves small reporting requests and the other one serves big reporting requests. Tasks RSClient and VSClient are pseudo tasks. They do not represent any processing, delay or system

functions. They are only used to capture and report the service times of Reporting requests and Viewing requests respectively.

There are 6 kinds of processors in the model as explained below.

- ClientP represents the processor on which Client tasks (Client, RSClient and VSClient) are executed.
- ScheduP refers to the processor that hosts the task of Scheduler.
- WebP is the processor that hosts the task of WebS (Web Server).
- AsP is the processor on which application server tasks are executed. (DSP is the dispatching task. SmallReport1 and SmallReport2 are two tasks processing small reporting requests. BigReport1 and BigReport2 are tasks processing big reporting requests. Big reports are segregated so they can be scheduled separately. ViewData is the task serving viewing requests. ReportGen is for generating reports.)
- CachP is the processor that hosts the task of CacheInfo which is the cache server for caching reusable data.
- DBP refers to the processor on which database server tasks are running. It serves big report data request, small report data request and caching data request. Caching data means those reusable results data that are stored for caching server task CacheInfo to access.

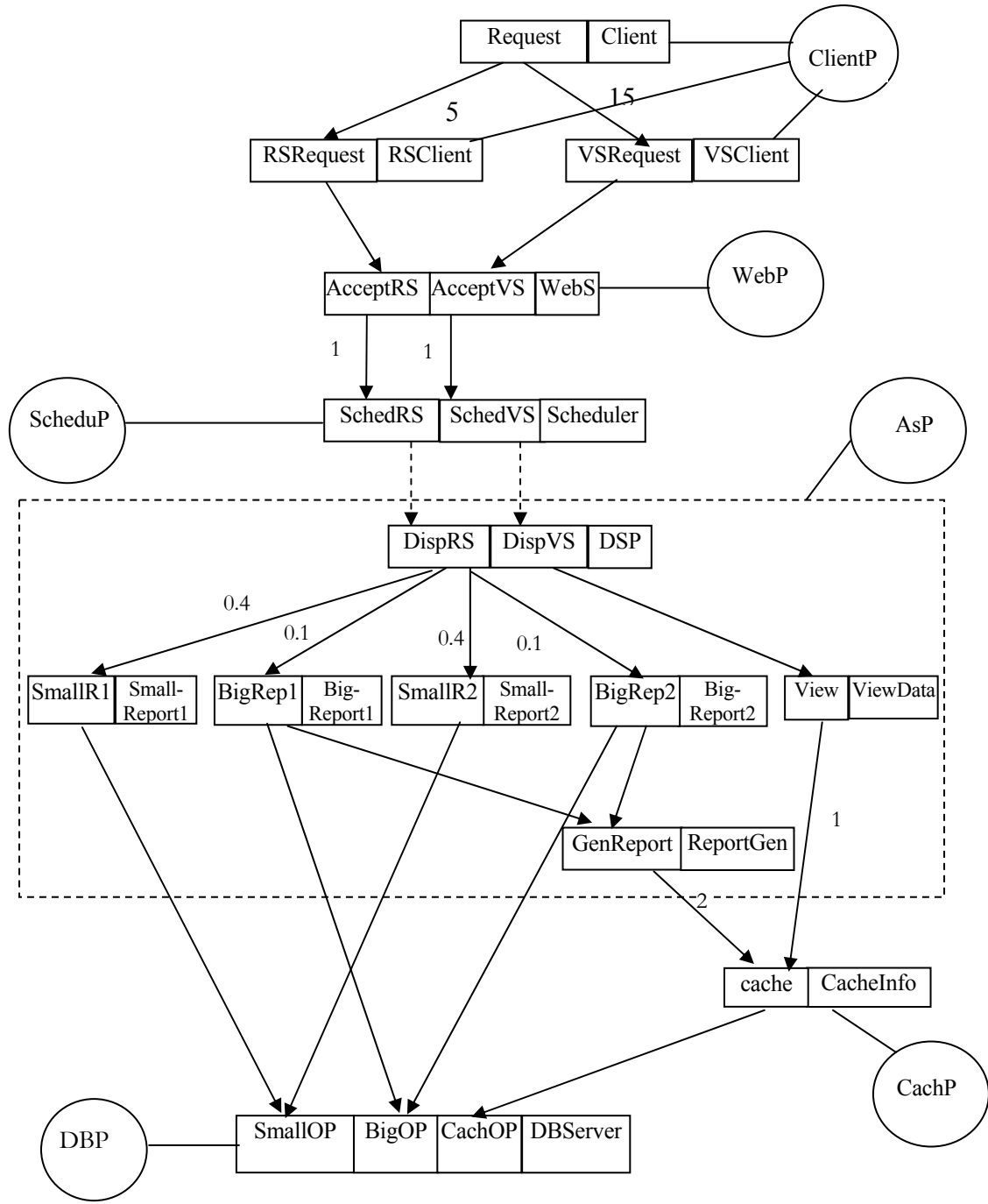


Figure 5-1 A Conceptual Model for the MIS

This study was conducted when the system was planned. This application system needs to meet some performance goals as listed below.

- The mean response time for a viewing request that a user perceives should be less than 10 seconds. For a reporting request, it should be less than 1 minute.
- It must be able to scale up to accept 300 concurrent users.
- Cost is important; it can be measured by the number of application nodes.

Therefore, this case study is carried out regarding the following performance issues.

1. System capacity and scalability. What will the system response time be for different number of concurrent users?
2. The limiting factors of performance. What's the bottleneck in the system?
3. Configuration.
 - What's the impact of multi-threading levels? Which one should be multi-threaded?
 - What's the impact of adding more hardware or more powerful hardware? Which processor has the strongest impact on system performance?
 - What about replication?

To avoid revealing confidential data, the values of the workload and traffic parameters of the model described have been invented.

5.2 The Component Based Approach to Model the MIS

This section describes a component based approach to model the MIS system. This approach is based on a system assembly model and a library of sub-models. In this case study, since the main concern is on the application server, the web server and the database server are simply modeled as single tasks. However, if necessary, the web server and the database server can also be modeled in detail. For instance, the web server component might be modeled as shown in Figure 2-1 in Chapter 2. In paper [9], the authors have described an approach to derive the model parameters from measurement for the layered

queuing model of the web server. In paper [31], a method has been elaborated on modeling a distributed database system using layered queuing network modeling technique.

The MIS case in the thesis has involved one major sub-model for the application server. The rest are represented as single tasks.

Instead of this single sub-model, there could be a collection of sub-models for different products in a software product line. For instance, there may be no caching involved. Or it may need to access some additional databases. Other possible options may be that it needs to access additional servers for specific analysis (e.g. for data mining or optimization). By taking different parameters, it can also model the application node whose internals may have different software but still accomplish the similar functionalities with different performance attributes.

5.2.1 The Assembly Model for the MIS

The assembly model shown in Figure 4-2 has been adapted to this MIS as shown below in Figure 5-2. The label AppS is the slot id for the application server component. The DBServer now has three entries serving three different kinds of requests, big report request, small report request and caching data request. The slot for application server has two incoming interfaces for processing reporting request and viewing request respectively. It has three outgoing interfaces that send three kinds of requests which are named as bigReq for big report request, smallReq for small report request and cachReq for caching data request. These interfaces bind the corresponding interfaces of the application server sub-model.

The XML document for this assembly model is listed in Appendix C.1 with the name of MISAssemble.xml. Some of the parameters in this model presented in Appendix C are made up in order to keep the model anonymous.

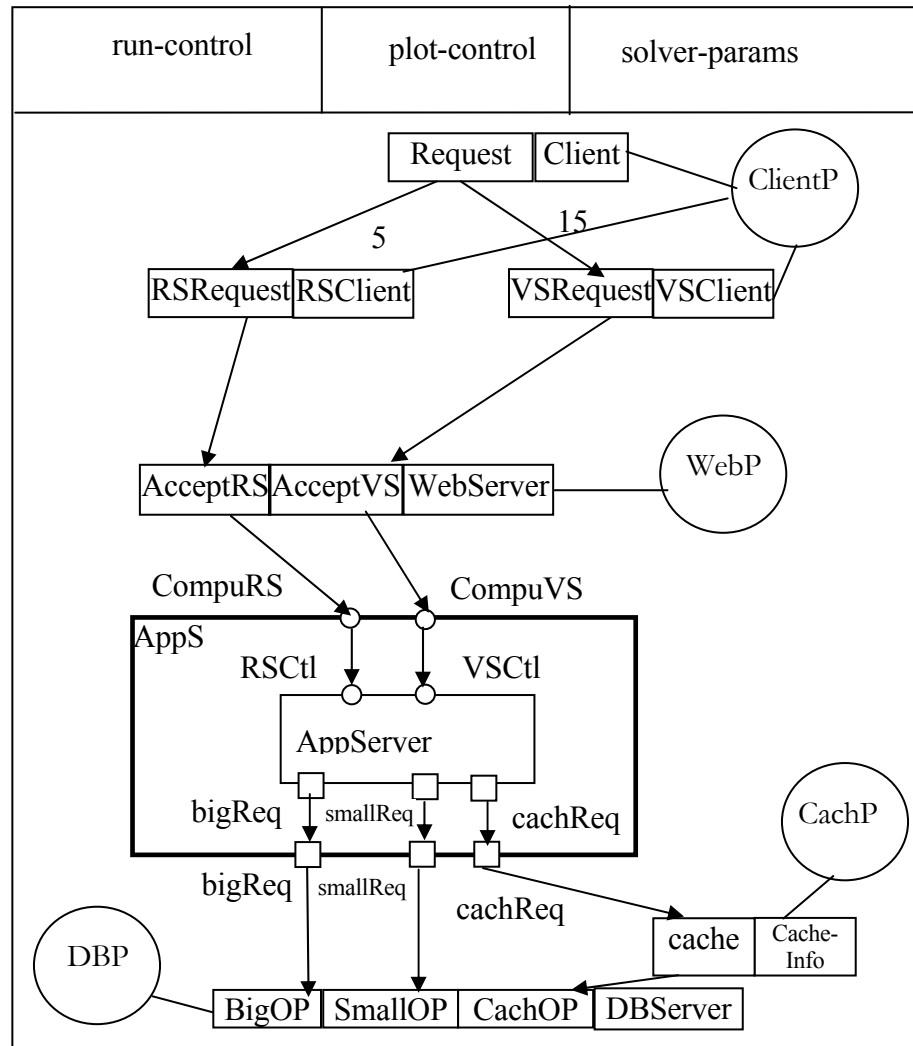


Figure 5-2 The Assembly Model for the MIS

5.2.2 The Application Server Component Model

The internals of the application server component model for the MIS are shown in Figure 5-3 below. The internal processor ScheduP is not replaceable, while processor AsP is replaceable. The XML document for this component model is listed in Appendix C.2 with the name of AppComp.xml.

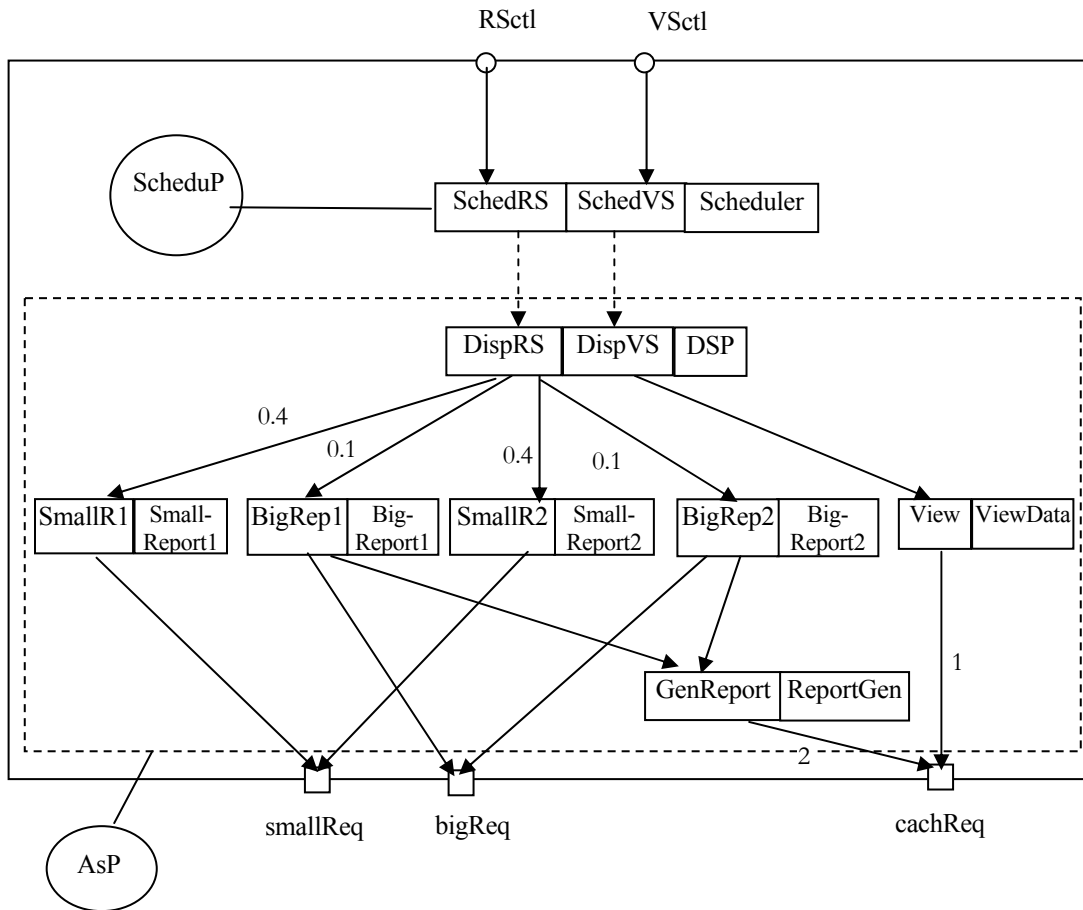


Figure 5-3 The Component Model for the Application Server

5.2.3 Resulting Model from LQComposer

Invoke the tool by running the following command from the DOS environment.

```
LQComposer MISAssemble
```

The final resulting model is in the file named MISAssemble.lqn. This file is attached in Appendix C.3. There is an intermediate file named MISAssemble_ftl.xml which is the output from LQNAssemble, the first part of the LQComposer. This final model is compared to a model that was built manually and they are the same except that, now the

tasks that are within the Application Server have prefix of 'AppS'. The final model can be displayed by Jlnqdef (which is an editing tool for LQN models) and is shown in Figure 5-4. The performance results obtained from this model are the same as those from the manually created model.

This tool has also created a model in which the replaceable processors are replaced by the system processor CachP. In other words, in this model, all the tasks hosted by AsP and the task hosted by CachP share one system processor. This is one of the system deployment options. It is also convenient to create models that have multiple replicated application nodes.

