Time/Performance Budgeting for Software Designs

By

Khalid H. Siddiqui

A thesis is submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements of the degree of

Master of Engineering

Ottawa-Carleton Institute of Electrical Engineering

Faculty of Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

November 12, 2001

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

Time/Performance Budgeting for Software Designs

Submitted by Khalid H. Siddiqui, B.Eng. (EE) in partial fulfillment of the requirements for the degree of Master of Engineering

Chair, Department of Systems and Computer Engineering

Thesis Supervisor

Carleton University

November 12, 2001

To My Family Members in

Pakistan, Lebanon, Saudi Arabia

and Canada: If everyone had

family members like mine,

the world would be a much better place.

Abstract

Budgeting of time or system resources is a possible approach to achieving performance targets for communications software. The role of any kind of budgeting is to allocate the available resources to different activities or different parts of a system. Here, the total permitted time to complete a response is being divided among the activities that must be completed. However, because allowances must be made for environment delays, overheads, latencies and so on, a performance model is essential. The model integrates the various sources of delay, including the execution delay of each activity executed by the system. Activities are given a budget for resources such as CPU time, which a developer can track for the activity in isolation, using well-known tools such as profilers.

Acknowledgements

First of all, I am very thankful to my lord and best friend, my lovely Allah, Who gave me the courage, the guidance and the love to complete this research. At this moment, I will not forget the ideal man of the world and most beautiful and respectable personality for whom Allah created the whole universe i.e. Prophet Mohammed (Peace Be Upon Him).

My mentor and the ocean of the knowledge, Professor C. M. Woodside, without your excellent guidance I wouldn’t have completed this research successfully. I like few people of the world and you are one of them.

My lovely parents, with your prayers and love I made your dream to come true. My all-family members from Baba Bhai to little Nadeen, I wouldn’t have finished my work without your prayers and love.

Tauseef, my dearest friend and Bhabi (Ooji), the best example of the friendships. Tauseef, I can’t forget the day when my two brothers (Hassan and Mohammed) left Canada to see family members and we explored Ottawa on December 24, 1999 in –35 degrees Celcius.

Gunter Musbachar and Tom Gray of Mitel Corporation who provided me great professional guidance with their experience to make this research successful.

All the members of RADS lab (Dorin, Mohammed, Olivia and Jun Feng), who provided an excellent environment for the work, lunches and regular group meetings.

Last but not least, I would like to dedicate my research to all of my family members, especially my lovely sister Narmeen. I pray Allah to give her eyes back so that she would be able to see her brother and all the family members and have an enjoyable life.

Table of Contents

List Of Figures

List Of Tables

Glossary of Terms

DSM	Distributed Shared Memory
IPC	Inter Process Communication
JDK	Java Development Kit
JVM	Java Virtual Machine
LQN	Layered Queuing Network
LQNS	Layered Queuing Network Solver
MCNT	Multicast Network Task
NSS	Network Subsystem
ORB	Object Request Broker
POTS	Plain Old Telephone System
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SDL	Specification and Description Language
SNCT	Single Network Component Task
SRST	Shared Resource Server Task
SPE	Software Performance Engineering
UCM	Use Case Map
UML	Unified Modeling Language

1.0 Introduction

1.1 Motivation

When designing the software architecture of a complex distributed software system, an important consideration is the performance of the resulting system. The abilities to estimate the future performance of a large and complex distributed software system at design time, and to iteratively refine these estimates can significantly reduce the overall cost and time.

Different approaches to performance analysis are available to achieve performance targets of any software system. These approaches involve steps to ensure that a software system meets its performance objectives. Satisfactory performance achievement is a significant problem for any software system. This thesis describes a new approach to performance, which is applied throughout the lifecycle of software system.

What is Performance Analysis?

IEEE and C. U. Smith give two useful definitions of performance.

The definition of performance given in the IEEE Standard Glossary of Software Engineering Terminology is:

“The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage” [IEEE-610.12].

Smith describes the increasing importance of performance analysis of software systems:

“Performance refers to system responsiveness: either the time required to respond to specific events, or the number of events processed in a given time interval. For traditional information systems, performance considerations are associated with usability issue such as response time for user commands. For ‘soft’ real-time systems, performance considerations relate to the ability to adequately handle the external load placed on the system. In ‘hard’ real-time systems, performance concerns can become correctness issues. In these systems, failure to meet deadlines or throughput requirements is equivalent to producing incorrect results” [Smith93].

1.1.1 Performance Analysis Approaches

Smith documents several instances where such an analysis would be useful. The analysis may make use of fix-it-later versus predictive, early modeling approaches.

The first and simplest approach of performance analysis is ‘fix-it-later’. This is a very common and risky approach. It is applied after the design and development of a software system. The goal of this approach is to save time, expense, and maintenance cost. It advocates the correctness of software and defers performance consideration to the integration-testing phase. If performance problems are detected then corrective measures are applied in the form of additional hardware or software tuning/optimization. **The most serious flaw is, because it is late, it is costly and may have limited capability. It is might be possible that corrective measures would not correct such a performance problem at all.**

The second approach to performance analysis called Software Performance Engineering (SPE) is performance modeling by C.U. Smith. Smith introduces the approach of performance prediction by modeling. This technique is used in the early stages of a system design. It pinpoints and rectifies key problem areas before system development and implementation. It is less laborious and more flexible than taking post-development measurements because the model operates on the key aspects of the system [Smith01].

The third approach to performance analysis is a possibility midway between these two approaches is to measure different components and prototypes of the system as development progress. Instead of a building a model, use common sense. **The main drawback of this technique is lack of predictive power.**

In summary, all three approaches mentioned in this section can be applied at any stage of software development. However, a major drawback to these approaches is that performance analysis is done on the overall system where as the approach discussed in this thesis enables the analyst to do a performance analysis at the component as well as the overall system level throughout the lifecycle of a software system. If a software system consists of many software components and some of the components are already used in some other systems and the performance of these components are already analyzed then this can easily speed up the performance analysis process for the new system. The performance analysis of the new system will be based on the budgeted values of the existing software components with new budgets for new software components. The budgeted values of new components will be based on intuition, experience and sensitivity analysis. This supports the idea of time and performance budgeting of any software system.

1.2 Objective

The objective of the thesis work is a budgeting approach using a “divide and conquer” policy. First divides the problem into parts that correspond to system responsibilities, and to the work of separate developer groups, then estimates (using a model) the overall performance that will result if all the budgets are met. The model also estimates the effects of contention, overheads and latencies.

This division gives developer targets, which he or she can understand and attempt to achieve, without having to understand the entire system. The understanding of the entire system is focused where it should be, at the managerial level, perhaps with some specialist participation in working with the model. The tools minimize the effort on a single project, since information about components and the execution environment can be stored, and re-used in later work.

The objective of this research is:

To describe a performance analysis approach for the development of software systems, which is based on budgets or estimated figures for the resource demands of all the parts and operations of the system before the design and development. The budget or estimated figure may be based on prediction, intuition or results of past experiments. The key elements of this approach are the planning of budgets for the resource demands of each the parts and operations of system and validation check for the required performance.

1.3 Overview of the thesis work

The primary effort in this study is the development of an approach, which is based on UCMs, resource demand budgets, system completions, and derived performance models. UCM notation is used to describe design specification. The UCM captures the system specification as a set of scenarios, at a suitable level of abstraction, and defines software components and responsibilities. The use of UCMs is described in Chapter 2. Budgeted value estimation is based on guess, intuition, prediction and sensitivity analysis. The term “completion” defines “completion” for a system in the form of additional hardware and software components or different performance parameters. These “completions” can be defined in the UCM domain as well as in the form of additional “completions” in LQN domain. This leads to a suitable performance model. The performance model can be constructed by hand or can be obtained by some automation. An automated tool, UCM2LQN converter, is available for UCM to LQN conversion. This model is solved and results are analyzed to check whether the budgeted values achieve satisfactory performance goals or not. If the results are not satisfactory then it is required to make some changes in parameters or in the architecture at the hardware or software level as. The road map elaborating these analysis steps is shown in Figure 3.1.

A challenge is to create the budget values for the demands on individual responsibilities. It can be done in the same way that financial budget values are created under uncertainty, using experience and a reserve for contingencies [McNamara99]. To assist the budgeting, prototyping experiments can be done to refine the estimates, and the demand value can be tracked during development. Model experiments can be done to estimates the effects of different kinds of overshoots. If a particular responsibility needs more time, taking time from other responsibilities, changing the design, or boosting the hardware capability can accommodate it.

It is essential to use a model because we are analyzing a system, which has not yet been built. The challenge for practical modeling is to reduce the effort and cost of building a model, which is done here by model generation based on Use Case Maps (UCM) described in Chapter 2. The generation process is mostly automated, apart from steps to define the various kinds of “completions”. These include descriptions of software components that are to be included “as is”, of protocols, middleware, file systems and operating systems overheads, of hardware properties, and of competing workloads that may impact the performance.