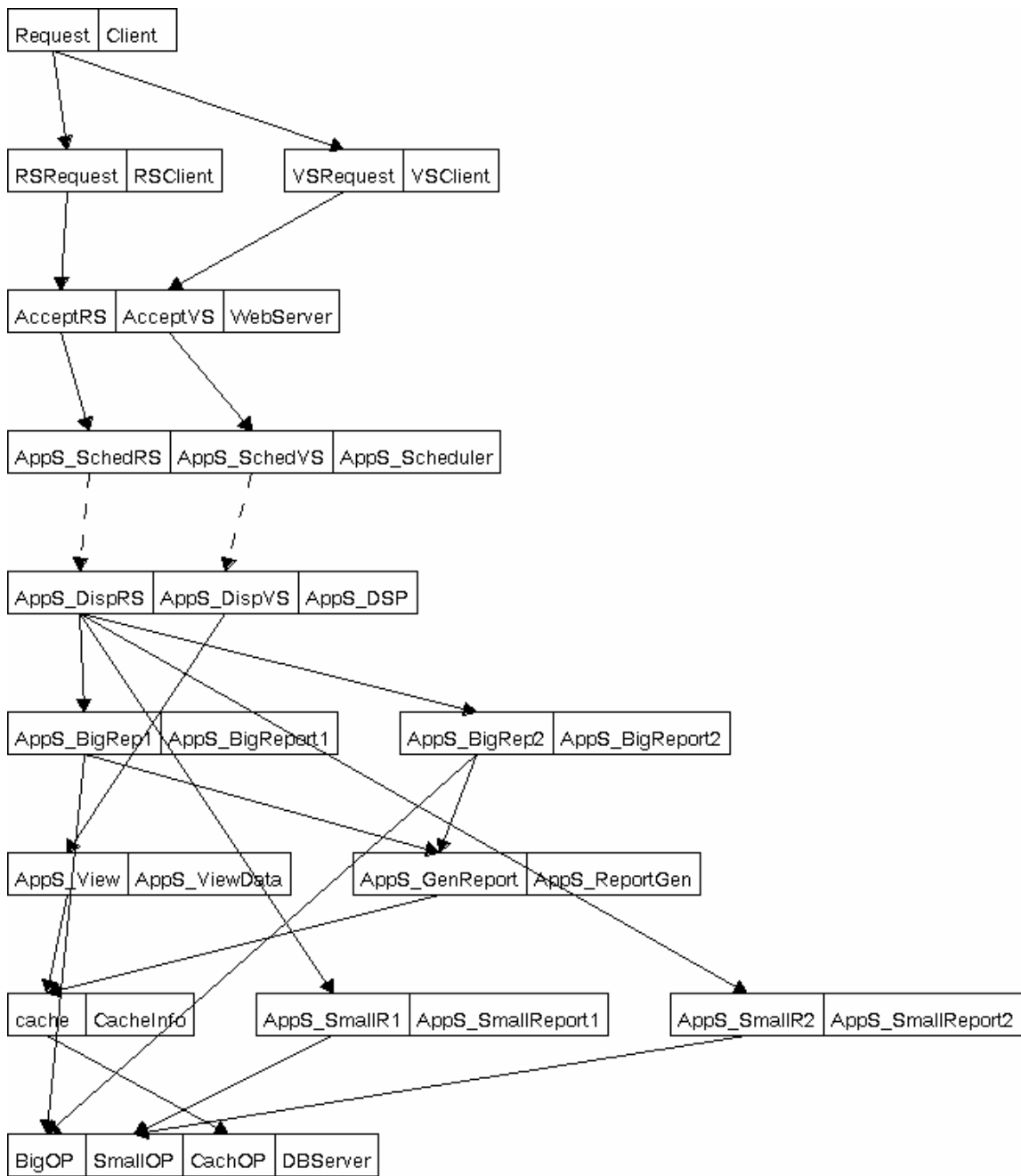


Figure 5-4 Graphical Representation of MISAssemble.lqn

5.3 Performance Results for the Base Case

This section first describes the performance results of the base case. From the results, it identifies some performance problems. Then a series of tests have been devised to see how the system performance could be affected by different changes such as multithreading, adding more processors as well as different configurations. These will be described in the following sections.

The base case is as follows:

- A user session includes an average of 20 requests, divided between 5 Reporting service requests and 15 Viewing service requests.
- The Viewing service, on average, needs to access the cache server CacheInfo just once.
- For the Reporting service, its report generating service, on average, needs to access the cache server CacheInfo twice.
- There are 2 report servers in the system.
- Web Server, Scheduler, Dispatcher, Viewing Service and Database latency are modeled as infinite threaded, pure delays. Some of these are infinite threaded but have a single processor. So they may have processor contentions.
- The caching task CacheInfo is single threaded while AsP (the processor of Application Server) has 2 processors.

The results of system response time (in seconds) for the two types of requests are shown in Figure 5-5 below. The results of system throughput (responses per second) are shown in Figure 5-6.

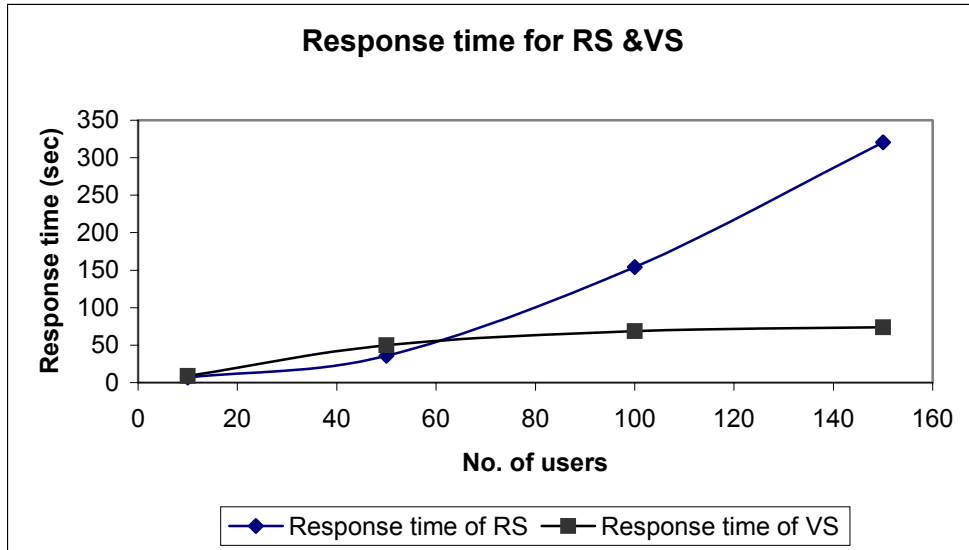


Figure 5-5 Response time of the Base Case

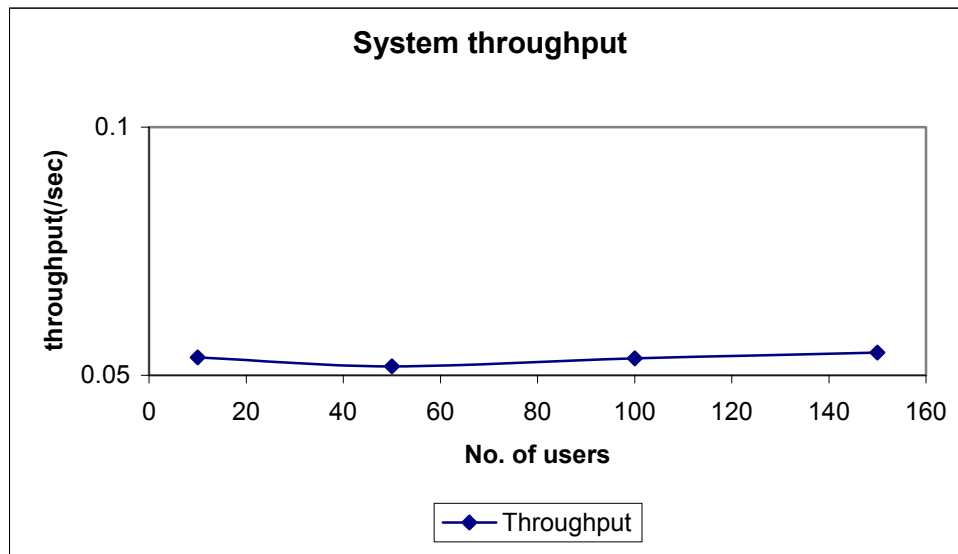


Figure 5-6 System throughput of the Base Case

The response times are quite long for the Reporting service. However, this includes creating quite large reports, possibly hundreds of pages, so some reports will take a long time. The results also show that, in the base case, the system is saturated at CacheInfo (Cache Server) even when it is lightly loaded. Since each viewing service needs access to CacheInfo, its response time is a little larger than Reporting service due to the fact that

only 20% of the Reporting requests need access to CacheInfo. However, when the number of users increases, Report Server is also saturated which causes long delay to the Reporting requests. Report Server becomes saturated because it has to wait longer for CacheInfo. So CacheInfo is the real bottleneck which “pushes back” to its upper layer, the Report Server. When Report Server becomes saturated, viewing service is much quicker than reporting service.

In order to see how the performance can be improved, a series of changes have been made to the Base Case. These changes can be divided into three groups.

- Group I. Changes involving multithreading and/or adding processors.
- Group II. Changes involving replications and multithreading.
- Group III. Study of the limiting factors on the system performance.

The results and analysis of these changes are presented in the following sections.

5.4 Performance Results with Multithreading

In the Base Case, there are software bottlenecks in the system. Hardware devices are not constraining. In order to demonstrate this, the following test case has been designed in which the computing power of AsP has been increased.

Case I-1 The number of processors of AsP increases to 4

In this case, the rest of the model remains the same as in the Base Case. The results are shown below in Figure 5-7 and Figure 5-8.

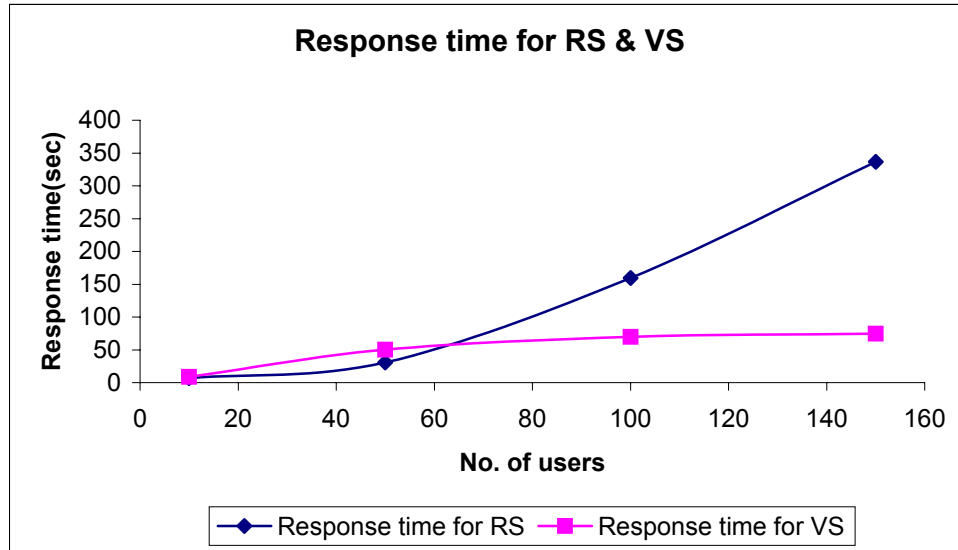


Figure 5-7 System response time for Case I-1

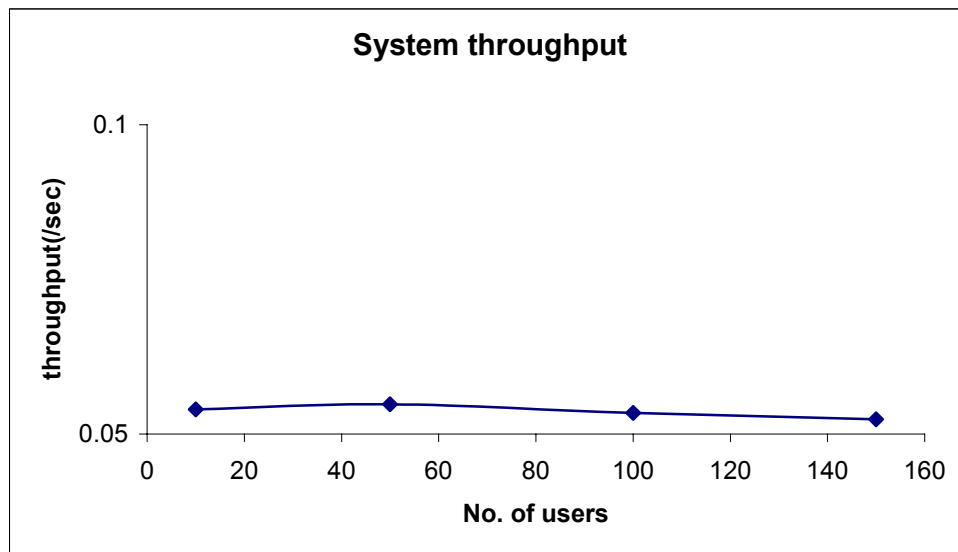


Figure 5-8 System throughput for Case I-1

Results of Case I-1 are almost the same as the previous Base Case. CacheInfo is still a Software Bottleneck. The most effective way to eliminate this kind of performance problem is to make the software bottleneck task multithreaded. Therefore the next case has been conducted in which CacheInfo is multithreaded.

Case I-2 CacheInfo is multithreaded to 10.

In this case, the rest of the model remains the same as in the Base Case. The results of this case are shown in Figure 5-9 and Figure 5-10 below.

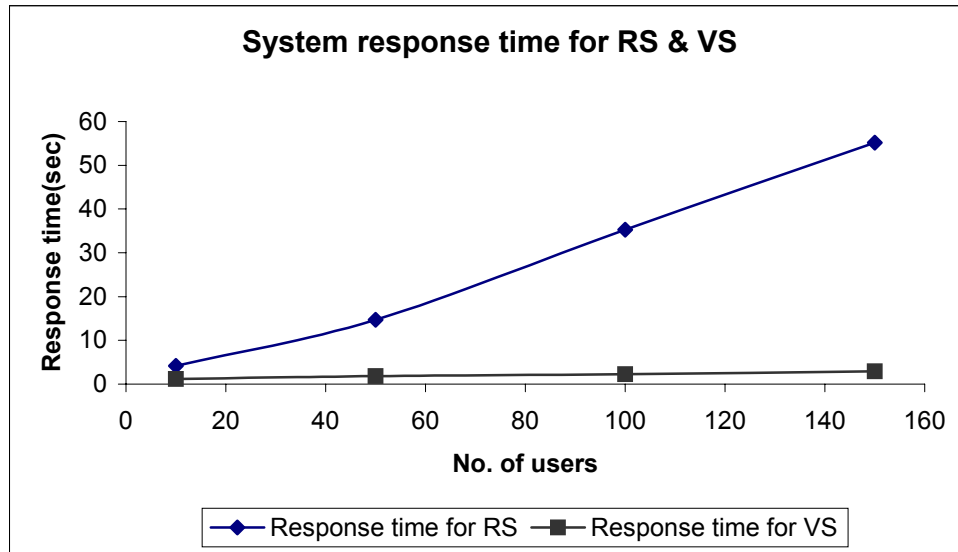


Figure 5-9 System response times of RS & VS for Case I-2

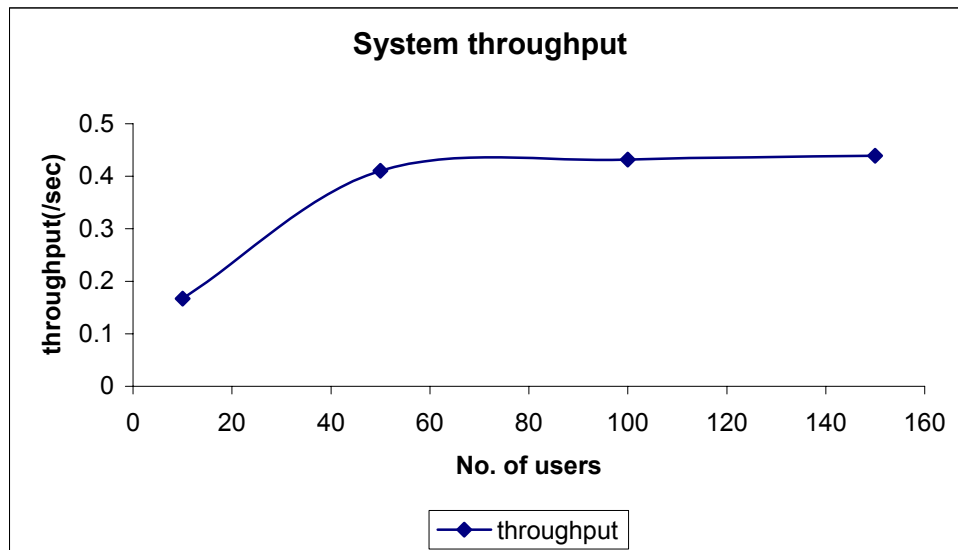


Figure 5-10 System throughput for Case I-2

The results in this case are very encouraging. The response time of both reporting request and viewing request has improved significantly. System throughput is nearly 10 times higher. In addition, Viewing service is always much quicker than Reporting Service. In this case, the bottleneck has moved to Report Server, especially at SmallR1 and SmallR2 threads. Since this system bottleneck occurs on software tasks, not on hardware processors, increasing the number of processors won't improve the system. This test was done but not listed here.

The results show that the utilization of CachP (CacheInfo processor) reaches 70% when the number of users increases to 150. This is the highest processor utilization in the system.

In order to see if more threads of CacheInfo can bring better response time for the system, the following case is experimented.

Case I-3 CacheInfo is multithreaded to 20.

In this case, the rest of the model remains the same as in the base case. The results of this case are shown in Figure 5-11 and Figure 5-12 below.

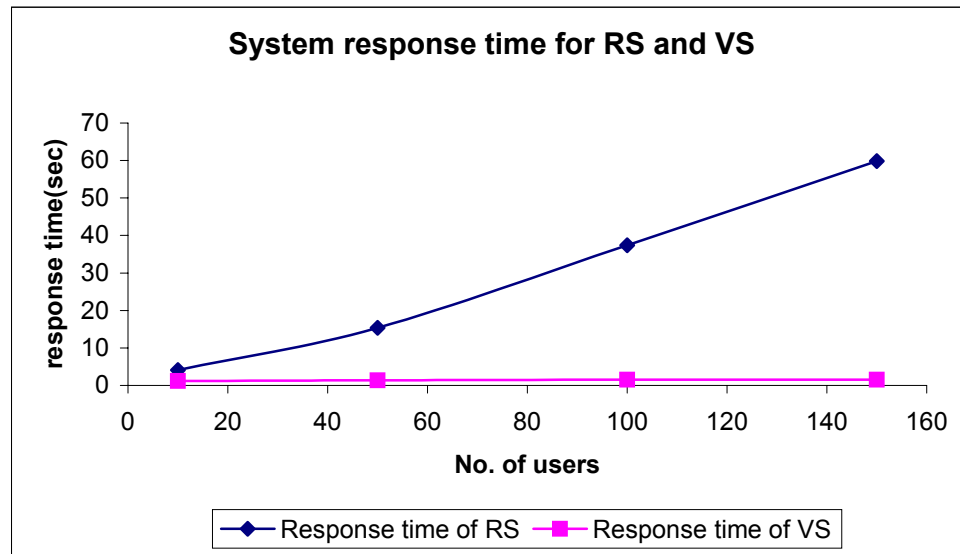


Figure 5-11 System response time of RS and VS for Case I-3

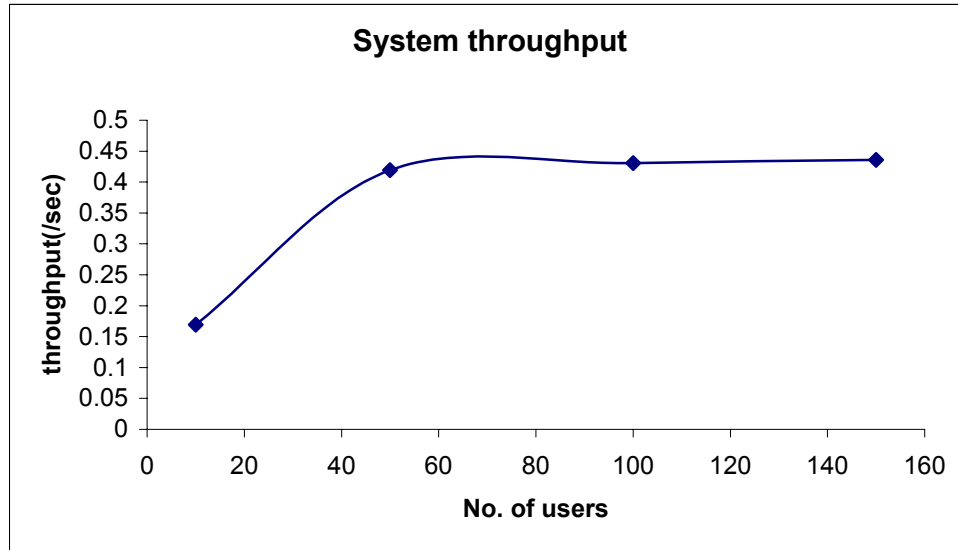


Figure 5-12 System throughput for Case I-3

The results of this case have not shown much difference compared to the previous Case I-2. This is because the system is now saturated at Report Server which cannot be solved by increasing the number of threads of CacheInfo. The results of this case indicate that on average, only 9 threads of CacheInfo have been utilized. In this case, it also shows that CacheInfo processor can become a future hardware bottleneck since it is now the most heavily utilized processor in the system. Its utilization reaches 74% under 150 concurrent users.

In order to see how much can be gained if CacheInfo, SmallReport1 and SmallReport2 are all multithreaded, the following case is conducted.

Case I-4 CacheInfo is multithreaded to 20. SmallReport1 and SmallReport2 are multithreaded to 2.

The rest are the same as in base case. The results of this case are shown in Figure 5-13 and Figure 5-14 below.

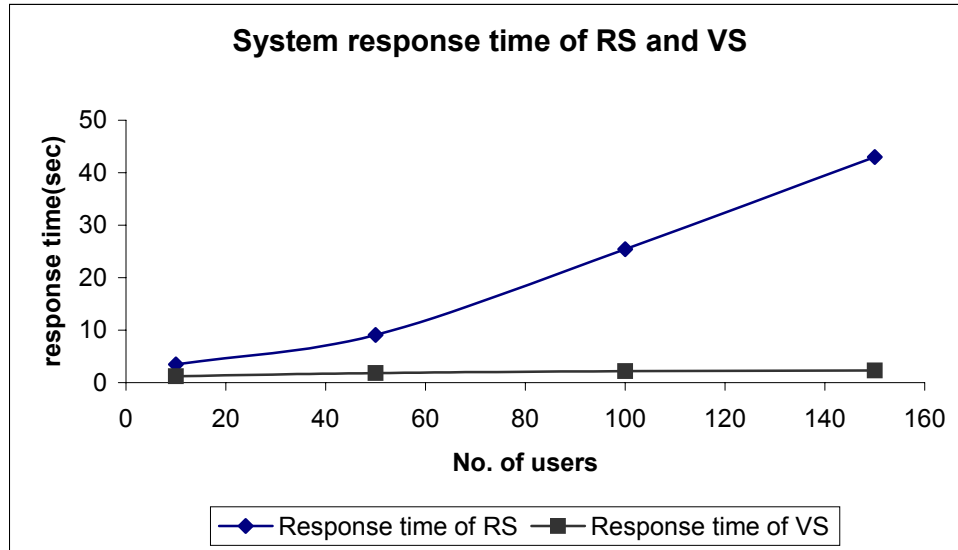


Figure 5-13 System response time of RS and VS for Case I-4

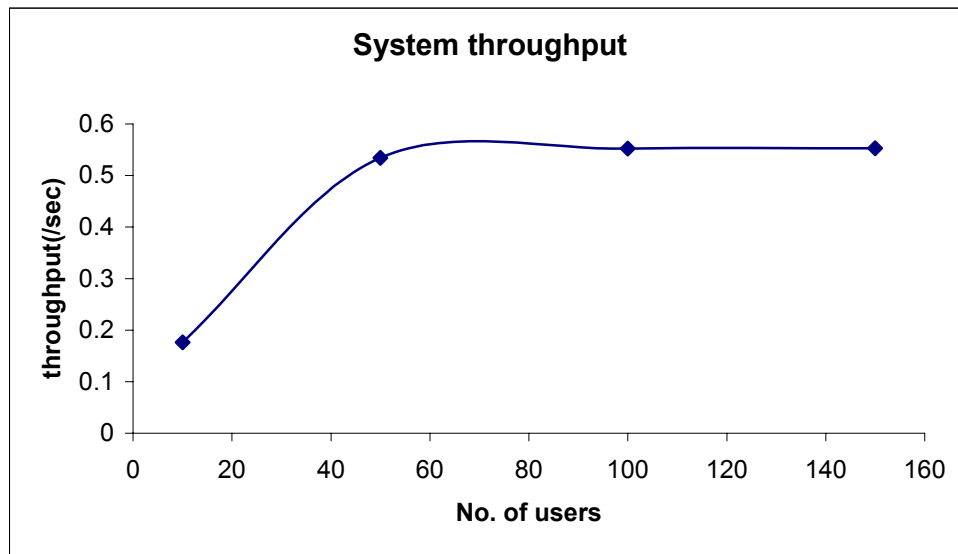


Figure 5-14 System throughput for Case I-4

The results of this case indicate that:

- Now the bottleneck has moved to the threads of BigReport1 and BigReport2 in Report Servers.
- In this case, when the number of users increases to 50, CachP Processor has become the hardware bottleneck in the system since its utilization reaches 90%.

- If AsP has only one processor, it is also a hardware bottleneck since its utilization approaches 90%.

Conclusions from the Base Case to Group I

By analyzing the results of these tests, the following observations can be made.

1. There are actually two software bottlenecks in the system as shown in Figure 5-15 below. The open rectangles above the tasks represent queues of the task. The shaded areas represent the potential bottlenecks. The bottlenecks are moving back and forth between two parts in the system. The CacheInfo process is the first software bottleneck and Report Server is the second one. By multithreading these software processes, it improves the system a lot. System response time decreases significantly and higher throughput can be obtained.
2. There could be a hardware bottleneck in the system, too. The CacheInfo processor CachP could become a hardware bottleneck if more threads of CacheInfo are available. To solve this, use a multiprocessor CachP.

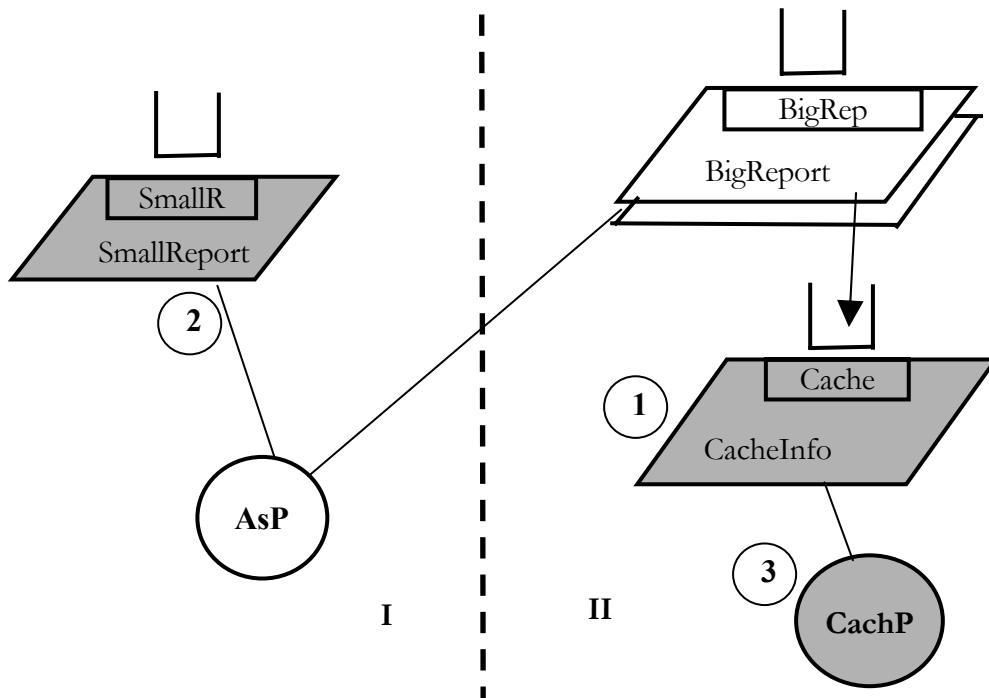


Figure 5-15 Bottlenecks in the system

(Circled numbers indicate the orders that the task or processor may become a bottleneck in the system)

The Figure 5-15 indicates that if the thread is increased on one side, then the software bottleneck moves to the other side. Both can be increased together and this is examined later in Group III in Section 5.6.

5.5 Performance Results with Replicated Application Nodes

This group of experiments is concerned with replication issues using multiple processors. There are two possible ways to deploy this application system. One deployment could be that the Application Server is deployed on one single node that has powerful computing capabilities such as having multi-processors. The other one could be that the Application Server is replicated on several nodes with each node less powerful, having single processor.

This group of tests replicated the processor of AsP and the tasks within the dotted box as shown in Figure 5-1. Each replicated application node has one processor. The results are compared with those in the Base Case and Group I.

Case II-1 Two Replicated Nodes

The parameters for this model are the same as the base case except that the tasks hosted by the processor of AsP are replicated to 2. No multithreading has been introduced in this case. The results are then compared with the Base Case as shown in Figure 5-16, Figure 5-17 and Figure 5-18.