Performance models are the natural way to provide predictions for a range of environmental conditions or design alternatives. These predictions can be used to detect problems and suggest corrections. The Layered Queuing Network (LQN) model is a kind of queuing model based on the UCM and its parameters, and on the expected execution environment. In defining the environment, additional components are added to the system, and these may be held in a component sub-model library. The Layered Queuing Network Solver (LQNS) offers a variety of output measures, which need to be tailored to the analysis, to make it easy to use.

The approach integrates work from two different research areas: Software Performance Engineering approaches and distributed communication software systems engineering.

1.4 Contributions

The main contributions of this work are:

- *The approach of Performance Budgeting process* with Road Map described in Chapter 3. Performance Budgeting provides different steps to carry out performance analysis before the development and implementation of any software system to meet performance goal.
- *Definition of the term "completion"* to overcome the incompleteness of a design specification. We suggest and provide ways to add missing components and parameters to an incomplete specification of a software design. An incomplete specification shows the thinking of a software designer at some early stage of development. The specification is very abstract and may omit some necessary details, which are important for a suitable performance model.
- *Demonstration of the "completion"* described in Chapter 5. The development of four network filters provides automation to the "completion" at LQN level only (model level). Different network components are incorporated in different examples in the form of a single LQN task representing a simple message passing network component (a single network component), a generalized protocol suite (LQN network subsystem), a shared resource server LQN task and a multicast network LQN task.

- *Measurements and modeling of tuple spaces*. Measurements are done on J-Spaces (J-Space Technologies Inc.) [JSpaces] and the performance model of J-Spaces is constructed and analyzed described in Chapter 6.
- *Two end-to-end Case Studies* on UCM design examples from industry, described in Chapter 7.

The significance of these contributions is:

- *They can be used with several software development methodologies,* and support an iterative software development cycle.
- *They provide component based performance analysis as well as overall system based performance analysis.* Every component is given a separate budget. If the budgets are met then performance goals are achieved. If not then derive the equipment performance required to achieve for given budgeted values to obtain satisfactory performance. For example, what processor speeds will make it work? Hardware or software architectural adjustment is another option to achieve performance targets. This permits the analyst to focus on the principle of Software Performance Engineering rather than model building.
- *They provide a framework to allow for the reuse of performance information*
- *A process for tracking and correcting performance attributes* of designer driven the project.

1.5 Thesis Outline

Chapter 2 provides background material for the topics, which are fundamental developing the performance budgeting approach.

Chapter 3 is an overview of the performance budgeting approach.

Chapter 4 describes the "c*ompletion*" to complete an incomplete specification of a software design and a number example to elaborate the "c*ompletion*".

Chapter 5 describes the designs and implementations of four network component filters: Single Network Component Task (SNCT) filter, Network Subsystem (NSS) filter, Shared Resource Server Task (SRST) filter and Multicast Network Task (MCNT) filter, to demonstrate the "c*ompletion*" in the LQN domain (model level) only.

Chapter 6 describes measurements and performances modeling of commercially available tuple spaces called J-Spaces by J-Space Technology Inc [JSpaces].

Chapter 7 introduces two case studies, a UCM Pattern Example based on SX-2000 UCM design style of Mitel Networks Inc. and Plain Old Telephone System (POTS) UCM Design.

Chapter 8 describes architecture adjustment using Plain Old Telephone System (POTS) UCM Design.

Chapter 9 provides conclusions and recommendation for future research where this work can be extended.

1.6 Summarry

This chapter provided a brief overview of the following:

1. Motivation for the research work including various approaches to performance analysis,
2. The objective of the research,
3. The overview of the thesis,
4. The contributions to the research, and
5. The outline of each chapter of the thesis.

2.0 Background

2.1 Introduction

This chapter will describe background material for the understanding of this thesis. This includes descriptions of Software Performance Engineering (SPE), Use Case Maps (UCMs), Layered Queuing Network (LQN) Modeling, Performance Budgeting, automated tool support for Performance Budgeting, and tuple spaces.

2.2 Software Performance Engineering (SPE)

Software Performance Engineering (SPE) refers to a set of methods for designing and constructing software systems to meet some performance objectives. SPE methods include performance data collection, quantitative analysis techniques, prediction strategies, and management of uncertainties, data presentation, and tracking, model verification and validation, critical successes factors, and performance design principles [Smith01].

SPE prevents and solves performance problems by analyzing a software system and suggesting corrective measures when necessary. One of the SPE approaches uses a performance model of the system during the analysis. Gathering information about a system's structure and execution is challenging but necessary for constructing a performance model. Various approaches are available to develop the structure and parameters of a performance model and two main approaches are *empirical* and *predictive*.

The empirical approach is based on obtaining performance measurements from a prototype of a software system. Performance measurements include response times, system throughputs and network delays. If these measurements do not meet performance requirements, the system is fine-tuned to obtain better results [Smith 90].

The predictive approach characterizes software execution patterns, constructs a performance model to generate performance estimates, and then uses the predictions to guide modifications. It provides earlier warnings and supports more effective adjustments. The weakness of the predictive approach arises in determining the model structure and parameters without an implementation as guide.

Analysts while creating a model have to review software descriptions, design documents or source code design descriptions. Important end-to-end system behavior, the involved software components, device usage by each component, and the interaction between components are identified and this information converted into a performance model. This large quantity of information becomes unwieldy even for a moderate-size system. Consequently, models become expensive to develop and validate.

Various approaches to SPE applied over the span of software design and development cycle are briefly mentioned in the next section, leading towards software performance modeling.

2.3 Various SPE Approaches

Software performance engineering may use a variety of techniques for performance evaluation such as fix-it-later, performance models, and performance measurement. The use of these techniques is heavily dependent on the architecture of the computer system being analyzed.

2.3.1 Fix-it-Later

Fix-it-later is the simplest approach of performance analysis. Smith describes this approach in [Smith01]. It is very common and usually applied after the design and development of software system. The rationale of this approach is to save time, expense, and maintenance cost. It emphasis the correctness of software and defers performance consideration to the integration-testing phase. If performance problems are detected then corrective measures are applied in the form of additional hardware or software tuning/optimization. The obvious drawback to this approach is that it is extremely difficult to change any major components or redesign the system, once it is implemented and performance difference is almost unsubstantial. This approach is suitable when there are fewer users and systems competing for the resources or when the resource demand is not high. This approach is not suitable for mission critical systems and embedded systems which require critical performance.

2.3.2 Performance Modeling

Software performance predicted by modeling is described by Smith [Smith01]. This technique is used in early stages of a system design. It pinpoints and rectifies key problem areas before system development and implementation. It eliminates the need of redesigning the software system after it is operational. This is especially important in mission critical systems and real time systems where all operations should meet strict performance requirements.

This technique is based on the performance model of a system. A model should describe the key aspects of a system. It should include behavioral structural aspects of workload elements in terms of resource usage. Performance predictions can be developed using this information. These predictions can be analyzed for different system configurations and workload conditions. Performance modeling seems to be a suitable technique of SPE. It certainly achieves the goal of SPE in a feasible manner.

2.3.3 Performance Measurement

Performance measurement is a possible approach to SPE, which falls between the two approaches discussed in previous subsections. In this technique, the performance metrics are used as a criterion to evaluate the performance of the system. Metrics may be obtained by performing a number experiments [Jain91]. Smith divides this technique into sub techniques: recording the events and observing and monitoring the states [Smith01]. The measurement of environmental factor such as workload, the time of day and the duration of measurements need to be controlled. This technique may reveal bottlenecks of the system.

It provides reliable, statistical and accurate measurements of the system. The main drawback of this technique is that it takes time to do the measurements and the system must be operational during the experiments. It is also possible to change the design of the system to improve the accuracy of measurements. This may involve the redesigning of the whole system or a component of the system, which is not feasible from performance viewpoint. It is also a problem to choose a real workload, which will provide the exact measurements for certain operating conditions.

In the next section, we will discuss Use Case Maps (UCM) notations, which are playing an important role in performance modeling and also in system specification.

2.4 Use Case Maps (UCMs) Notation

This section is intended to provide an overview of Use Case Maps (UCMs) [Buhr95] [Buhr98]. UCMs capture the system specifications as a set of scenarios, at a suitable level of abstraction, and define software components and responsibilities. Within an early architecture definition based on Use Case Maps (UCMs), demand budgets are allocated to responsibilities and verified by a semi-automated performance analysis using Layered Queuing Network (LQN) Models.

UCMs are defined as causal scenarios, architectural entities or behavior patterns. They help us to describe and understand emerging behavior of complex and dynamic systems. The notation is intended to be useful for requirement specification, design, testing, maintenance, adaptation and evolution. UCMs notation aims to link behavior and structure in an explicit and visual way.