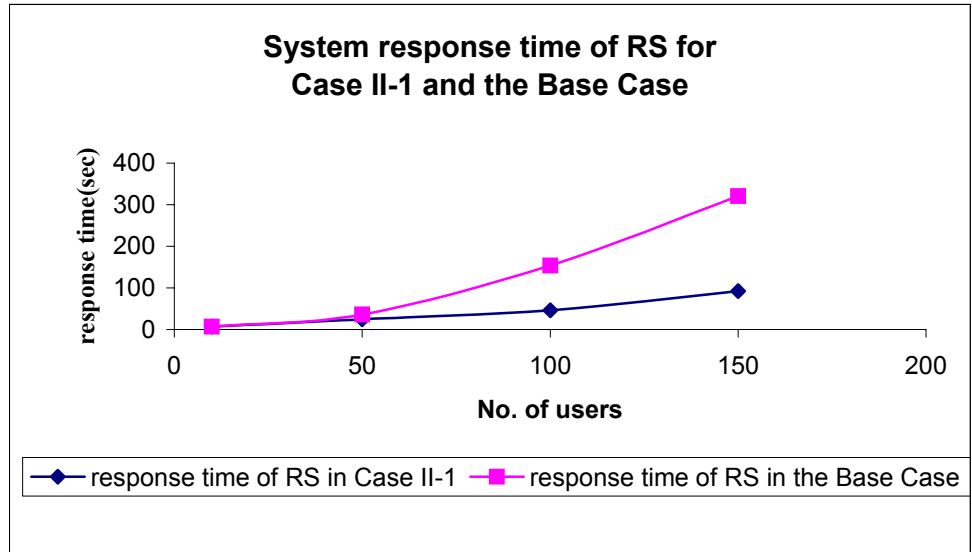


Figure 5-16 System response time of RS in Case II-1 and the Base Case

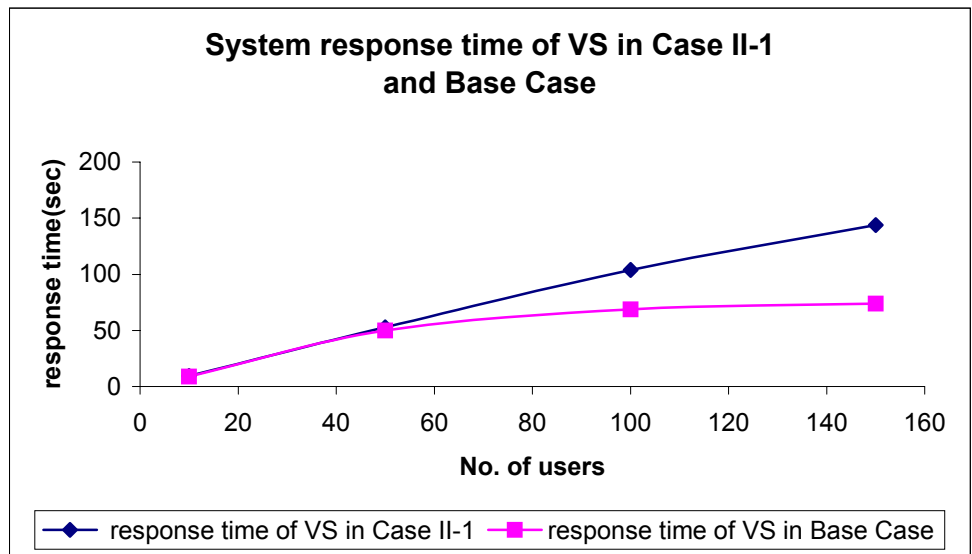


Figure 5-17 System response time of VS in Case II-1 and the Base Case

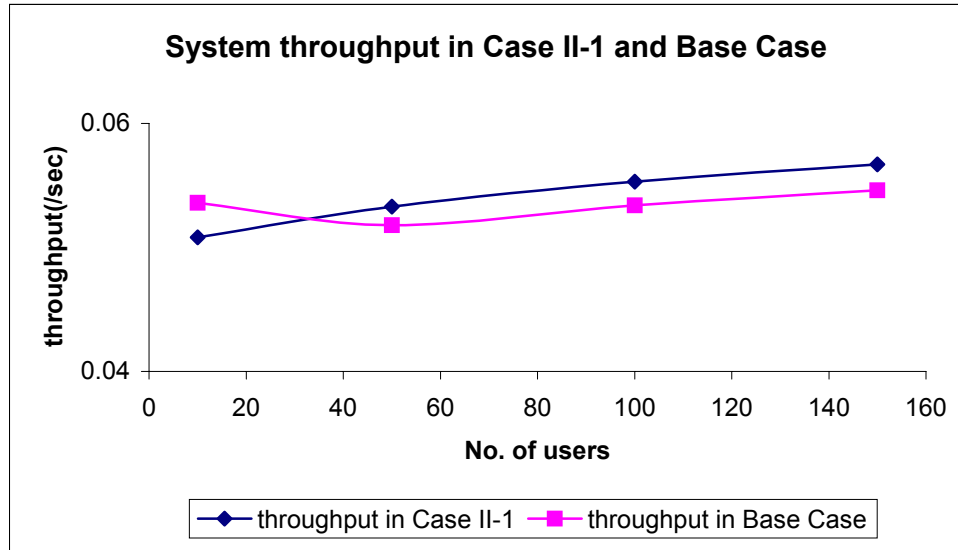


Figure 5-18 System throughput in Case II-1 and Base Case

By analyzing these results, the following observations can be made.

- Under light loads, this replicated Case II-1 has slightly lower throughput and almost the same response time as those in the Base Case.
- Under heavy loads, replication gives more throughput and much smaller response time for reporting requests, but have somewhat longer response time for viewing service requests. This is because viewing requests are being flooded out by reporting requests when they contend for the CacheInfo.

This shows that response times of two types of requests are sensitive to this kind of configuration. It takes longer for a Viewing request while shorter for a Reporting request and this is contrary to the Base Case. It also shows that Report Server saturates a little more slowly than in the Base Case. The different behavior in saturation is due to the heavier loads on CacheInfo, which affects viewing service requests more. In order to make the task CacheInfo more capable of handling more requests, multithreading is introduced in the next test case.

Case II-2 Two Replicated Nodes and 10 Threads of CacheInfo.

In this case, there are again two replicas of the processor of AsP and the corresponding tasks on that processor. Now the CacheInfo task is multithreaded to 10 threads. Figure 5-19 and Figure 5-20 show the results.

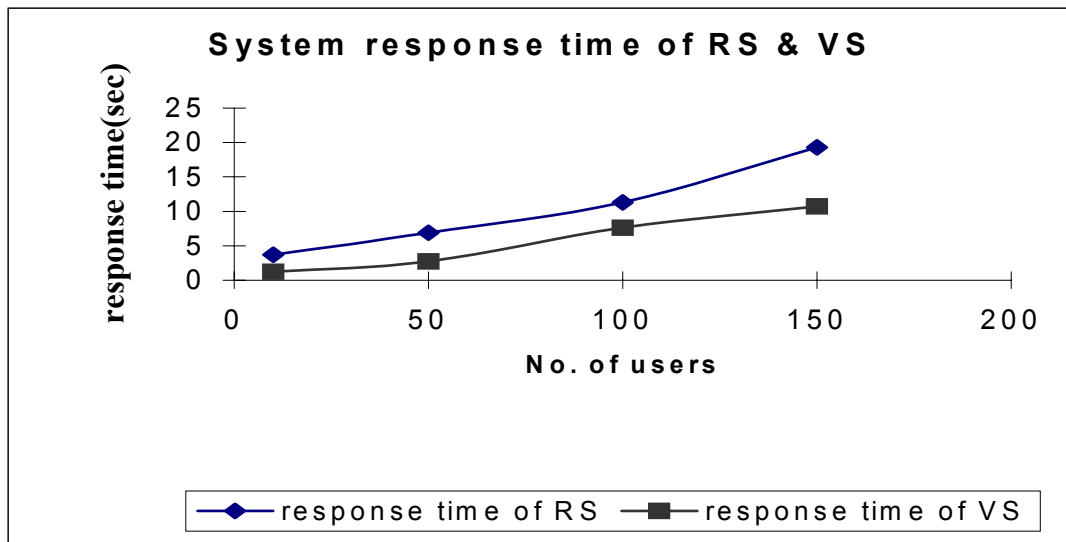


Figure 5-19 System response time of RS and VS for Case II-2

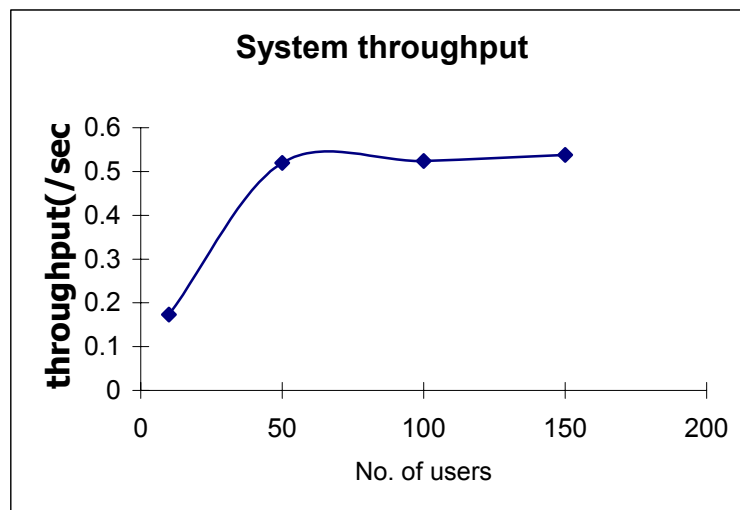


Figure 5-20 System throughput for Case II-2

These results indicate that the process of CacheInfo is still the software bottleneck in the system. It can also be predicted that the next software bottleneck is Report Server, specifically, the BigReport threads and the hardware bottleneck will be the CachP processor. Compared to Case I-2 in which CacheInfo is multithreaded to 10 and AsP has 2 processors, the task of CacheInfo is saturated faster in this case while Report Server is

saturated more slowly. The response time of Reporting request is much shorter while the time of Viewing request is a little longer. System throughput has improved quite a bit in this case.

Conclusions from Groups II

By analyzing the results from Group II and comparing them with the Base Case and Group I, the following conclusions can be drawn.

Replication of AsP makes very little difference to the overall throughput. CacheInfo is still the software bottleneck. However, it does shift the response time between two classes of requests; improving the RS (Reporting Service) response time to less than half (which makes little use of CacheInfo) and making the VS (Viewing Service) a little bit longer. These changes are substantial.

5.6 Scaling Limits

It may be that more threads of the Report Server and CacheInfo could be usefully provided. To examine how many would be utilized, if they were there, some studies were done that assumed infinite numbers of threads at the CacheInfo and at two Report Servers, each with SmallReport and BigReport thread pools. The number of AsP processors was varied, while an infinite processor pool was provided for CachP.

The results are shown in Figure 5-21, Figure 5-22 and Figure 5-23 respectively.

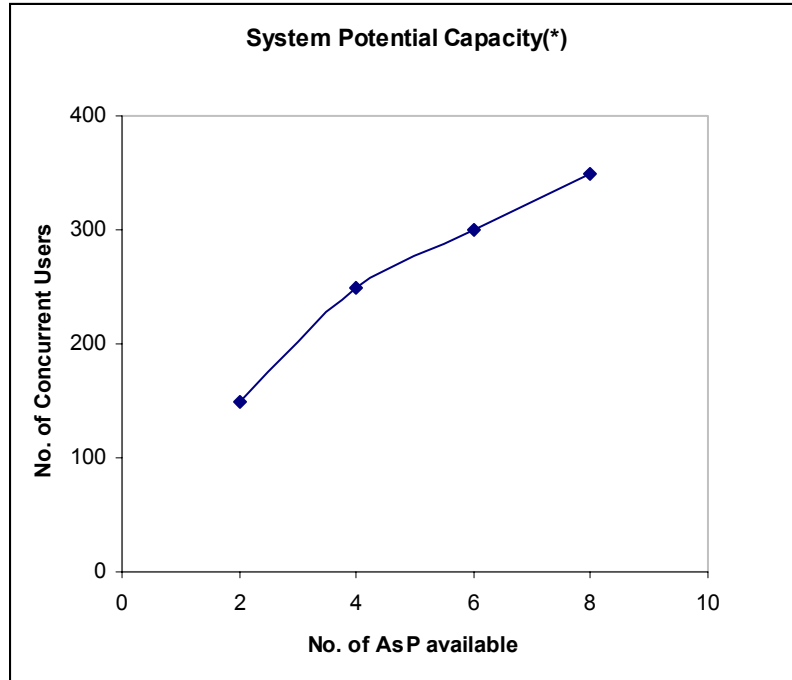
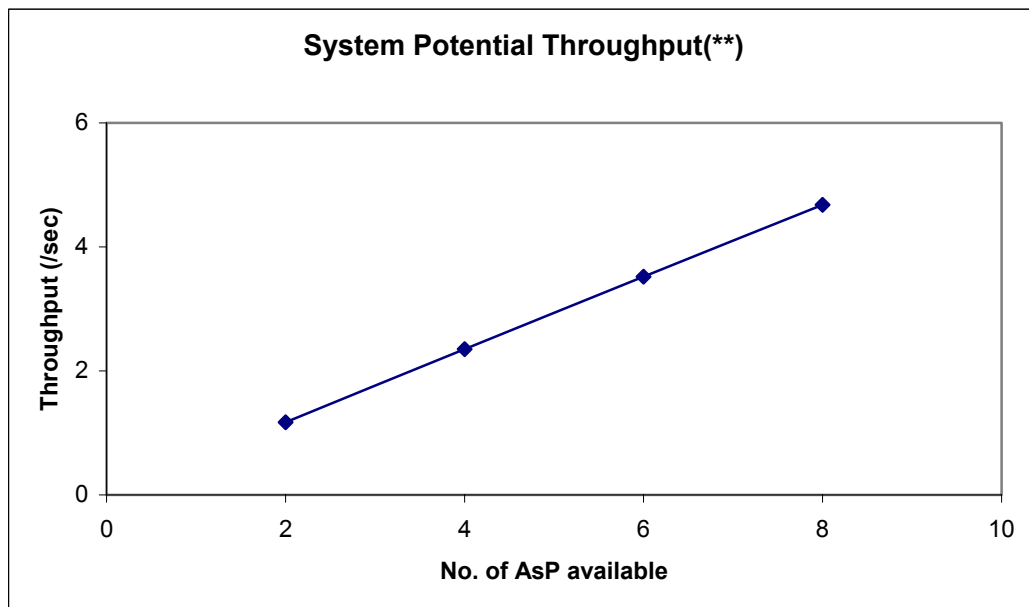


Figure 5-21 System Potential Capacity (*)

The results in figure indicate that the system can accept around 300 concurrent users and meet the performance goals of delays for two types of requests provided that it has enough resources. These resources include 8 AsP processors, about 100 threads of CachInfo, 10 threads of SmallReport, 30 threads of BigReport and 6 CachP processors. These required resources are illustrated in Figure 5-23.



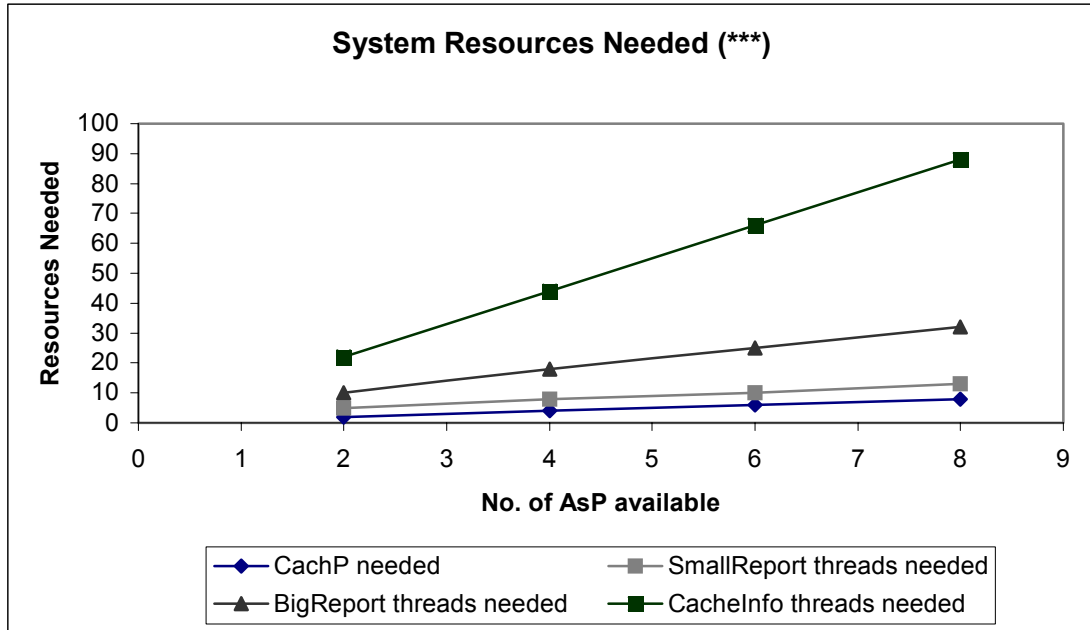


Figure 5-23 System Resources Needed (*)**

- (*) System capacity refers to the maximum number of concurrent users that the system can handle. The assumptions here are that the number of processors of AsP are available as well as the required resources shown in Figure 5-23 are also provided.
- (**) The maximum throughputs that can be obtained provided the number of processors of AsP available and the required resources shown in Figure 5-23 are also provided.
- (***) System Resources Needed means those resources that are required in order to obtain the maximum throughput and capacity provided that the number of processors of AsP are available.