UCM paths are architectural entities that describe causal relationships between responsibilities, which are bound to underlying organizational structures of abstract components [Buhr98]. These paths represent scenarios that intend to bridge the gap between requirements and detailed design.

With UCMs, scenarios are expressed above the level of messages exchanged between components; hence they are not necessarily bound to a specific organizational structure. This feature promotes the evaluation of architectural alternatives early in the design process. UCMs provide a bird eye view of system functionality, they allow for dynamic behavior and structures to be represented and evaluated, and they improve the level of reusability of scenarios.

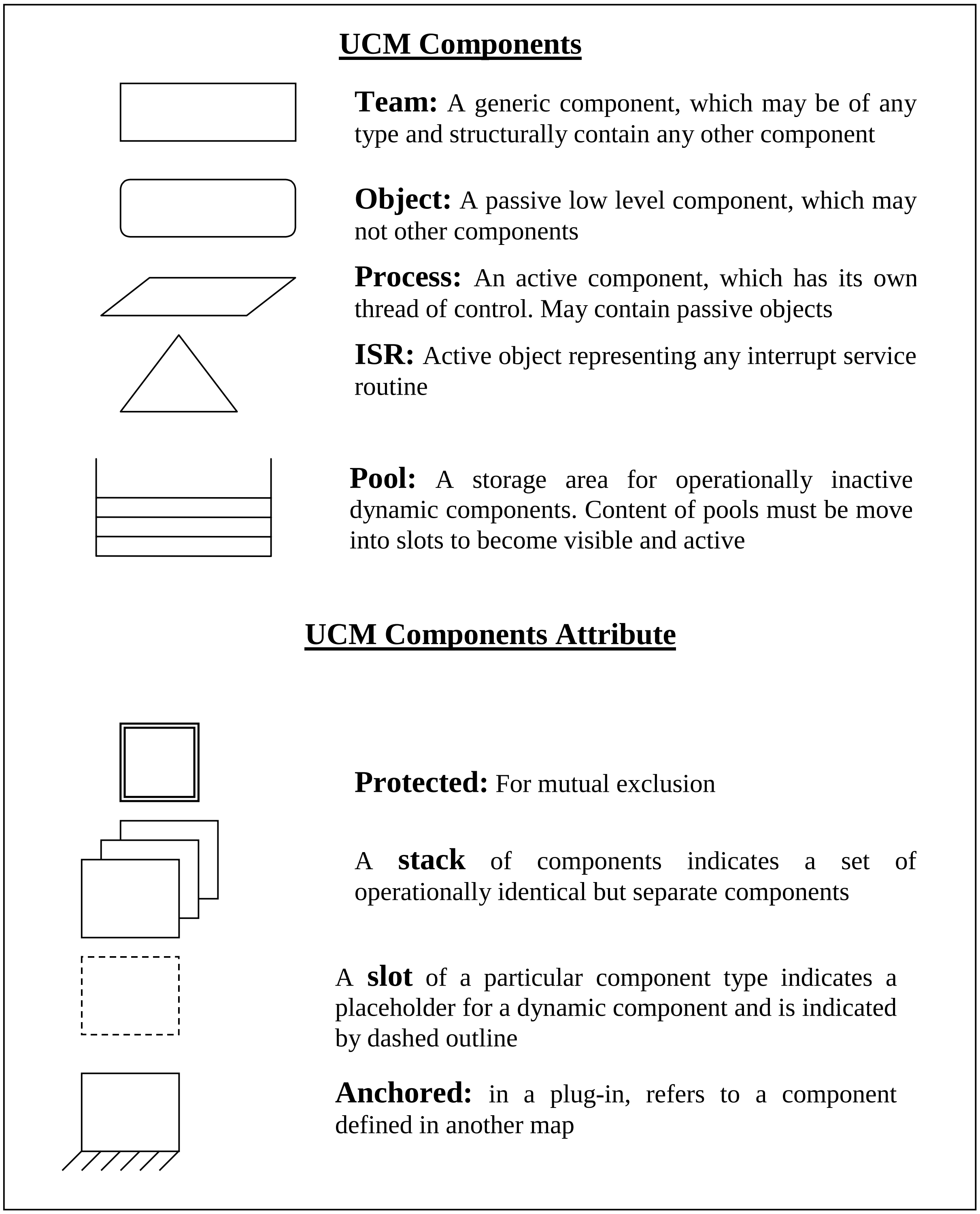
The UCM methodology is based on the concept of describing end-to-end scenarios through a system as a causal flow of events, and actions through an underlying component substrate. Details of inter-component communications are considered lower level detail, which can be determined at a later stage. The UCM methodology allows end-to-end causal scenarios to be described at any level of abstraction allowing the behavior of complex systems to be described with ease. Support for design scalability is built into the UCM methodology into two forms, support for the concept of stubbing which allows path segments to be expanded as separate diagrams and support for the concept of layering where a UCM of a certain layer describes certain behavior and considers other operations to be lower level detail, to be described by a UCM of a lower layer. A complex system described properly with a set of UCMs allows for much greater understanding by all designers and implementers resulting in increased efficiency for a software system.

2.4.1 Requirements for the Construction of UCMs

UCMs can be extracted from informal requirements if they are available. Responsibilities should be stated or can be extracted from these requirements. UCMs can be created for individual system functionalities or even for individual scenarios for illustration purpose. However, the strength of this notation mainly resides in the integration of scenarios. It is important to clearly define the interface between the environment and the system under description. This interface will lead to the start points and end points of the UCM paths, and it also corresponds to the messages exchanged between the system and its environment. Designers are often the ones who create UCMs, some design information may be relevant. In theory, UCM can be composed of paths where responsibilities are not allocated to any component. However, designers are likely to include architectural elements such as internal components. In this case, the description of these components, their nature, and some relationships, e.g., components that include sub-components, are required. Communication links between components are usually not required, but they can be added.

2.4.2 UCM Components

Components can be of different natures, allowing for a better and more appropriate description of some entities in a system. Figure 2.1 shows the component used in UCM and also shows the detail of components. There are different types of components and they have different types of attributes. Buhr suggested several types of components and attributes for complex systems. These systems can be real-time, object-oriented, dynamic or agent based etc [Buhr95, Buhr98 and UCMOrg].



UCM Components

Team: A generic component, which may be of any type and structurally contain any other component

Object: A passive low level component, which may not other components

Process: An active component, which has its own thread of control. May contain passive objects

ISR: Active object representing any interrupt service routine

Pool: A storage area for operationally inactive dynamic components. Content of pools must be move into slots to become visible and active

UCM Components Attribute

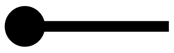
Protected: For mutual exclusion

A **stack** of components indicates a set of operationally identical but separate components

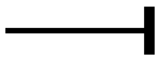
A **slot** of a particular component type indicates a placeholder for a dynamic component and is indicated by dashed outline

Anchored: in a plug-in, refers to a component defined in another map

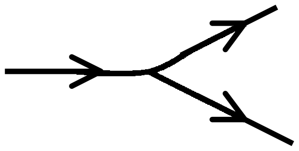
Figure 2.1: UCM components and components attributes



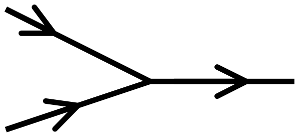
Start Point: Starting delimiter for a UCM path, which signifies the starting of a scenario when appropriate stimulus is received.



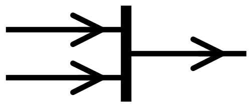
End Point: Terminating delimiter for a UCM path, which signifies the end of a scenario



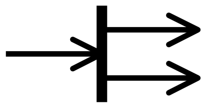
OR-Fork: Indicates routes that share common causal segments, Alternative may be identified by labels or by conditions



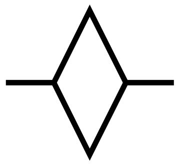
OR-Join: Indicates that many concurrent paths are synchronized to one path



AND-Fork: Indicates that a single path split into many concurrent forks.

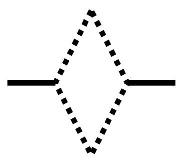


AND-Join: Indicates that numerous concurrent paths synchronize into a single path.

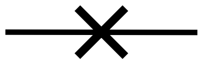


Static Stub: An element of decomposition in UCMs where a sub map may be defined. This is a static stub with a single sub map.

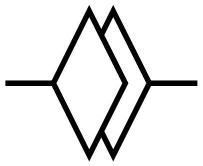
Figure 2.2 (a): UCM Path Elements



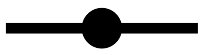
Dynamic Stub: An element of decomposition in UCMs where a sub map may be defined. This is a dynamic stub with many sub maps. The selection can be determined at run-time according to a selection policy usually describe as pre-conditions.



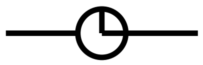
Responsibility: Specifies an action to be performed by the end system that point of the path. May be bound to of software component.



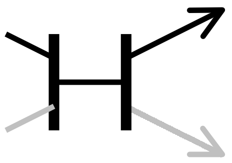
Shared Stub: Specifies that the sub map will be shared by different paths.