The results show that

- As more processors become available, more threads of Reporting services and CacheInfo are required in order to obtain the maximum throughput of the system. Otherwise, these will become the software bottleneck in the system.
- The capacity scales up smoothly, with additional threads of Reporting Service and CacheInfo. It is quite interesting that the number of threads of CacheInfo required is proportional to the number of AsP processors.

- The number of CachP processors required is the same as the number of AsP processors. This shows that CachP processors could become the hardware bottleneck if otherwise.
- More AsP processors will bring significant benefits to the system capacity only if the CachP processors, and the numbers of threads of reporting service and CacheInfo are scaled at the same time. In addition, many more threads of the CacheInfo are required than that of reporting services.
- The results also indicate that considerably more Report Server and CacheInfo threads are indicated, than are used in the previous experiments presented in this chapter.
- Overall the design appears to be scalable if sufficient threads are provided.

Chapter 6 Conclusions

This section draws the conclusions of the thesis work. It also points out some future work.

6.1 Conclusions

In this thesis, an XML based language for describing performance sub-models and assembly models have been developed which has been elaborated in Chapter 3. The parameterized sub-model reflects the performance attributes of the software component in different environments. Component sub-models are reusable, just as software components themselves. The nesting component sub-model definition allows building sub-models nestedly which is the case that a software component might consist of other components. Since the model definition is based on XML language, it can present the model in a more portable and understandable way. It also matches the structured LQN model in nature. As many tools are now available supporting XML documentation, it makes applying LQN models in a wider range of performance modeling.

An assembler tool and a methodology to automatically generate performance models for component-based systems have been introduced and presented in this thesis. The design and testing of the tools were described in Chapter 4. The assembly model derived from the software architecture is reusable and adaptable. The assembly model for a generic E-business application system can be applied to different applications that have their own application servers. By changing the interfaces of the slot of application server and its related bindings, a system model for the specific application can be obtained.

Finally, an industrial case study has been carried out and some substantial results have been obtained. The case study was presented in Chapter 5. The tool has also been applied to this case and it has demonstrated the automated process to create system models. The analysis of the model was carried through to show how bottlenecks and sensitivity can be investigated, and scalability plans can be evaluated.

6.2 Limitations

Although the XML language can characterize structured data in a more understandable way, it has the disadvantages of verbosity. Editing an LQN model in XML language needs tool support. Otherwise, it could be a pain to write the lengthy document.

Currently, the sub-model parameterization does not support the case where a slot can be bound to a variable component. In other words, a component model cannot be passed in as an argument to the slot. At current stage, the tool LQComposer cannot directly flatten a nested model that involves multiple levels of nesting. It has to go through multiple steps to accomplish this.

Although a sub-model can have a variable parameter of the task replication number, the assembly tool has not calculated the corresponding values of fanin and fanout.

The output system model from the tool can be solved by LQNS and ParaSRVN. The features of run-control and plot-control that are used in SPEX have not been implemented.

The tool and the methodology have not been applied to many different cases.

6.3 Future Research

The future work should improve the LQML to allow variable component bindings. Improve the tool to be able to process multiple levels of recursions in the sub-model definition, and to be able to generate models that incorporate run-control and plot-control so that the output model can be solved by SPEX. Apply LQML to performance modeling, especially to model product lines. Develop practical libraries of component sub-models.

References

- [1] F. Bause and P. S. Kritzinger. “Stochastic Petri nets: an introduction to the theory”, Wiesbaden, Vieweg Verlag, Germany, 2002
- [2] J. Bosch. “Design and use of software architectures; Adopting and evolving a product-line approach”, Addison-Wesley, 2000
- [3] X. Cai, M. R. Lyu, K. F. Wong, R. Ko, “Component-based software engineering: technologies, development frameworks, and quality assurance schemes”, Proceedings of APSEC 2000: Software Engineering Conference, 2000. Seventh Asia-Pacific, 5-8 Dec. 2000, pp 372 –379
- [4] S. Chen, I. Gorton, A. Liu, Y. Liu, “Performance Prediction of COTS Component-based Enterprise Applications”, Proceedings of 5th ICSE Workshop on Component-Based Software Engineering: Benchmarks for Predictable Assembly, <http://www.sei.cmu.edu/pacc/CBSE5/liu-cbse5-29.pdf>, Orlando, Florida USA, May 19-20, 2002
- [5] J. Clark, “XSL Transformations (XSLT) Version 1.0”, <http://www.w3c.org/TR/xslt.html#section-Introduction>, November, 1999
- [6] P. Clements and L. Northrop. “Software Product Lines; Practices and Patterns”, Addison-Wesley, 2000
- [7] E. Connell, F. Knop and V. Rego, “ParaSol: A Multithreaded System for Parallel Simulation Based on Mobile Threads”, <http://www.cs.purdue.edu/research/PaCS/ps/pswsc95.pdf>, 1995 Winter Simulation Conference
- [8] O. Das, M. Woodside, "Evaluating Layered Distributed Software Systems with Fault Tolerant Features", Performance Evaluation, v 45, issue 1, May 2001, pp 57 - 76.
- [9] J. Dille, R. Friedrich, T. Jin and J. Rolia. “Web server performance measurement and modeling techniques.” Performance Evaluation, vol. 33, pp. 5-26. 1998
- [10] G. Franks, "Performance Analysis of Distributed Server Systems", Report OCIEE-00-01, Jan. 2000, PhD. thesis, Carleton University, Ottawa

- [11] R. G. Franks, C. M. Woodside, "Effectiveness of Early Replies in Client-Server Systems", Performance Evaluation, v 36-37, pp 165 - 184, 1999
- [12] G. Franks, M. Woodside, "Performance of Multi-level Client-Server Systems with Parallel Service Operations", Proc. First Int. Workshop on Software and Performance (WOSP98), pp. 120-130, Santa Fe, October 1998
- [13] G. Franks, A. Hubbard, S. Majumdar, J. Neilson, D.C. Petriu, J.A. Rolia and C.M. Woodside, "A Toolset for Performance Engineering and Software Design of Client-Server Systems", Performance Evaluation, vol. 24, pp117-136, 1995.
- [14] G. T. Heineman, W. T. Councill, "Component-Based Software Engineering; Putting the Pieces Together", Addison-Wesley, 2001
- [15] A. Hubbard, "SPEX: Software Performance Experiment Driver",
<http://www.sce.carleton.ca/rads/lqn/lqn-documentation/spex.txt>, August 1997
- [16] T. Israr "A Lightweight Technique for Extracting Software Architecture and Performance Models from Traces", thesis,
ftp://ftp.sce.carleton.ca/pub/cmw/israr_thesis.pdf, April 2001
- [17] P. Jogalekar and C.M. Woodside, "Evaluating the Scalability of Distributed Systems", Proc. of Hawaii Int. Conference on System Sciences, January 1998
- [18] M. Kay, "XSLT Programmer's Reference, 2nd Edition", Wrox Press, Birmingham. 2001
- [19] M. Kempa & V. Linnemann. "XML-Based Applications Using XML Schema", XML-Based Data Management and Multimedia Engineering-EDBT 2002 Workshops, Springer, Lecture Notes in Computer Science, pp67-90
- [20] J. Luthi. "Interval matrices for the bottleneck analysis of queueing network models with histogram-based parameters", Proceedings of IPDS '98 (International Computer Performance and Dependability Symposium, 7-9 Sept. 1998). pp142-151
- [21] S. Majumdar and T. Phillips. "Performance of scheduling strategies for client-server systems", Proceedings of International Conference on Parallel and Distributed Systems, 3-6 June 1996. pp 448 -455
- [22] D. McMullan, "Components In Layered Queuing Networks",
<http://www.sce.carleton.ca/rads/lqn/lqn-documentation/component3.pdf>, October 2, 2001

- [23] D. A. Menasce. "Two-Level Iterative Queuing Modeling of Software Contention", Proceedings of MASCOTS 2002, October, Fort Worth, Texas, USA. pp267-280
- [24] J.E. Neilson, C.M. Woodside, D.C Petriu and S. Majumdar, "Software Bottlenecking in Client-Server Systems and Rendezvous Networks", IEEE Trans. On Software Engineering, Vol. 21, No. 9, pp. 776-782, September 1995
- [25] J.Q. Ning, "Component-based software engineering (CBSE)", Proceedings of the Fifth International Symposium on Assessment of Software Tools and Technologies, 1997, 2-5 June 1997 pp 34 -43
- [26] D. Petriu, M. Woodside, "Analysing Software Requirements Specifications for Performance", Proc. Third Int. Workshop on Software and Performance, Rome, July 2002
- [27] D. Petriu and C. M. Woodside, "Software Performance Models from System Scenarios in Use Case Maps", Proc. 12 Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation (Performance TOOLS 2002), London, April 2002
- [28] D.B. Petriu, D. Amyot, C. M. Woodside, "Scenario-Based Performance Engineering with UCMNav", report SCE-03-07, Dept. of Systems and Computer Engineering, Carleton University, Feb. 2003
- [29] D.C. Petriu, R.G. Franks and A. Hubbard. "SRVN Input File Format", <http://www.sce.carleton.ca/rads/lqn/lqn-documentation/format.pdf>, November 24,1998
- [30] J.A. Rolia and K.C. Sevcik, "The Method of Layers", IEEE Transactions on Software Engineering, Vol. 21, No. 8 , Aug. 1995, pp 689 -700
- [31] F. Sheikh and C.M. Woodside, "Layered Analytic Performance Modelling of a Distributed Database System", Proc. 1997 International Conf. on Distributed Computing Systems, May 1997, pp. 482-490
- [32] K. H. Siddiqui and C.M. Woodside "Performance aware software development (PASD) using resource demand budgets" In the Proceedings of the third international workshop on Software and performance, pp.275 – 285, July 2003

- [33] M. Sitaraman, G. Kulczycki, J. Krone, W. F. Ogden and A.L.N. Reddy “Performance Specification of Software Components”, Proceedings of SSR '01, pp. 3-10. ACM/SIGSOFT, May 2001
- [34] C. U. Smith and L. G. Williams, “Performance Solutions”, Addison-Wesley, 2002
- [35] C. U. Smith, L. G. Williams, “Performance Engineering Evaluation of Object-Oriented Systems with *SPE·ED*”, Lecture Notes in Computer Science 1245: Computer Performance Evaluation Modelling Techniques and Tools, Springer, 1997
- [36] C. Szyperski, “Component Software; Beyond Object-Oriented Programming”, Addison-Wesley, 1998
- [37] M. Woodside "Scalability metrics and analysis of mobile agent systems", Proc. Workshop on Infrastructure for Scalable Mobile Agent Systems, at Autonomous Agents 2000, Barcelona, June 3, 2000
- [38] C.M. Woodside and C. Schramm "Scalability and Performance Experiments using Synthetic Distributed Server Systems", Distributed Systems Engineering, vol. 3, pp. 2-8, 1996
- [39] C.M. Woodside, J.E. Neilson, D.C. Petriu, S. Majumdar, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software", IEEE Transactions on Computers, Vol.44, Nb.1, pp 20-34, January 1995
- [40] C.M. Woodside, "Throughput Calculation for Basic Stochastic Rendezvous Networks", Performance Evaluation, Vol. 9, No. 2, April 1989, pp143-160
- [41] X. Wu, D. McMullan and M. Woodside. “Component Based Performance Prediction”, Proceedings of 6th ICSE Workshop on Component-Based Software Engineering; Automated Reasoning and Prediction, pp13-18, Portland, Oregon, USA, May 3-4, 2003
- [42] R. A. Wyke and A. Watt. “XML Schema Essentials”, Wiley Computer Publishing, 2002
- [43] S. Yacoub. “Performance Analysis of Component-Based Applications”, Proceedings of the Second Software Product Line Conference, pp.299-315, San Diego, CA, USA, August 2002

[44] S. Yacoub, H. Ammar, and A. Mili “Characterizing a Software Component”,
<http://www.sei.cmu.edu/cbs/icse99/papers/34/34.htm>, May 1999

Appendix A XSD Schema for LQML

A.1 XSD Schema for LQN Core (lqn-core.xsd)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- lqn-core is the kernel of lqn sub-model and assembly model-->
  <xsd:element name="lqn-core" type="Lqn-CoreType"/>
  <xsd:complexType name="Lqn-CoreType">
    <xsd:sequence>
      <xsd:element name="processor" type="ProcessorType"
maxOccurs="unbounded"/>
      <xsd:element name="slot" type="SlotType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <!--end of Lqn-CoreType -->
  <!--here goes the schema of SlotType -->
  <xsd:complexType name="SlotType">
    <xsd:sequence>
      <xsd:element name="Interface">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="in-port" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:attribute name="name" type="xsd:string" use="required"/>
                <xsd:attribute name="connect-from">
                  <xsd:simpleType>
                    <xsd:list itemType="xsd:string"/>
                  </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="description" type="xsd:string"/>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="out-port" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:attribute name="name" type="xsd:string" use="required"/>
                <xsd:attribute name="connect-to">
                  <xsd:simpleType>
                    <xsd:list itemType="xsd:string"/>
                  </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="description" type="xsd:string"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="binding" type="BindType" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="bind-target" type="xsd:string" use="required"/>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="replic_num" type="xsd:int"/>
  </xsd:complexType>
  <!--end of SlotType definition-->

```

```

<xsd:complexType name="BindType">
  <xsd:sequence>
    <xsd:element name="parameter" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="value" type="xsd:string" use="required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="processor-binding" minOccurs="0"
maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="source" type="xsd:string" use="required"/>
        <xsd:attribute name="target" type="xsd:string" use="required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="port-binding" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="source" type="xsd:string" use="required"/>
        <xsd:attribute name="target" type="xsd:string" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<!--here goes the schema of ProcessorType-->
<xsd:complexType name="ProcessorType">
  <xsd:sequence>
    <xsd:element name="processor-params" minOccurs="0">
      <xsd:complexType>
        <xsd:attribute name="multiplicity" type="xsd:string" default="1"/>
        <xsd:attribute name="speed-factor" type="xsd:decimal" default="1"/>
        <xsd:attribute name="scheduling" type="SchedulingType"
default="fcfs"/>
        <xsd:attribute name="replication" type="xsd:string" default="1"/>
        <xsd:attribute name="quantum" type="xsd:decimal"/>
        <!--SchedulingType to be defined-->
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="task" type="TaskType" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<!--here goes the schema of SchedulingType-->
<xsd:simpleType name="SchedulingType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="fcfs"/>
    <xsd:enumeration value="ps"/>
    <xsd:enumeration value="pp"/>
    <xsd:enumeration value="r"/>
    <xsd:enumeration value="h"/>
  </xsd:restriction>
</xsd:simpleType>
<!--here goes the schema of TaskSchedulingType-->
<xsd:simpleType name="TaskSchedulingType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="ref"/>
    <xsd:enumeration value="n"/>
  </xsd:restriction>
</xsd:simpleType>

```