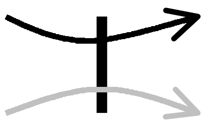
Waiting Place: Specifies a synchronization point for a scenario where a scenario pauses until a triggering event is received.



Time Waiting Place: A waiting place, which may have a timeout period, defined at which point an exception action is taken.

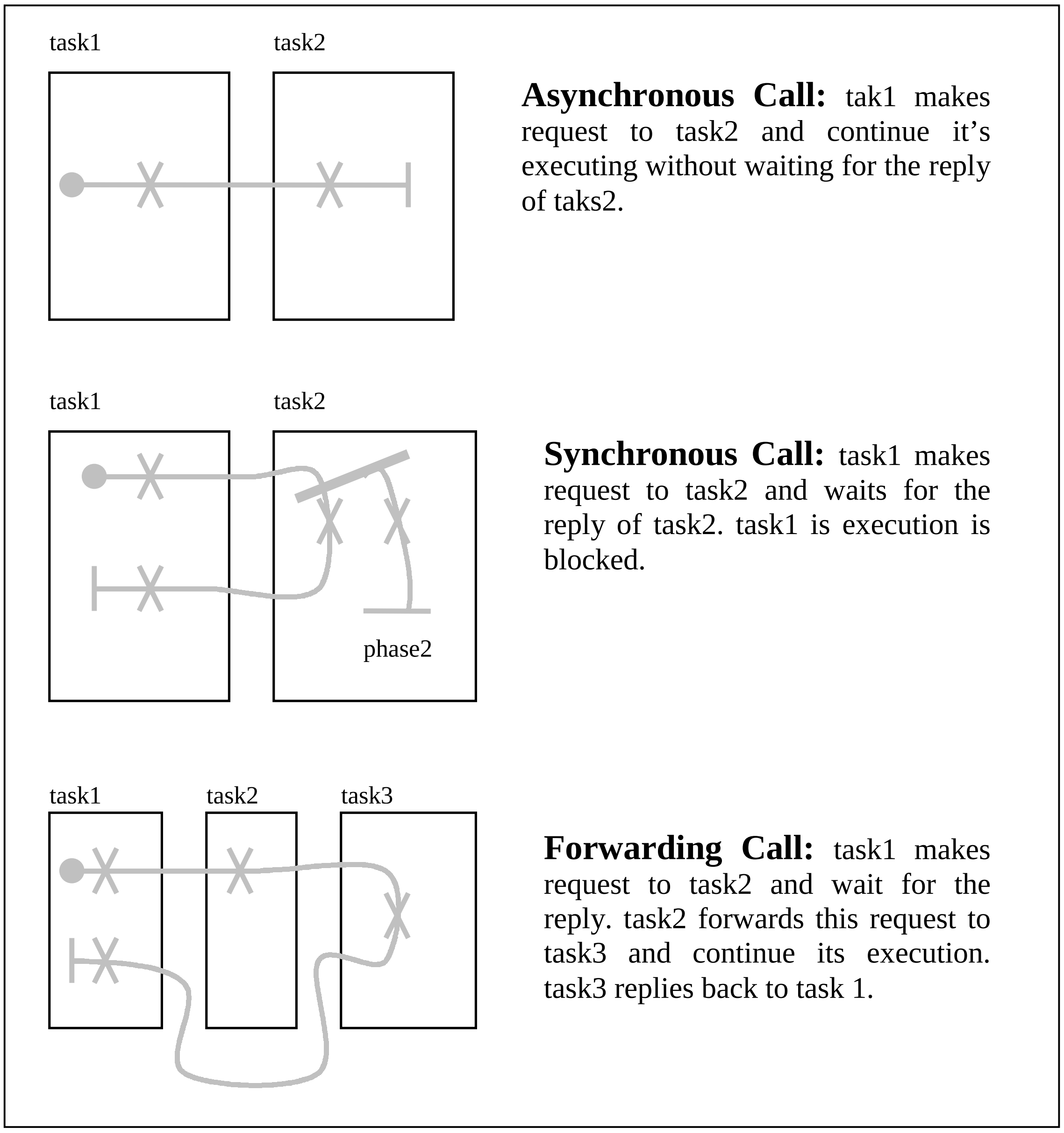


Rendezvous: N:1 and 1:N



Synchronize: N:N

Figure 2.2 (b): UCM Path Elements



Asynchronous Call: tak1 makes request to task2 and continue it's executing without waiting for the reply of taks2.

Synchronous Call: task1 makes request to task2 and waits for the reply of task2. task1 is execution is blocked.

Forwarding Call: task1 makes request to task2 and wait for the reply. task2 forwards this request to task3 and continue its execution. task3 replies back to task 1.

Figure 2.3: Types of Calls

2.4.3 UCM Path Elements

The basic path notation [Woodside01 and UCMOrg] addresses simple operators for causally linking responsibilities in sequences, as alternatives, and in parallel. More advanced operators can be used for structuring UCMs hierarchically and for representing exceptional scenarios and dynamic behavior.

Figure 2.2(a) and 2.2(b) show the UCM path elements with their descriptions. When UCM scenarios become very complex then it is good to divide a big and complex map into small sub maps. A top-level UCM is referred as a root map, which can include containers called stubs, which are replaced by sub-maps called plug-in maps. Figure 2.2 (a) and 2.2(b) show two kinds of stubs with their descriptions. Figure 2.3 shows the type of calls or interactions, which can be implemented in UCM scenarios. The next two sections will describe examples for simple and complex UCM scenarios.

2.4.4 Example: Simple UCM Scenario

This section will discuss how a UCM scenario looks like and how to incorporate UCM components and path elements into a UCM scenario. Figure 2.4 shows a very basic, simple and abstract UCM scenario. There are two paths in this scenario. These paths are distinguished by their colors. It shows how the paths start and how they end. The respected responsibilities associated with each path are labeled as 'x'. The components are shown as rectangular boxes, which can be different task of a software system. The 'Start Point' is represented as filled circles, which show pre-conditions or triggering causes of a path.

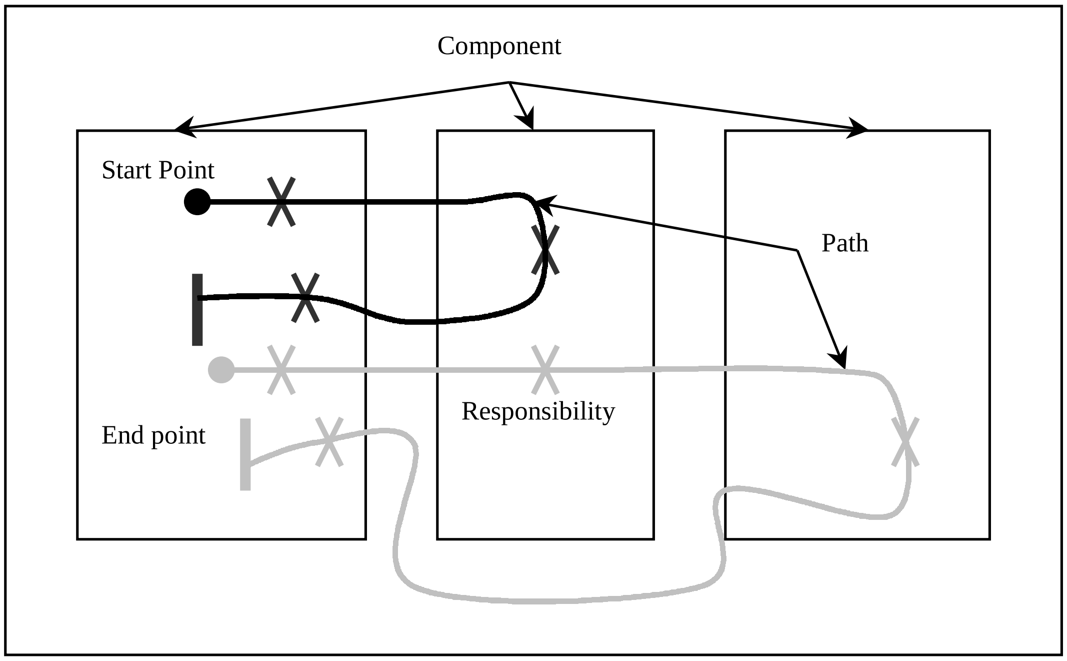


Figure 2.4: Simple UCM Scenario

An ‘End Point’ is represented as a small vertical line, which can show post-conditions or the resulting effect of a path. The lines connecting ‘Start Point’, ‘End Point’ and responsibilities are the paths.

A responsibility is bound to a component when the cross is inside the component. In this case, the component is responsible to perform the action, task, or function represented by responsibility. Start points may have preconditions attached, while responsibilities and end points can have post conditions. This route, which traverses paths and associated responsibilities from a start point to an end point, is known as scenario.