```

        <xsd:enumeration value="fcfs"/>
        <xsd:enumeration value="b"/>
        <xsd:enumeration value="P"/>
        <xsd:enumeration value="h"/>
    </xsd:restriction>
</xsd:simpleType>

<!--here goes the schema of TaskType-->
<xsd:complexType name="TaskType">
    <xsd:sequence>
        <xsd:element name="task-params" minOccurs="0">
            <xsd:complexType>
                <xsd:attribute name="mult" type="xsd:string" default="1"/>
                <xsd:attribute name="replication" type="xsd:string" default="1"/>
                <xsd:attribute name="scheduling" type="TaskSchedulingType"
default="n"/>
                <xsd:attribute name="think-time" type="xsd:string" default="0"/>
                <xsd:attribute name="priority" type="xsd:int"/>
                <xsd:attribute name="activity-graph" type="TaskOptionType"/>
                <!--OptionType to be defined Yes|NO-->
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="entry" type="EntryType" maxOccurs="unbounded"/>
        <xsd:element name="service" type="ServiceType" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element name="task-activities" minOccurs="0">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="activity" type="ActivityDefType"
maxOccurs="unbounded"/>
                    <xsd:element name="precedence" type="PrecedenceType"
maxOccurs="unbounded"/>
                    <xsd:element name="reply-entry" maxOccurs="unbounded">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="first-activity">
                                    <xsd:complexType>
                                        <xsd:attribute name="name" type="xsd:string"/>
                                    </xsd:complexType>
                                </xsd:element>
                                <xsd:element name="reply-activity" type="xsd:string"
maxOccurs="unbounded"/>
                            </xsd:sequence>
                            <xsd:attribute name="name" type="xsd:string"/>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:sequence>
</xsd:complexType>
<!-- here goes the schema of TaskOptionType-->
<xsd:simpleType name="TaskOptionType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="YES"/>
    </xsd:restriction>
</xsd:simpleType>

```

```

    <xsd:enumeration value="NO"/>
  </xsd:restriction>
</xsd:simpleType>
<!-- here goes the schema of EntryType-->
<xsd:complexType name="EntryType">
  <xsd:sequence>
    <xsd:element name="entry-params" minOccurs="0">
      <xsd:complexType>
        <xsd:attribute name="open-arrival-rate" type="xsd:string"/>
        <xsd:attribute name="priority" type="xsd:int"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="forwarding" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="dest" type="xsd:string" use="required"/>
        <xsd:attribute name="probability" type="xsd:decimal"
use="required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="entry-activities" minOccurs="0">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="activity" type="ActivityDefType"
maxOccurs="unbounded"/>
          <xsd:element name="activity-sequence" type="Activity-Sequence-
Type" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<!--end of EntryType-->
<xsd:attributeGroup name="MakingCallType">
  <xsd:attribute name="dest" type="xsd:string" use="required"/>
  <xsd:attribute name="calls-mean" type="xsd:string" use="required"/>
  <xsd:attribute name="calls-cvsq" type="xsd:decimal"/>
  <xsd:attribute name="fanout" type="xsd:int" default="1"/>
  <xsd:attribute name="fanin" type="xsd:int" default="1"/>
</xsd:attributeGroup>
<xsd:attributeGroup name="CallListType">
  <xsd:attribute name="dest" type="xsd:string" use="required"/>
  <xsd:attribute name="calls-cvsq" type="xsd:decimal"/>
  <xsd:attribute name="fanout" type="xsd:int" default="1"/>
  <xsd:attribute name="fanin" type="xsd:int" default="1"/>
</xsd:attributeGroup>
<xsd:complexType name="ServiceType">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:simpleType name="CallOrderType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="STOCHASTIC"/>
    <xsd:enumeration value="DETERMINISTIC"/>
    <xsd:enumeration value="LIST"/>
  </xsd:restriction>
</xsd:simpleType>

```

```

<xsd:group name="Call-Group">
  <xsd:sequence>
    <xsd:element name="synch-call" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attributeGroup ref="MakingCallType"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="asynch-call" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attributeGroup ref="MakingCallType"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:group>
<xsd:group name="Call-List-Group">
  <xsd:sequence>
    <xsd:element name="call-list" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="synch-call" minOccurs="0"
maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:attributeGroup ref="CallListType"/>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="asynch-call" minOccurs="0"
maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:attributeGroup ref="CallListType"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:group>
<!--here goes the schema for ActivityDefType-->
<!--ActivityDefType defines activity-params and calls-->
<xsd:complexType name="ActivityDefType">
  <xsd:sequence>
    <xsd:element name="activity-params">
      <xsd:complexType>
        <xsd:attribute name="host-demand-mean" type="xsd:string"
use="required"/>
        <xsd:attribute name="host-demand-cvsq" type="xsd:decimal"/>
        <xsd:attribute name="think-time" type="xsd:decimal"/>
        <xsd:attribute name="max-service-time" type="xsd:decimal"/>
        <xsd:attribute name="call-order" type="CallOrderType"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:choice>
      <xsd:group ref="Call-List-Group"/>
      <xsd:group ref="Call-Group"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>

```

```

</xsd:complexType>
<!--here ends the schema for ActivityDefType.-->
<!--The following definition is for activity precedence relationship --
>
<xsd:complexType name="ActivityType">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="OrListType">
  <xsd:sequence>
    <xsd:element name="activity" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="prob" type="xsd:string" default="1"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AndListType">
  <xsd:sequence>
    <xsd:element name="activity" type="ActivityType"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Activity-Sequence-Type">
  <xsd:sequence>
    <xsd:choice>
      <xsd:group ref="Sequence-Group"/>
      <xsd:group ref="Phase-Group"/>
    </xsd:choice>
    <xsd:element name="reply-activity" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="PhaseOrSequence" use="required"/>
</xsd:complexType>
<xsd:simpleType name="PhaseOrSequence">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="PH1PH2"/>
    <xsd:enumeration value="GRAPH"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:group name="Phase-Group">
  <xsd:sequence>
    <xsd:element name="phase1" type="xsd:string"/>
    <xsd:element name="phase2" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>
<xsd:group name="Sequence-Group">
  <xsd:sequence>
    <xsd:element name="first-activity">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="precedence" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>

```

```

    <xsd:choice>
      <xsd:element name="pre" type="xsd:string"/>
      <xsd:element name="pre-OR" type="OrListType"/>
      <xsd:element name="pre-AND" type="AndListType"/>
      <xsd:element name="pre-LOOP" type="xsd:string"/>
    </xsd:choice>
  </xsd:choice>
  <xsd:choice>
    <xsd:element name="post" type="xsd:string"/>
    <xsd:element name="post-OR" type="OrListType"/>
    <xsd:element name="post-AND" type="AndListType"/>
    <xsd:element name="post-LOOP">
      <xsd:complexType>
        <xsd:attribute name="count" type="xsd:decimal" use="required"/>
        <xsd:attribute name="head" type="xsd:string" use="required"/>
        <xsd:attribute name="end" type="xsd:string" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:group>
<!--here ends the schema of Sequence-Group-->
<!-- here goes the schema of PrecedenceType-->
<xsd:complexType name="PrecedenceType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:element name="pre" type="xsd:string"/>
      <xsd:element name="pre-OR" type="OrListType"/>
      <xsd:element name="pre-AND" type="AndListType"/>
      <xsd:element name="pre-LOOP" type="xsd:string"/>
    </xsd:choice>
    <xsd:choice>
      <xsd:element name="post" type="xsd:string"/>
      <xsd:element name="post-OR" type="OrListType"/>
      <xsd:element name="post-AND" type="AndListType"/>
      <xsd:element name="post-LOOP">
        <xsd:complexType>
          <xsd:attribute name="count" type="xsd:decimal" use="required"/>
          <xsd:attribute name="head" type="xsd:string" use="required"/>
          <xsd:attribute name="end" type="xsd:string" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
<!--here ends the schema of PrecedenceType-->
</xsd:schema>

```


A.2 XSD Schema for LQN Sub-model (lqn-sub.xsd)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="lqn-core.xsd"/>
  <xsd:element name="lqn-submodel" type="Lqn-SubType"/>
  <xsd:complexType name="Lqn-SubType">
    <xsd:sequence>
      <xsd:element name="Interface">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="in-port" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:attribute name="name" type="xsd:string" use="required"/>
                <xsd:attribute name="connect-to" type="xsd:string"/>
                <xsd:attribute name="description" type="xsd:string"/>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="out-port" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:attribute name="name" type="xsd:string" use="required"/>
                <xsd:attribute name="connect-from" type="xsd:string"/>
                <xsd:attribute name="description" type="xsd:string"/>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="Replaceable-Processor" minOccurs="0"
maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:attribute name="name" type="xsd:string" use="required"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Parameter" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="para" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:attribute name="name" type="xsd:string"/>
                <xsd:attribute name="default" type="xsd:string"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="processor" type="ProcessorType"
maxOccurs="unbounded"/>
      <xsd:element name="slot" type="SlotType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:schema>
```

A.3 XSD Schema for LQN Assembly Model (lqn.xsd)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="lqn-core.xsd"/>
  <xsd:element name="lqn-model" type="LqnModelType"/>
  <xsd:complexType name="LqnModelType">
    <xsd:sequence>
      <xsd:element name="run-control" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="para" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:choice>
                  <xsd:sequence>
                    <xsd:element name="start-value" type="xsd:int"/>
                    <xsd:element name="end-value" type="xsd:int"/>
                    <xsd:element name="step-value" type="xsd:int"/>
                  </xsd:sequence>
                  <xsd:sequence>
                    <xsd:element name="value" type="xsd:int"
maxOccurs="unbounded"/>
                  </xsd:sequence>
                </xsd:choice>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="plot-control" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="first-plot">
              <xsd:complexType>
                <xsd:attribute name="variable" type="xsd:string"/>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="plot" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:attribute name="variable" type="xsd:string"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="solver-params" minOccurs="0">
        <xsd:complexType>
          <xsd:attribute name="comment" type="xsd:string" default="LQN
comment"/>
          <xsd:attribute name="conv_val" type="xsd:decimal"
default="0.000001"/>
          <xsd:attribute name="it_limit" type="xsd:int" default="50"/>
          <xsd:attribute name="print_int" type="xsd:int" default="0"/>
          <xsd:attribute name="underrelax_coeff" type="xsd:decimal"
default="0.5"/>
        </xsd:complexType>
      <!--use default values if this is not present-->
    </xsd:sequence>
  </xsd:complexType>

```

```
</xsd:element>
  <xsd:element name="processor" type="ProcessorType"
maxOccurs="unbounded"/>
  <xsd:element name="slot" type="SlotType" minOccurs="0"
maxOccurs="unbounded"/>
</xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="description" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

Appendix B Some Input and Output Documents for LQN Models

B.1 XML Document for SingleMod (SingleMod.xml)

```
<lqn-submodel name="SingleMod" description="an xml version of
OneTaskMod" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn-sub.xsd">
  <Interface>
    <in-port name="p1" connect-to="SingleE2" description="read data from
database"/>
    <in-port name="p2" connect-to="SingleE3" description="update data to
database"/>
    <out-port name="p3" connect-from="SingleE2" description="read request
to file sever"/>
    <out-port name="p4" connect-from="SingleE3" description="update
request to file server"/>
  <Replaceable-Processor name="SinglePr"/>
</Interface>
<Parameter>
  <para name="$SingleT2_mult" default="1"/>
  <para name="$SingleE2_demand" default="1.0"/>
</Parameter>
<processor name="SinglePr">
  <processor-params multiplicity="1"/>
  <task name="SingleT1">
    <task-params mult="1" activity-graph="NO" scheduling="ref"/>
    <entry name="SingleE1">
      <entry-activities>
        <activity name="SingleE1_ph1">
          <activity-params host-demand-mean="1.0"/>
          <synch-call dest="SingleE3" calls-mean="1"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>SingleE1_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
  <task name="SingleT2">
    <task-params mult="$SingleT2_mult" activity-graph="NO"/>
    <entry name="SingleE2">
      <entry-activities>
        <activity name="SingleE2_ph1">
          <activity-params host-demand-mean="$SingleE2_demand"/>
          <synch-call dest="p3" calls-mean="1"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>SingleE2_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry-activities>
  </task>
</processor>
</lqn-submodel>
```

```
</entry>
<entry name="SingleE3">
  <entry-activities>
    <activity name="SingleE3_ph1">
      <activity-params host-demand-mean="1"/>
      <synch-call dest="p4" calls-mean="1"/>
    </activity>
    <activity-sequence type="PH1PH2">
      <phase1>SingleE3_ph1</phase1>
    </activity-sequence>
  </entry-activities>
</entry>
</task>
</processor>
</lqn-submodel>
```

B.2 XML Document for NestedMod (NestedMod.xml)

```
<lqn-submodel name="NestedMod" description="an xml version of
SimpleMod" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn-sub.xsd">
  <Interface>
    <in-port name="SimpleP1" connect-to="SimpleE1" description="SimpleE1
of SimpleT1"/>
    <out-port name="SimpleP2" connect-from="SimpleE3"
description="request from SimpleE3"/>
    <out-port name="SimpleP3" connect-from="SimpleE4"
description="request from SimpleE4"/>
    <Replaceable-Processor name="SimplePr2"/>
  </Interface>
  <Parameter>
    <para name="$SimpleT1_mult" default="1"/>
    <para name="$SingleE2_demand" default="1.0"/>
  </Parameter>
  <processor name="SimplePr1">
    <processor-params multiplicity="1"/>
    <task name="SimpleT1">
      <task-params mult="$SimpleT1_mult" activity-graph="NO"/>
      <entry name="SimpleE1">
        <entry-activities>
          <activity name="SimpleE1_ph1">
            <activity-params host-demand-mean="1"/>
            <synch-call dest="S1.service1" calls-mean="1"/>
            <synch-call dest="S1.service2" calls-mean="2"/>
            <synch-call dest="SimpleE3" calls-mean="1"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>SimpleE1_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
    <task name="SimpleT3">
      <task-params mult="2" activity-graph="NO"/>
      <entry name="SimpleE3">
        <entry-activities>
          <activity name="SimpleE3_ph1">
            <activity-params host-demand-mean="1.5"/>
            <synch-call dest="SimpleP2" calls-mean="2"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>SimpleE3_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
  </processor>
  <processor name="SimplePr2">
    <task name="SimpleT4">
      <task-params activity-graph="NO"/>
    </task>
  </processor>

```

```

<entry name="SimpleE4">
  <entry-activities>
    <activity name="SimpleE4_ph1">
      <activity-params host-demand-mean="5.0"/>
      <synch-call dest="SimpleP3" calls-mean="1"/>
    </activity>
    <activity-sequence type="PH1PH2">
      <phase1>SimpleE4_ph1</phase1>
    </activity-sequence>
  </entry-activities>
</entry>
</task>
</processor>
<slot id="S1" bind-target="SingleMod">
  <Interface>
    <in-port name="service1" connect-from="SimpleE1"/>
    <in-port name="service2" connect-from="SimpleE1"/>
    <out-port name="request1" connect-to="SimpleE3"/>
    <out-port name="request2" connect-to="SimpleE4"/>
  </Interface>
  <binding>
    <!--parameter assignment here for SingleMod-->
    <parameter name="$SingleT2_mult" value="4"/>
    <!--the rest parameters are defined as parameters for NestedMod -->
    <processor-binding source="SinglePr" target="SimplePr1"/>
    <!--source refers to elements in the inner component -->
    <!--target refers to elements in the slot -->
    <port-binding source="p1" target="service1"/>
    <port-binding source="p2" target="service2"/>
    <port-binding source="p3" target="request1"/>
    <port-binding source="p4" target="request2"/>
  </binding>
</slot>
</lqn-submodel>

```

B.3 XML Document for Flattened NestedMod (NestedMod_flat.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<lqn-submodel name="NestedMod" description="an xml version of SimpleMod"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn-sub.xsd">
  <Interface>
    <in-port name="SimpleP1" connect-to="SimpleE1" description="SimpleE1
of SimpleT1"/>
    <out-port name="SimpleP2" connect-from="SimpleE3" description="request
from SimpleE3"/>
    <out-port name="SimpleP3" connect-from="SimpleE4" description="request
from SimpleE4"/>
    <Replaceable-Processor name="SimplePr2"/>
  </Interface>
  <Parameter>
    <para name="$SimpleT1_mult" default="1"/>
    <para name="$SingleE2_demand" default="1.0"/>
  </Parameter>
  <processor name="SimplePr1">
    <processor-params multiplicity="1"/>
    <task name="SimpleT1">
      <task-params mult="$SimpleT1_mult" activity-graph="NO"/>
      <entry name="SimpleE1">
        <entry-activities>
          <activity name="SimpleE1_ph1">
            <activity-params host-demand-mean="1"/>
            <synch-call dest="S1_SingleE2" calls-mean="1"/>
            <synch-call dest="S1_SingleE3" calls-mean="2"/>
            <synch-call dest="SimpleE3" calls-mean="1"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>SimpleE1_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
    <task name="SimpleT3">
      <task-params mult="2" activity-graph="NO"/>
      <entry name="SimpleE3">
        <entry-activities>
          <activity name="SimpleE3_ph1">
            <activity-params host-demand-mean="1.5"/>
            <synch-call dest="SimpleP2" calls-mean="2"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>SimpleE3_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
    <task name="S1_SingleT1">
      <task-params mult="1" activity-graph="NO" scheduling="ref"/>
      <entry name="S1_SingleE1">
```



```

    <entry-activities>
      <activity name="SingleE1_ph1">
        <activity-params host-demand-mean="1.0"/>
        <synch-call dest="S1_SingleE3" calls-mean="1"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>SingleE1_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
<task name="S1_SingleT2">
  <task-params mult="4" activity-graph="NO"/>
  <entry name="S1_SingleE2">
    <entry-activities>
      <activity name="SingleE2_ph1">
        <activity-params host-demand-mean="$$SingleE2_demand"/>
        <synch-call dest="SimpleE3" calls-mean="1"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>SingleE2_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
  <entry name="S1_SingleE3">
    <entry-activities>
      <activity name="SingleE3_ph1">
        <activity-params host-demand-mean="1"/>
        <synch-call dest="SimpleE4" calls-mean="1"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>SingleE3_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
</processor>
<processor name="SimplePr2">
  <task name="SimpleT4">
    <task-params activity-graph="NO"/>
    <entry name="SimpleE4">
      <entry-activities>
        <activity name="SimpleE4_ph1">
          <activity-params host-demand-mean="5.0"/>
          <synch-call dest="SimpleP3" calls-mean="1"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>SimpleE4_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
</processor>
</lqn-submodel>

```

B.4 Output of Transforming SimpleAbl (SimpleAbl.lqn)

```
#This is the output from xml2LQN
G
"This is a test case"
0.00001
50
1
0.8
-1
P 0
p SimplePr1 f
p SimplePr2 f
-1
T 0
t SimpleT1 r SimpleE1 -1 SimplePr1 m 3
t SimpleT3 n SimpleE3 -1 SimplePr1 m 2
t S1_SingleT1 r S1_SingleE1 -1 SimplePr1
t S1_SingleT2 n S1_SingleE2 S1_SingleE3 -1 SimplePr1 m 4
t SimpleT4 n SimpleE4 -1 SimplePr2
-1
E 0
s SimpleE1 1 -1
y SimpleE1 S1_SingleE2 1 -1
y SimpleE1 S1_SingleE3 2 -1
y SimpleE1 SimpleE3 1 -1
s SimpleE3 1.5 -1
s S1_SingleE1 1.0 -1
y S1_SingleE1 S1_SingleE3 1 -1
s S1_SingleE2 5.5 -1
y S1_SingleE2 SimpleE3 1 -1
s S1_SingleE3 1 -1
y S1_SingleE3 SimpleE4 1 -1
s SimpleE4 5.0 -1
-1
```

B.5 XML Document for a Database Model (par-db.xml)

```
<lqn-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn.xsd">
  <solver-params comment="Test case of fork-join" conv_val="0.00001"
it_limit="50" print_int="5" underrelax_coeff="0.9"/>
  <processor name="Pusers">
    <processor-params multiplicity="i"/>
    <task name="Users">
      <task-params mult="2" scheduling="ref" activity-graph="NO"/>
      <entry name="user">
        <entry-activities>
          <activity name="user_ph2">
            <activity-params host-demand-mean="1.0" think-time="50"/>
            <synch-call dest="applic" calls-mean="2.0"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase2>user_ph2</phase2>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
  </processor>
  <processor name="Papplic">
    <task name="Applic">
      <task-params activity-graph="YES"/>
      <entry name="applic">
        <entry-activities>
          <activity name="x1">
            <activity-params host-demand-mean="0.0"/>
          </activity>
          <activity name="a1">
            <activity-params host-demand-mean="0.3"/>
          </activity>
          <activity name="a2">
            <activity-params host-demand-mean="0.1"/>
            <synch-call dest="db1" calls-mean="1.0"/>
            <synch-call dest="db2" calls-mean="0.2"/>
          </activity>
          <activity name="b1">
            <activity-params host-demand-mean="0.05"/>
            <synch-call dest="db1" calls-mean="1.0"/>
          </activity>
          <activity name="b2">
            <activity-params host-demand-mean="0.08"/>
            <synch-call dest="db2" calls-mean="1.0"/>
          </activity>
          <activity name="b3">
            <activity-params host-demand-mean="0.01"/>
            <synch-call dest="db3" calls-mean="1.0"/>
          </activity>
          <activity name="c1">
            <activity-params host-demand-mean="0.1"/>
          </activity>
        </entry-activities>
      </entry>
    </task>
  </processor>
</lqn-model>
```

```

<activity name="y1">
  <activity-params host-demand-mean="0.0"/>
</activity>
<activity-sequence type="GRAPH">
  <first-activity name="x1"/>
  <precedence>
    <pre>x1</pre>
    <post-OR>
      <activity name="a2" prob="0.5"/>
      <activity name="a1" prob="0.5"/>
    </post-OR>
  </precedence>
  <precedence>
    <pre>a1</pre>
    <post-AND>
      <activity name="b1"/>
      <activity name="b2"/>
      <activity name="b3"/>
    </post-AND>
  </precedence>
  <precedence>
    <pre-AND>
      <activity name="b1"/>
      <activity name="b2"/>
      <activity name="b3"/>
    </pre-AND>
    <post>c1</post>
  </precedence>
  <precedence>
    <pre-OR>
      <activity name="c1"/>
      <activity name="a2"/>
    </pre-OR>
    <post>y1</post>
  </precedence>
  <reply-activity>y1</reply-activity>
</activity-sequence>
</entry-activities>
</entry>
</task>
</processor>
<processor name="Pdb1">
  <task name="Db1">
    <task-params mult="2" activity-graph="NO"/>
    <entry name="db1">
      <entry-activities>
        <activity name="db1_ph1">
          <activity-params host-demand-mean="0.04"/>
          <synch-call dest="d1" calls-mean="1.0"/>
          <synch-call dest="d2" calls-mean="1.0"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>db1_ph1</phase1>
          <reply-activity>db1_ph1</reply-activity>
        </activity-sequence>
      </entry-activities>
    </task>
  </processor>

```

```

    </entry>
  </task>
</processor>
<processor name="Pdb2">
  <task name="Db2">
    <task-params mult="3" activity-graph="NO"/>
    <entry name="db2">
      <entry-activities>
        <activity name="db2_ph1">
          <activity-params host-demand-mean="0.04"/>
          <synch-call dest="d3" calls-mean="1.0"/>
          <synch-call dest="d4" calls-mean="1.0"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>db2_ph1</phase1>
          <reply-activity>db2_ph1</reply-activity>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
</processor>
<processor name="Pdb3">
  <task name="Db3">
    <task-params activity-graph="YES"/>
    <entry name="db3">
      <entry-activities>
        <activity name="a1">
          <activity-params host-demand-mean="0.01"/>
        </activity>
        <activity name="b1">
          <activity-params host-demand-mean="0.01"/>
          <synch-call dest="d5" calls-mean="1.0"/>
        </activity>
        <activity name="b2">
          <activity-params host-demand-mean="0.01"/>
          <synch-call dest="d6" calls-mean="1.0"/>
        </activity>
        <activity name="b3">
          <activity-params host-demand-mean="0.01"/>
          <synch-call dest="d7" calls-mean="1.0"/>
        </activity>
        <activity name="b4">
          <activity-params host-demand-mean="0.01"/>
          <synch-call dest="d8" calls-mean="1.0"/>
        </activity>
        <activity name="c1">
          <activity-params host-demand-mean="0.01"/>
        </activity>
      <activity-sequence type="GRAPH">
        <first-activity name="a1"/>
        <precedence>
          <pre>a1</pre>
          <post-AND>
            <activity name="b1"/>
            <activity name="b2"/>
            <activity name="b3"/>
          </post-AND>
        </precedence>
      </activity-sequence>
    </entry>
  </task>
</processor>

```

```

        <activity name="b4"/>
    </post-AND>
</precedence>
</precedence>
    <pre-AND>
        <activity name="b1"/>
        <activity name="b2"/>
        <activity name="b3"/>
        <activity name="b4"/>
    </pre-AND>
    <post>c1</post>
</precedence>
    <reply-activity>c1</reply-activity>
</activity-sequence>
</entry-activities>
</entry>
</task>
</processor>
<processor name="Pd1">
    <task name="D1">
        <task-params activity-graph="NO"/>
        <entry name="d1">
            <entry-activities>
                <activity name="d1_ph1">
                    <activity-params host-demand-mean="0.03"/>
                </activity>
                <activity-sequence type="PH1PH2">
                    <phase1>d1_ph1</phase1>
                    <reply-activity>d1_ph1</reply-activity>
                </activity-sequence>
            </entry-activities>
        </entry>
    </task>
</processor>
<processor name="Pd2">
    <task name="D2">
        <task-params activity-graph="NO"/>
        <entry name="d2">
            <entry-activities>
                <activity name="d2_ph1">
                    <activity-params host-demand-mean="0.03"/>
                </activity>
                <activity-sequence type="PH1PH2">
                    <phase1>d2_ph1</phase1>
                    <reply-activity>d2_ph1</reply-activity>
                </activity-sequence>
            </entry-activities>
        </entry>
    </task>
</processor>
<processor name="Pd3">
    <task name="D3">
        <task-params activity-graph="NO"/>
        <entry name="d3">
            <entry-activities>
                <activity name="d3_ph1">

```

```

        <activity-params host-demand-mean="0.03"/>
    </activity>
    <activity-sequence type="PH1PH2">
        <phase1>d3_ph1</phase1>
        <reply-activity>d3_ph1</reply-activity>
    </activity-sequence>
</entry-activities>
</entry>
</task>
</processor>
<processor name="Pd4">
    <task name="D4">
        <task-params activity-graph="NO"/>
        <entry name="d4">
            <entry-activities>
                <activity name="d4_ph1">
                    <activity-params host-demand-mean="0.03"/>
                </activity>
                <activity-sequence type="PH1PH2">
                    <phase1>d4_ph1</phase1>
                    <reply-activity>d4_ph1</reply-activity>
                </activity-sequence>
            </entry-activities>
        </entry>
    </task>
</processor>
<processor name="Pd5">
    <task name="D5">
        <task-params activity-graph="NO"/>
        <entry name="d5">
            <entry-activities>
                <activity name="d5_ph1">
                    <activity-params host-demand-mean="0.03"/>
                </activity>
                <activity-sequence type="PH1PH2">
                    <phase1>d5_ph1</phase1>
                    <reply-activity>d5_ph1</reply-activity>
                </activity-sequence>
            </entry-activities>
        </entry>
    </task>
</processor>
<processor name="Pd6">
    <task name="D6">
        <entry name="d6">
            <entry-activities>
                <activity name="d6_ph1">
                    <activity-params host-demand-mean="0.03"/>
                </activity>
                <activity-sequence type="PH1PH2">
                    <phase1>d6_ph1</phase1>
                    <reply-activity>d6_ph1</reply-activity>
                </activity-sequence>
            </entry-activities>
        </entry>
    </task>

```

```

</processor>
<processor name="Pd7">
  <task name="D7">
    <entry name="d7">
      <entry-activities>
        <activity name="d7_ph1">
          <activity-params host-demand-mean="0.03"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>d7_ph1</phase1>
          <reply-activity>d7_ph1</reply-activity>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
</processor>
<processor name="Pd8">
  <task name="D8">
    <entry name="d8">
      <entry-activities>
        <activity name="d8_ph1">
          <activity-params host-demand-mean="0.03"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>d8_ph1</phase1>
          <reply-activity>d8_ph1</reply-activity>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
</processor>
</lqn-model>

```


B.6 Output of Transforming par-db.xml (par-db.lqn)

```
#This is the output from xml2LQN
G
"Test case of fork-join"
0.00001
50
5
0.9
-1
P 0
p Pusers f i
p Papplic f
p Pdb1 f
p Pdb2 f
p Pdb3 f
p Pd1 f
p Pd2 f
p Pd3 f
p Pd4 f
p Pd5 f
p Pd6 f
p Pd7 f
p Pd8 f
-1
T 0
t Users r user -1 Pusers m 2
t Applic n applic -1 Papplic
t Db1 n db1 -1 Pdb1 m 2
t Db2 n db2 -1 Pdb2 m 3
t Db3 n db3 -1 Pdb3
t D1 n d1 -1 Pd1
t D2 n d2 -1 Pd2
t D3 n d3 -1 Pd3
t D4 n d4 -1 Pd4
t D5 n d5 -1 Pd5
t D6 n d6 -1 Pd6
t D7 n d7 -1 Pd7
t D8 n d8 -1 Pd8
-1
E 0
s user 0 1.0 -1
Z user 0 50 -1
y user applic 0 2.0 -1
A applic x1
s db1 0.04 -1
y db1 d1 1.0 -1
y db1 d2 1.0 -1
s db2 0.04 -1
y db2 d3 1.0 -1
y db2 d4 1.0 -1
A db3 a1
s d1 0.03 -1
s d2 0.03 -1
```