The black color path shows one component/task is interacting with another component/task synchronously. The first component or task waits for the reply of other component/task. Gray color path shows that one component/task is interacting with other component/task. The other component/task forwards the request to another component /task which reply back to first component/task. During this period first component/task was blocked because it was waiting for the reply.

2.4.5 Example: Complex UCM Scenario

Figure 2.5 shows a complex UCM scenario. The big complex map is called the root map. The small rectangular boxes at the top show sub maps or plug-in maps. This is paradigm of a backup process. When a user requests to perform a backup process, first it is validated whether the user is an authentic user or not. If it is not then the request is rejected other wise the request is accepted for backup and forwarded it to backup process. The first rectangular box of root map is “User Process”, backup user uses this task to make a request to perform backup on a device. “User Process” sends this request to “Authentication Process”. “Authentication Process” has solid line diamond. This diamond is a static stub, which represents sub map or a plug-in map. Since this is a static stub that’s why only one sub map is associated with it.

The plug-in map of “Authentication Process” is an OR path. It accepts user request at “in” at static stub and checks whether the use is authentic user or not. The corresponding responsibilities are labeled for “Authorized User” and “Unauthorized User”. The resulting paths from plug-in map are labeled as “out 1” and “out2” outside the static stub in root map.

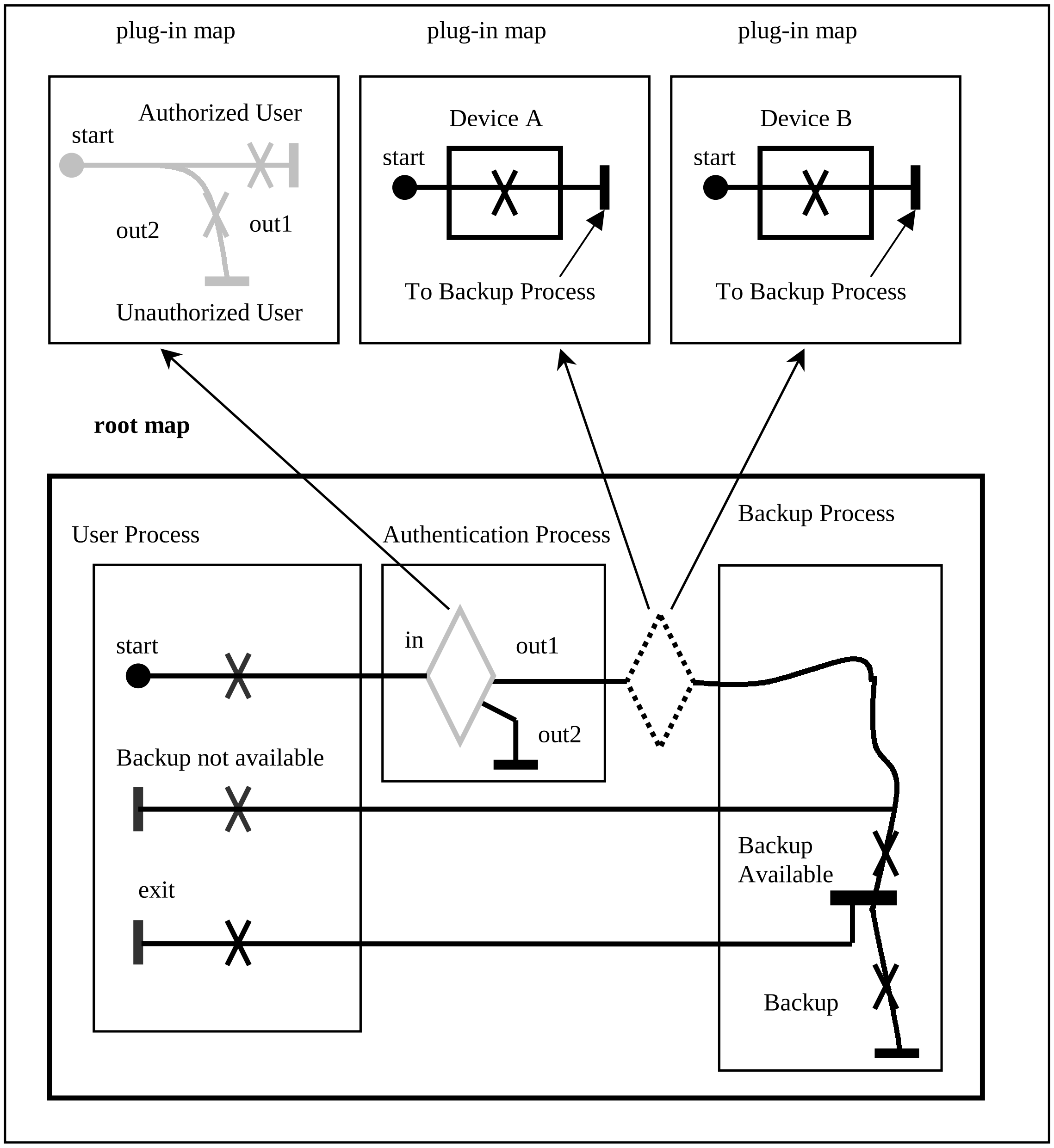


Figure 2.5: Complex UCM Scenario

Authenticated user request if forwarded to dotted line stub, which is a dynamic stub. Since it is a dynamic stub more than one sub maps can be attached to it. Here, in this example two sub maps are attached to this dynamic stub. These sub maps are two different maps for different devices. The corresponding responsibilities of each map are labeled for each device. User selects one of the devices according to his choice for the backup.

The last rectangular box in root map is a "Backup Process". This component decides whether backup is available or not. If it is not then the request is return to "User Process" that backup is not available. The corresponding responsibility is labeled, as "Backup not available" in "User Process" component. If backup is available then "Backup available" is forked into two processes. These are concurrent processes. "Backup Process" informs "User Process" about successful back up process. The corresponding responsibility is labeled as "exit". The other process is the backup process, which starts a backup. The corresponding responsibility is labeled as "Backup".

2.5 Layered Queuing Network (LQN) Modeling

A Layered Queuing Network (LQN) Model [Woodside95] is proposed for the evaluation because it is closely matched to a wide range of software design styles, and it is also a generalization of well known, and robust queuing network models. It is a new adaptation of extended queuing models for software systems proposed by Woodside [Rolia95, Woodside89, and Woodside95b]. It can be solved by analytic or simulation techniques; other analytic or simulation models could also be used. It is capable of modeling most of the features the are important from performance point of view such as multi-threaded processors, devices, locks, communication and so on [Franks99].

The LQN model is a kind of queuing model based on the UCM and its parameters, and on the expected execution environment. In defining the environment, additional components are added to the system, and these may be held in a component sub-model library. Layered Queuing Network Solver (LQNS) offers a variety of output measures, which need to be tailored to the analysis, to make it easy to use [Woodside95c].

LQN model represents software resources in a natural way so that they are the parts of the framework and thus approximations do not have to be developed for every system. The model is closely linked to software descriptions and provides a transparent representation of the software architecture, which makes models easy to develop and understand. The model is well suited for parallel processes running on a multiprocessor or on a network, such as client-server system.

LQN is used to build model in order to analyze the performance of software system. The models consist of tasks with associated entries and lists of activities. The tasks are organized in conceptual layers interacting with each other through synchronous, asynchronous, or forwarding calls. The entries represent the call reception points for tasks and are linked to activities that make service requests to lower level tasks and/or to resources such as processors, disks, and/or other underlying services.

A LQN model is a generalization of queuing networks that represents these in terms of requests for service between tasks and the queuing of messages at an actor. An LQN can represent any queuing model as special case. In LQN, tasks can accept service request messages at an LQN entry. An entry describes and models a service provided by an actor, and also the associated resource demand. If a task is a kind of object, an entry is a kind of method. Requests for service are characterize as entry-to-entry interactions sent in a synchronous (i.e. send-wait-reply), or asynchronous (i.e. send-and-continue) fashion. The interaction types affect performance because they affect task blocking, queuing delays at devices.

Hardware and software objects are represented as tasks in LQN model, which may execute concurrently. There are three categories of tasks in LQN model. The first categories of tasks are referred as pure server tasks. These tasks are used to model hardware devices such as processors or disks. They receive requests form other tasks but cannot make their own requests. The second categories of tasks are referred as client tasks. These tasks only send requests and are used to model input sources such as users and so on. The third categories of tasks are referred as active server tasks. These tasks can receive requests, and make their own requests.

The interaction or call types are also very important in LQN model. There are three types of calls in LQN model. These are synchronous, asynchronous and forwarding calls. The description of each call will be discussed next.