```

s d3 0.03 -1
s d4 0.03 -1
s d5 0.03 -1
s d6 0.03 -1
s d7 0.03 -1
s d8 0.03 -1
-1
A Applic
s x1 0.0
s a1 0.3
s a2 0.1
y a2 db1 1.0
y a2 db2 0.2
s b1 0.05
y b1 db1 1.0
s b2 0.08
y b2 db2 1.0
s b3 0.01
y b3 db3 1.0
s c1 0.1
s y1 0.0
:
x1 -> (0.5)a2+(0.5)a1;
a1 -> b1&b2&b3;
b1&b2&b3->c1;
c1+a2->y1;
y1[applic]
-1
A Db3
s a1 0.01
s b1 0.01
y b1 d5 1.0
s b2 0.01
y b2 d6 1.0
s b3 0.01
y b3 d7 1.0
s b4 0.01
y b4 d8 1.0
s c1 0.01
:
a1 -> b1&b2&b3&b4;
b1&b2&b3&b4->c1;
c1[db3]
-1

```

B.7 XML Document for an LQN Model that Has Task Activities (task-act.xml)

```
<lqn-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn.xsd">
  <solver-params comment="Test case of task-activities"
conv_val="0.00001" it_limit="50" print_int="5" underrelax_coeff="0.9"/>
  <processor name="P1">
    <processor-params multiplicity="i"/>
    <task name="t0">
      <task-params mult="2" scheduling="ref" activity-graph="NO"/>
      <entry name="user">
        <entry-activities>
          <activity name="user_ph2">
            <activity-params host-demand-mean="1.0" think-time="50"/>
            <synch-call dest="e1" calls-mean="1.0"/>
            <synch-call dest="e2" calls-mean="1.0"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase2>user_ph2</phase2>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
    <task name="t1">
      <entry name="e1">
        <entry-activities>
          <activity name="e1_ph1">
            <activity-params host-demand-mean="1.5"/>
            <synch-call dest="e3" calls-mean="1.0"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>e1_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
    <task name="t2">
      <entry name="e2">
        <entry-activities>
          <activity name="e2_ph1">
            <activity-params host-demand-mean="0.5"/>
            <synch-call dest="e4" calls-mean="1.0"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>e2_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
  </processor>
  <processor name="P2">
    <task name="t3">
      <task-params activity-graph="YES"/>
      <entry name="e3"/>
      <entry name="e4"/>
      <task-activities>
```

```
<activity name="a1">
  <activity-params host-demand-mean="1.0"/>
</activity>
<activity name="a2">
  <activity-params host-demand-mean="1.0"/>
</activity>
<activity name="c1">
  <activity-params host-demand-mean="1.0"/>
</activity>
<precedence>
  <pre-AND>
    <activity name="a1"/>
    <activity name="a2"/>
  </pre-AND>
  <post>c1</post>
</precedence>
<reply-entry name="e3">
  <first-activity name="a1"/>
  <reply-activity>c1</reply-activity>
</reply-entry>
<reply-entry name="e4">
  <first-activity name="a2"/>
  <reply-activity>c1</reply-activity>
</reply-entry>
</task-activities>
</task>
</processor>
</lqn-model>
```

B.8 Output Model from Transforming task-act.xml (task-act.lqn)

```
#This is the output from xml2LQN
G
"Test case of task-activities"
0.00001
50
5
0.9
-1
P 0
p P1 f i
p P2 f
-1
T 0
t t0 r user -1 P1 m 2
t t1 n e1 -1 P1
t t2 n e2 -1 P1
t t3 n e3 e4 -1 P2
-1
E 0
s user 0 1.0 -1
Z user 0 50 -1
y user e1 0 1.0 -1
y user e2 0 1.0 -1
s e1 1.5 -1
y e1 e3 1.0 -1
s e2 0.5 -1
y e2 e4 1.0 -1
A e3 a1
A e4 a2
-1
A t3
s a1 1.0
s a2 1.0
s c1 1.0
:
a1&a2->c1;
c1[e3,e4]
-1
```

Appendix C Source Files for LQN Models in the Case Study

C.1 XML Document for MISAssemble (MISAssemble.xml)

```
<lqn-model name="MISAssemble" description="An Assembly model for MIS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn.xsd">
  <solver-params comment="An Assembly model for MIS" conv_val="0.00001"
it_limit="100" print_int="1" underrelax_coeff="0.9"/>
  <processor name="ClientP">
    <processor-params multiplicity="i"/>
    <task name="Client">
      <task-params mult="10" scheduling="ref"/>
      <entry name="Request">
        <entry-activities>
          <activity name="Request_ph2">
            <activity-params host-demand-mean="0.02"/>
            <synch-call dest="RSRequest" calls-mean="5"/>
            <synch-call dest="VSRequest" calls-mean="15"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase2>Request_ph2</phase2>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
    <task name="RSClient">
      <task-params mult="i"/>
      <entry name="RSRequest">
        <entry-activities>
          <activity name="RSRequest_ph1">
            <activity-params host-demand-mean="0"/>
            <synch-call dest="AcceptRS" calls-mean="1"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>RSRequest_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
    <task name="VSClient">
      <task-params mult="i"/>
      <entry name="VSRequest">
        <entry-activities>
          <activity name="VSRequest_ph1">
            <activity-params host-demand-mean="0"/>
            <synch-call dest="AcceptVS" calls-mean="1"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>VSRequest_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
    </task>
  </processor>
</lqn-model>
```

```

</processor>
<processor name="WebP">
  <task name="WebServer">
    <task-params mult="i" activity-graph="NO"/>
    <entry name="AcceptRS">
      <entry-activities>
        <activity name="AcceptRS_ph1">
          <activity-params host-demand-mean="0.03"/>
          <synch-call dest="AppS.CompuRS" calls-mean="1"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>AcceptRS_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
    <entry name="AcceptVS">
      <entry-activities>
        <activity name="AcceptVS_ph1">
          <activity-params host-demand-mean="0.025"/>
          <synch-call dest="AppS.CompuVS" calls-mean="1"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>AcceptVS_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
</processor>
<processor name="CachP">
  <task name="CacheInfo">
    <entry name="cache">
      <entry-activities>
        <activity name="cache_ph1">
          <activity-params host-demand-mean="0.04"/>
          <synch-call dest="CachOP" calls-mean="1"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>cache_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
  </task>
</processor>
<processor name="DBP">
  <task name="DBServer">
    <entry name="BigOP">
      <entry-activities>
        <activity name="BigOP_ph1">
          <activity-params host-demand-mean="4"/>
        </activity>
        <activity-sequence type="PH1PH2">
          <phase1>BigOP_ph1</phase1>
        </activity-sequence>
      </entry-activities>
    </entry>
    <entry name="SmallOP">

```

```

    <entry-activities>
      <activity name="SmallOP_ph1">
        <activity-params host-demand-mean="0.5"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>SmallOP_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
  <entry name="CachOP">
    <entry-activities>
      <activity name="CachOP_ph1">
        <activity-params host-demand-mean="0.4"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>CachOP_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
</processor>
<slot id="AppS" bind-target="AppComp">
  <Interface>
    <in-port name="CompuRS" connect-from="AcceptRS"/>
    <in-port name="CompuVS" connect-from="AcceptVS"/>
    <out-port name="bigReq" connect-to="BigOP"/>
    <out-port name="smallReq" connect-to="SmallOP"/>
    <out-port name="cachReq" connect-to="cache"/>
  </Interface>
  <binding>
    <!--parameter assignment here for AppComp-->
    <parameter name="$AsP_num" value="2"/>
    <!--processor-binding source="AsP" target="CachP"/-->
    <!--source refers to elements in the inner component -->
    <!--target refers to elements in the slot -->
    <port-binding source="RSctl" target="CompuRS"/>
    <port-binding source="VSctl" target="CompuVS"/>
    <port-binding source="bigReq" target="bigReq"/>
    <port-binding source="smallReq" target="smallReq"/>
    <port-binding source="cachReq" target="cachReq"/>
  </binding>
</slot>
</lqn-model>

```


C.2 XML Document for AppComp (AppComp.xml)

```
<lqn-submodel name="AppComp" description="A submodel for Application
Server in XML" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="lqn-sub.xsd">
  <Interface>
    <in-port name="RSctl" connect-to="SchedRS" description="Reporting
request controller"/>
    <in-port name="VSctl" connect-to="SchedVS" description="Viewing
request controller"/>
    <out-port name="bigReq" connect-from="BigRep1 BigRep2"
description="big report request"/>
    <out-port name="smallReq" connect-from="SmallR1 SmallR2"
description="samll report request"/>
    <out-port name="cachReq" connect-from="View GenReport"
description="obtaining results from cache"/>
    <Replaceable-Processor name="AsP"/>
  </Interface>
  <Parameter>
    <para name="$AsP_num" default="1"/>
  </Parameter>
  <processor name="ScheduP">
    <processor-params multiplicity="i"/>
    <task name="Scheduler">
      <task-params mult="i" activity-graph="NO"/>
      <entry name="SchedRS">
        <forwarding dest="DispRS" probability="1"/>
      </entry>
      <entry name="SchedVS">
        <forwarding dest="DispVS" probability="1"/>
      </entry>
    </task>
  </processor>
  <processor name="AsP">
    <processor-params multiplicity="$AsP_num"/>
    <task name="DSP">
      <task-params mult="i"/>
      <entry name="DispRS">
        <entry-activities>
          <activity name="DispRS_ph1">
            <activity-params host-demand-mean="1.0"/>
            <synch-call dest="SmallR1" calls-mean="0.4"/>
            <synch-call dest="SmallR2" calls-mean="0.4"/>
            <synch-call dest="BigRep1" calls-mean="0.1"/>
            <synch-call dest="BigRep2" calls-mean="0.1"/>
          </activity>
          <activity-sequence type="PH1PH2">
            <phase1>DispRS_ph1</phase1>
          </activity-sequence>
        </entry-activities>
      </entry>
      <entry name="DispVS">
        <entry-activities>
          <activity name="DispVS_ph1">
```

```

        <activity-params host-demand-mean="1.0"/>
        <synch-call dest="View" calls-mean="1"/>
    </activity>
    <activity-sequence type="PH1PH2">
        <phase1>DispVS_ph1</phase1>
    </activity-sequence>
</entry-activities>
</entry>
</task>
<task name="SmallReport1">
    <entry name="SmallR1">
        <entry-activities>
            <activity name="SmallR1_ph1">
                <activity-params host-demand-mean="0.03"/>
                <synch-call dest="smallReq" calls-mean="1"/>
            </activity>
            <activity-sequence type="PH1PH2">
                <phase1>SmallR1_ph1</phase1>
            </activity-sequence>
        </entry-activities>
    </entry>
</task>
<task name="SmallReport2">
    <entry name="SmallR2">
        <entry-activities>
            <activity name="SmallR2_ph1">
                <activity-params host-demand-mean="0.03"/>
                <synch-call dest="smallReq" calls-mean="1"/>
            </activity>
            <activity-sequence type="PH1PH2">
                <phase1>SmallR2_ph1</phase1>
            </activity-sequence>
        </entry-activities>
    </entry>
</task>
<task name="BigReport1">
    <entry name="BigRep1">
        <entry-activities>
            <activity name="BigRep1_ph1">
                <activity-params host-demand-mean="0.3"/>
                <synch-call dest="bigReq" calls-mean="1"/>
                <synch-call dest="GenReport" calls-mean="1"/>
            </activity>
            <activity-sequence type="PH1PH2">
                <phase1>BigRep1_ph1</phase1>
            </activity-sequence>
        </entry-activities>
    </entry>
</task>
<task name="BigReport2">
    <entry name="BigRep2">
        <entry-activities>
            <activity name="BigRep2_ph1">
                <activity-params host-demand-mean="0.3"/>
                <synch-call dest="bigReq" calls-mean="1"/>
                <synch-call dest="GenReport" calls-mean="1"/>
            </activity>
        </entry-activities>
    </entry>
</task>

```

```

    </activity>
    <activity-sequence type="PH1PH2">
      <phase1>BigRep2_ph1</phase1>
    </activity-sequence>
  </entry-activities>
</entry>
</task>
<task name="ViewData">
  <entry name="View">
    <entry-activities>
      <activity name="View_ph1">
        <activity-params host-demand-mean="0.06"/>
        <synch-call dest="cachReq" calls-mean="1"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>View_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
<task name="ReportGen">
  <entry name="GenReport">
    <entry-activities>
      <activity name="GenReport_ph1">
        <activity-params host-demand-mean="0.26"/>
        <synch-call dest="cachReq" calls-mean="2"/>
      </activity>
      <activity-sequence type="PH1PH2">
        <phase1>GenReport_ph1</phase1>
      </activity-sequence>
    </entry-activities>
  </entry>
</task>
</processor>
</lqn-submodel>

```

C.3 Resulting Model in LQN Language (MISAssemble.lqn)

```
#This is the output from xml2LQN
G
"An Assembly model for MIS"
0.00001
100
1
0.9
-1
P 0
p ClientP f i
p WebP f
p CachP f
p DBP f
p AppS_ScheduP f i
p AppS_AsP f m 2
-1
T 0
t Client r Request -1 ClientP m 10
t RSCClient n RSRequest -1 ClientP i
t VSClient n VSRequest -1 ClientP i
t WebServer n AcceptRS AcceptVS -1 WebP i
t CacheInfo n cache -1 CachP
t DBServer n BigOP SmallOP CachOP -1 DBP
t AppS_Scheduler n AppS_SchedRS AppS_SchedVS -1 AppS_ScheduP i
t AppS_DSP n AppS_DisprRS AppS_DisprVS -1 AppS_AsP i
t AppS_SmallReport1 n AppS_SmallR1 -1 AppS_AsP
t AppS_SmallReport2 n AppS_SmallR2 -1 AppS_AsP
t AppS_BigReport1 n AppS_BigRep1 -1 AppS_AsP
t AppS_BigReport2 n AppS_BigRep2 -1 AppS_AsP
t AppS_ViewData n AppS_View -1 AppS_AsP
t AppS_ReportGen n AppS_GenReport -1 AppS_AsP
-1
E 0
s Request 0 0.02 -1
y Request RSRequest 0 5 -1
y Request VSRequest 0 15 -1
s RSRequest 0 -1
y RSRequest AcceptRS 1 -1
s VSRequest 0 -1
y VSRequest AcceptVS 1 -1
s AcceptRS 0.03 -1
y AcceptRS AppS_SchedRS 1 -1
s AcceptVS 0.025 -1
y AcceptVS AppS_SchedVS 1 -1
s cache 0.04 -1
y cache CachOP 1 -1
s BigOP 4 -1
s SmallOP 0.5 -1
s CachOP 0.4 -1
F AppS_SchedRS AppS_DisprRS 1 -1
F AppS_SchedVS AppS_DisprVS 1 -1
s AppS_DisprRS 1.0 -1
```

```
y AppS_Disprs AppS_SmallR1 0.4 -1
y AppS_Disprs AppS_SmallR2 0.4 -1
y AppS_Disprs AppS_BigRep1 0.1 -1
y AppS_Disprs AppS_BigRep2 0.1 -1
s AppS_Disprs 1.0 -1
y AppS_Disprs AppS_View 1 -1
s AppS_SmallR1 0.03 -1
y AppS_SmallR1 SmallOP 1 -1
s AppS_SmallR2 0.03 -1
y AppS_SmallR2 SmallOP 1 -1
s AppS_BigRep1 0.3 -1
y AppS_BigRep1 BigOP 1 -1
y AppS_BigRep1 AppS_GenReport 1 -1
s AppS_BigRep2 0.3 -1
y AppS_BigRep2 BigOP 1 -1
y AppS_BigRep2 AppS_GenReport 1 -1
s AppS_View 0.06 -1
y AppS_View cache 1 -1
s AppS_GenReport 0.26 -1
y AppS_GenReport cache 2 -1
-1
```