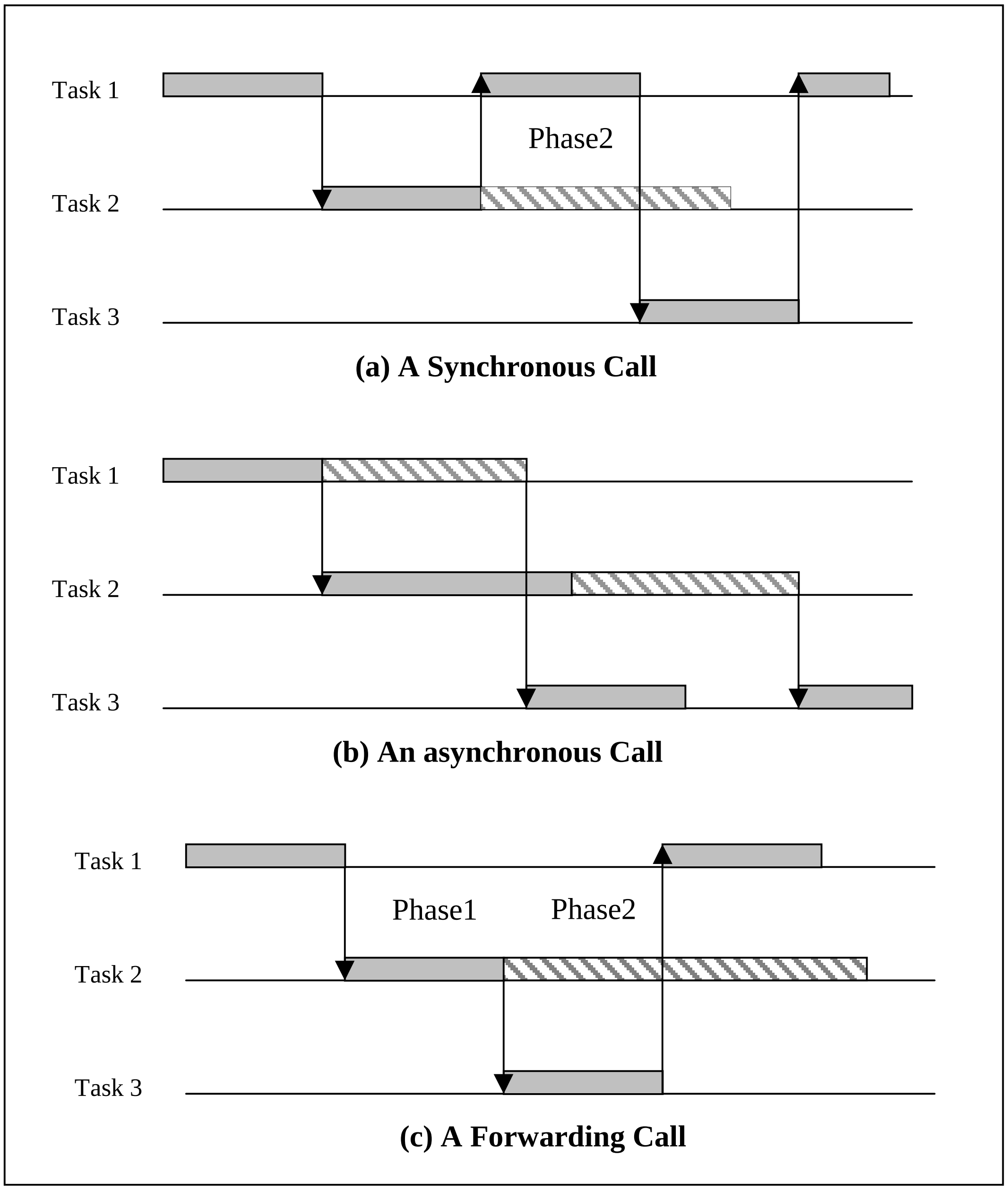


Figure 2.6: Call Patterns in LQN model

In a synchronous call, a task that requests service to other task waits for the reply. When the requesting task gets the reply from the task then it continues its normal execution. It means that during a synchronous call a task is blocked until and unless it gets the reply of its request. Synchronous call pattern is shown in Figure 2.6 (a).

In an asynchronous call, a task that requests service from other tasks does not wait for the reply of the request. The requesting task resumes its execution after making its request to other task. Asynchronous call pattern is shown in Figure 2.6 (b).

Forwarding call can be considered as special case of an asynchronous call. When a task sends a request to the other task that responding task sends this request to some other task and does not reply back to the blocked task that initiated the request. The last task in the chain sends reply back to the task that initiated the request. Forwarding call pattern is shown in the Figure 2.7 (c).

In a synchronous protocol type there may be two phases of execution. It can be seen in Figure 2.6(a) and Figure 2.6(c). During first phase the task accepts the request from other task and reply back or forward the request. The task, that reply back or forward the request, continue its execution in another phase, this is called phase2. The second phase identifies resource contention between the initiating task and the continuing responding task.

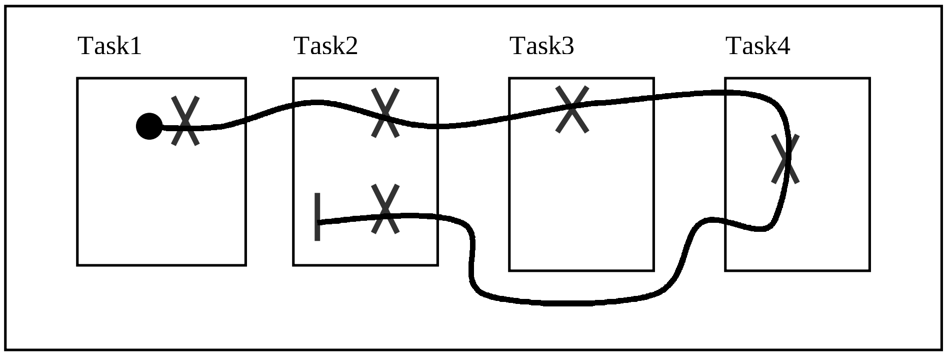


Figure 2.7: Simple UCM Scenario

2.5.1 Example: Simple LQN Model

Figure 2.7 shows a simple UCM scenario and Figure 2.8 shows LQN model of this UCM. In LQN model, tasks are shown as parallelograms and requests from one task to another task are made from and to service entries, which are ports or addresses of particular service offered by a task. An entry executes activities with precedence relationships, and activities have resources demands and can make requests to other tasks. For each activity, a resource consumption value such as CPU consumption, storage operations, and any other operations of the process to carry out the execution steps, must be made available from a repository of resource functions. Resource functions will not be discussed in this thesis.

The interaction or call types are distinguished by different type of arrows. A filled arrow represents the synchronous call, asynchronous call by normal arrow while forwarding call by dotted line arrow. This is the best way to represent different types of call in a LQN model.

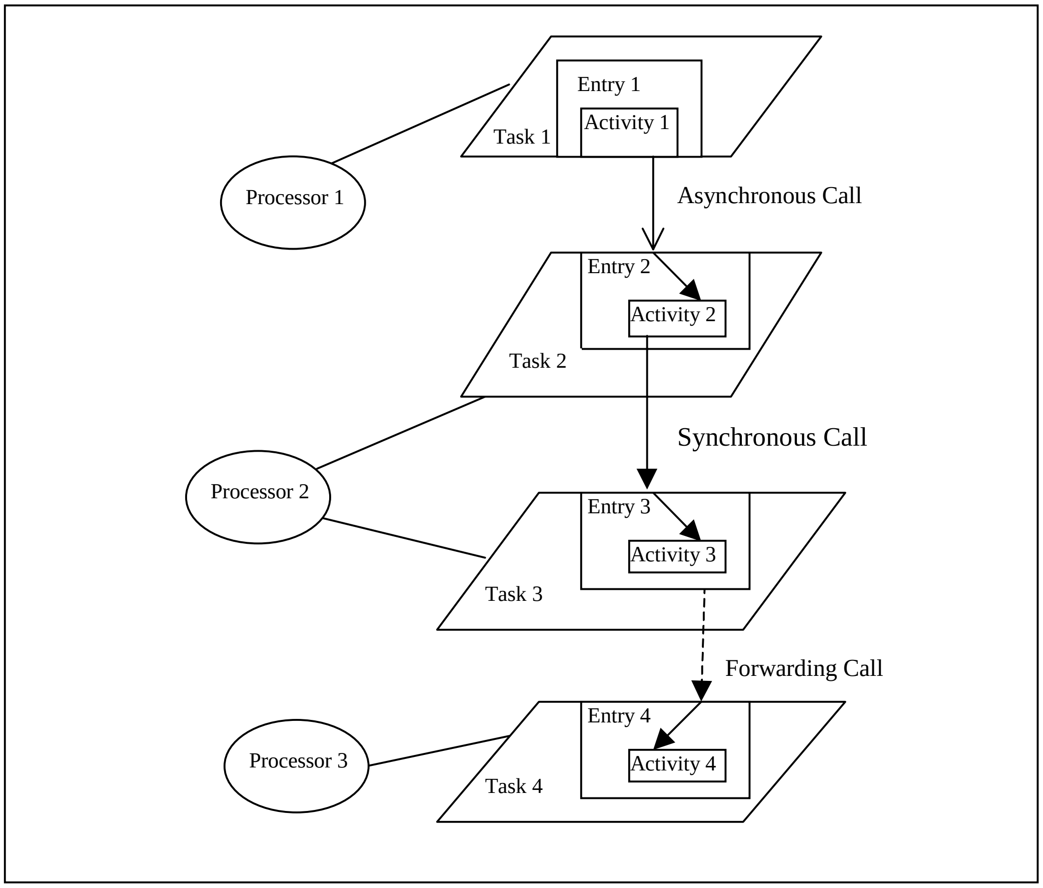


Figure 2.8: A Layered Queuing Network (LQN) Model

2.6 Performance Budgeting

Performance Budgeting considers performance issues and information from the earliest stages of development. It includes techniques for generating the information, including performance estimates, and techniques for guiding and coordinating the effort for development.

Performance Budgeting is another approach to achieve the performance targets of any software system. The idea behind the performance budgeting is to derive the estimates for the resource demands for a software system before the design. The estimated figure may be based on past experiments and results. The best thing before the design of software system is to plan the budgets for the resource demands of all the parts and operations then perform a validation check for the required performance. The real time system requirements such as timeliness, predictibility, dependendibility, and deadline scheduling are discussed in [Halang91]. The idea of validation of resource budgets is discussed in [Woodside99].

Performance budgeting allocates resources based on achieving specific measurable resource demands (i.e. CPU execution demand, disk operation demand and network demand etc) rather than the required resources selection and other costs to carry out an operation or activity. Resource demand target can be a starting point. Thus performance budgeting first asks not how many resources do we **expect** to put through for a software system but rather, how many resources do we **need** for software system. The numbers of resources required are resource demands and they are the basis of performance budgets. Resource demands are defined through a strategic planning process that considers the critical issues like predictions and past experiences etc.

Associated tools, which are available for performance budgeting, will be discussed in next section.

2.7 Automated tools support for Performance Budgeting

Automation plays an important role for any complex system. Many tools can be developed to provide automation. Tool support is desired for two main reasons, the increased ease of manipulation of designs and the desire to use design specifications as input to other tools. Proper tool support would provide a platform for designers to study the issues of large systems. Some tools are already available to automate completion process and new tools can also be designed and implemented. Research and development work is required in this direction to enhance this research work in future.

2.7.1 UCM Navigator (UCMNav)

The specification of large, complex distributed communication software systems often overwhelms designers with low level details but often does not provide a high level view at which the behavior of a system may be understood. These complex systems are complex to understand and difficult to modify and extend. This results in increased development costs and greater unreliability of software as systems, which cannot be understood, cannot be modified without introducing new bugs. The high level visualization of system is very necessary. The UCM methodology is developed to fill a void in the specification of software systems in that it aims to provide a high level overview of the behavior of a system. UCM Navigator (UCMNav) is a graphical editor, which is designed for this purpose [Miga98]. It supports the UCM methodology and provides support for multilevel designs, for description of the performance for its use as a front end for performance simulations and for the generation of a linear textual representation of entered designed allowing the UCM methodology to be used as frond end for other software engineering tools. UCM Navigator can be used to provide completion at UCM level. We will discuss in a detailed section, how this tool can be used to provide completion at UCM level.

2.7.2 Layered Queueing Network Solver (LQNS)

The LQNS is a tool that solves LQN models and returns performance parameters for the system [Woodside95c]. It can also be used to detect contention for certain services or devices, and conditions such as deadlocks, races, and/or bottlenecks. This tool can be used to provide completion at LQN level. The LQN model is a kind of queuing model based on the UCM and its parameters, and on the expected execution environment. In defining the environment, additional components are added to the system, and these may be held in a component sub-model library, which facilitate the “completion”. The LQNS offers a variety of output measures, which need to be tailored to the analysis, to make it easy to use.

2.7.3 Java Layered Queuing Network Definition tool (JLQNDef)

JLQNDef is a tool that provides an LQN editor with LQNS, as well as providing a graphical output of the layered architecture. This tool can be very helpful for the "completion", which will be described in next chapter. It can further be enhanced. New features can be introduced to provide completion at LQN level graphically. Features like one component/task replacement with multiple component/tasks by mouse drag and drop. Another feature, which will introduce disk, file server, and tuple space server in the system graphically. These components can be selected from the window menu and would be introduced at mouse click.

2.7.4 LQN Components Filter

LQN have been used to model and evaluate performance of many different systems such as web servers and network file systems. Many of these components may occur as parts of larger systems. If the same system occurs many times within the larger systems, it must be modeled repeatedly. As a result, building a LQN model of a large system that has number of identical or similar subsystems becomes tedious, especially if the subsystems are also large or have complex interactions with their environments. Evaluating alternative LQN models where particular subsystems are replaced by other subsystems requires the larger model to be reconstructed for each alternative. This is also tedious. To address these concerns, the concept of LQN components is introduced in [McMullan00]. This tool automates the process of component replacement. This is not a graphical tool. It replaces one task with a set of tasks. It is certainly helpful for the "completion" when it is required to replace one task with a set of tasks. It demonstrated two examples for task replacement in [McMullan00], which could be a kind of "completion". In first example, LinuxNFS component instance was replaced with two tasks NFS1 and NFS2. In second example, one instance of LinuxNFS component with its service interface was replicated to accommodate four different client tasks.

2.7.5 UCM to LQN Converter (UCM2LQN)

An automated generative tool UCM to LQN (UCM2LQN) Converter for creating performance models from UCMs is described in [Petriu01a] [Petriu01b]. This converter automates the transition from the UCM design model to a corresponding LQN model. The transition is based on certain interaction relationships between the UCM components, which are implicit in the UCM representation. This is really helpful for the "completion" because it automates the process of UCM to LQN conversion. UCM to

LQN conversion is done by hand for this thesis. The question is how advanced is this tool? Will it convert any new style set of UCMs or only works for any specific set of UCMs. At this time it does not have capability to handle dynamic stubs and complex UCM. It does not handle the looping problem of the UCM design.

2.7.6 Latency and Message Overhead Filter

The LQNS had some limitations. It lacked parameter values to describe the communication delay between pairs of processors. Ideally, there is no delay between a sender and receiver during message passing. In fact, there is always a delay between sender and receiver as messages passing due to priority for execution, or delay associated with acknowledgement.

This tool also facilitates the process of completion in the form of latency and message overheads [Lee01]. It provides “completion” at LQN level. It does this by incorporating the user think time as latency between the processors and message overheads in execution time between the entries. If processors are located on the same machine then latency will be zero. If they are located remotely then there will be some delay. At this time this tool handles synchronous calls only. Asynchronous and forwarding calls can be implemented in future to improve this tool.

2.8 Tuple Space

This section will describe tuple space communication that is used in different examples showing “completion” based on tuple space network components.

Network operating systems provide three basic mechanisms that are used to support the services provided by the operating system and applications. These mechanisms are Message Passing, Remote Procedure Call (RPC) and Distributed Shared Memory (DSM) [Dasgupta98]. These mechanisms support a feature called *Inter Process Communication* (IPC). While Message Passing and RPC are the mainstays of distributed programming, and are available on all network operating systems, DSM is not at all ubiquitous. DSM is a feature by which two or more processes on two or more machines can map a single shared memory segment to their address spaces. This shared segment behaves like real shared memory, that is, any change made by any process to any byte in the shared segment is instantaneously seen by all the processes that map the segment.

Tuple Space is one of the shared memory paradigms [Roberts92, Foster95, Walkner95, Gelerntner97, Dasgupta98 and Matloff99]. Tuple Space is also a coordination mechanism like any other coordination mechanism such as direct, meeting-oriented [Peine97] or blackboard. Tuple Space uses a Linda like coordination mechanism [Roberts92, Walkner95 and Dasgupta98].

In Linda-like coordination, accesses to a local blackboard are based on associative mechanisms. Linda generalizes the concept of the communication via common variables by offering an associative *shared memory* or a *global mailbox* called "tuple space", through which concurrently acting processes can cooperate and communicate. The data objects inside tuple space are referred to as "tuples". A tuple is an ordered *sequence of data* or *messages*.

Linda-based tuples are unstructured, therefore selection from multiple matches is arbitrary and implementation-dependent. To share data, generate a tuple. To request data, request a tuple. It provides many features such as destination uncoupling, space uncoupling and time uncoupling. In destination uncoupling, the receiver does not know the sender and the sender does not know the receiver. Space uncoupling provides associative versus physical addressing and it is machine and platform independent. In time uncoupling, tuple life span is independent of both creators and readers. It supports time-disjoint processes. The original Linda model defines four basic operators:

- **out** inserts a tuple, composed of an arbitrary mix of actual and formal fields, into a tuple space. This tuple becomes visible to all processes with access to that tuple space.
- **in** extracts a tuple from a tuple space, with its argument acting as the template, or *anti-tuple*, against which to match. Actual match tuple fields if they are of equal type and value; formals match if their field types are equal. If all corresponding fields of a tuple match the template the tuple is withdrawn and any actual it contains are assigned to formals in the template. Tuples are matched non-deterministically and **in** operations block until a suitable tuple can be found.
- **rd** is syntactically and semantically equivalent to **in** except that a matched tuple is copied, not withdrawn, from the tuple space and hence remains visible to other processes.

- **eval** is similar to **out**, except it creates *active* rather than *passive* tuples. The tuple is active because separate processes are spawned to evaluate each of its fields. The tuple subsequently evolves into a passive tuple resident in the tuple space.

In addition to the basic model described above more than a decade of research by the parallel programming community has led to a number of refinements and extensions to the paradigm. For example, many implementations support two new operators, **inp** and **rdp** which are non-blocking versions of **in** and **rd**.

Figure 2.9 shows a tuple space and different operations associated to it. Tuples are like vectors, except that their components can be of various mixed types, i.e. integers, floating-point numbers, character strings, and so on. For example, a tuple could consist of ("abc", 1.5, 12). The tuple (‘array x’, 5, [1 4 2 7 9]) is a tuple consisting of a string, an integer, and an array of integers. In Linda, processes only interact through the tuple space, and routines are provided for placing tuples into, and extracting tuples from, tuple space. The **out** operation places the tuple into tuple space. Linda is well suited for handling dynamic load balancing by treating tuple space as a “bag of tasks.” Tasks to be performed can be placed into tuple space with ***eval***, and on the “completion” of a task the results are put back into tuple space.

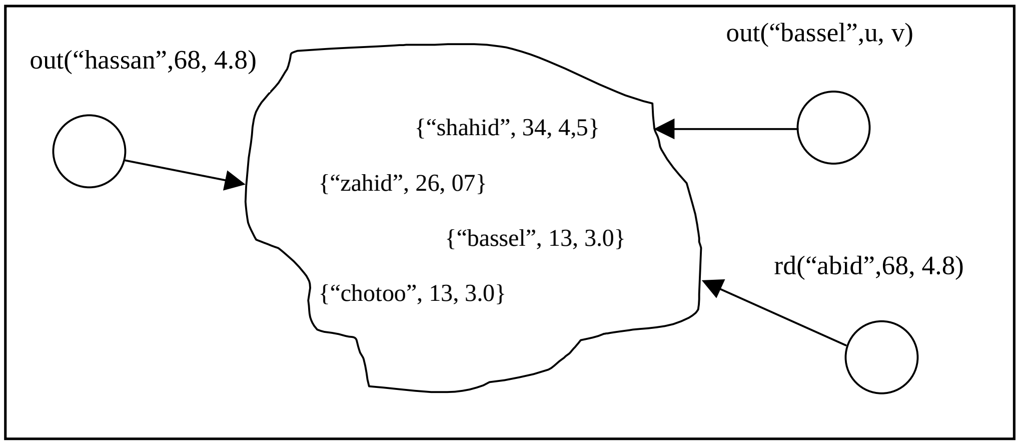


Figure 2.9: Tuple Space

There are many advantages of tuple space such as flexibility and scalability. Flexibility does not restrict format of tuples nor types of data contained. Scalability provides facility of anonymity of tuple operations.

Figure 2.10 shows a generalized tuple space UCM scenario in which ProcessA places a tuple in tuple space using **out** operation and ProcessB remove tuple from tuple space with **in** operation. The responsibilities show respective operation of each process and tuple space.

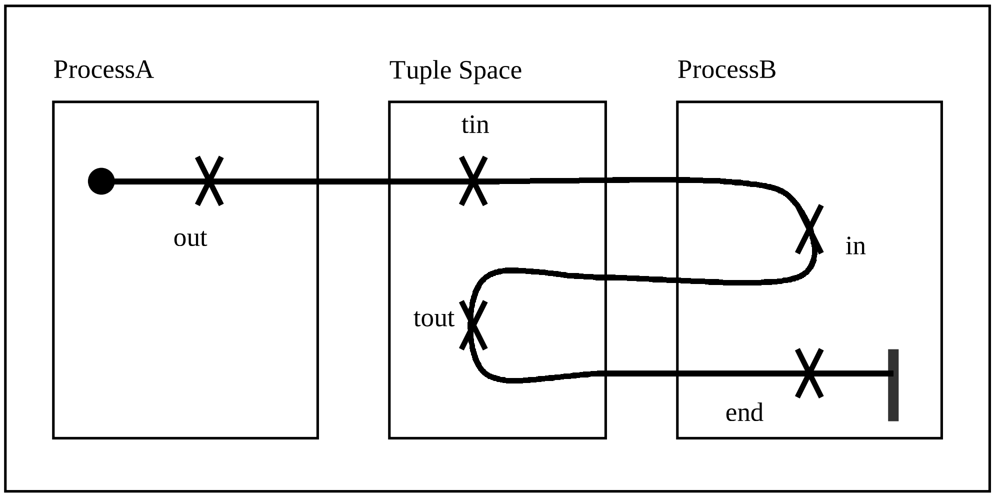


Figure 2.10: Tuple Space UCM Scenario

2.9 Summary

This chapter provided background material for the following:

1. Various approaches to performance engineering described with pros and cons,
2. Use Case Map (UCM) specification, a starting point of the research,
3. Layered Queuing Network (LQN) Modeling with an example showing the relationship between UCM(s) and LQN(s),
4. The need for performance budgeting,
5. The available automated tool support for the above, and
6. Tuple Space, which is used as one the communication mechanisms for the "completion"

3.0 Performance Budgeting in a Software Design Process

3.1 Introduction

This chapter will describe idea of performance budgeting, budget analysis road map, the steps involved in this road map, the concepts and terminology associated with the road map and an example, which will walk through this road map step by step.

3.2 Performance Budgeting

Budgets are a general approach to break a large problem down into manageable pieces; common examples are financial budgets and time budgets for projects. This section will describe a budgeting approach to deal with time in software design. It does two things: It establishes processing demands for separate parts that are performed by separate system components or activities, and which are the responsibilities of separate developer groups. It uses a model to estimate the resulting time delays, which have two parts, first part due to the execution of the software being developed which is measurable by the developer as it emerges and second part due to contention delays, environment overheads and system latencies, which are outside the designers' control.

What this breakdown achieves is, to give developer targets, which he or she can understand and attempt to achieve, without having to understand the entire system. The understanding of the entire system is focused where it should be, at the managerial level, perhaps with some specialist participation in working with the model. It also supports the re-use of information about the environment, in models of successive or evolving projects.

In evaluating the budget, time delays can be estimated from the budgeted demands using the performance model. The analysis can go in a forward direction, from execution demands to delays, or in a reverse direction from permitted delays to permitted values for execution demands. In the forward direction, execution demands in terms of CPU seconds, network or storage operations are allocated to each activity e.g. each responsibility in UCM. The model results are computed and examined to see if they are consistent with requirements. If adjustments are made to the budgets, they can be similarly evaluated. The forward direction is easy to understand, but it may require iteration to find suitable budget values. In the reverse direction, an automated search is carried out to find the budgets, which satisfy the delay requirements and any other necessary criteria and constraints. This thesis focuses on the forward direction, which can then be a key component in the larger operations in the reverse direction.

Performance models are the natural way to provide predictions for a range of environmental conditions or design alternatives and these predictions can be used to detect problems and suggest corrections. For many software design environments there are significant uncertainties in the execution demands/processing time and communication messages delays. The UCM model does not discuss the uncertainty issues, although it can specify variances for resource needs. These uncertainties will also be incorporated in the budgets and will be gradually resolved in the process of development.

Performance requirements and predictions may be stated as mean values, mean values with a given variance, or as percentile values (such as 95% of response should be completed within a given delay). Hard deadlines are a special case of percentiles (100% of responses in the given delay) but are rare in telecommunications. In evaluating models, simulation techniques can give average or percentile values while analytical models usually only give mean values. Preliminary analysis may be based on analytic results, and more careful analysis on simulations.

The difficulties in applying this budgeting idea fall into two groups: making a suitable performance model from the UCM and creating, evaluating and modifying the budget values. In the second category there are questions such as, how do we assign a budget to a developer? And what will be the largest possible budget? Intuition is a poor guide because there are so many factors, which are applicable to certain situations and data configurations only. The budgeting for performance can be question of trial and error where the intuition and experience of the design team will play a key role. Putative configurations can be proposed, the corresponding performance models can be derived and evaluated, and the prediction results can be compared against requirements and previous results. Budget can be varied. The question here is, how we go systematically?

If we have a time shortfall, how do we respond? It can be by a balanced budget change, a software design change, or a hardware change.

In previous research, there has been frequent mention of budgeting of time in software, particularly in designing for hard-real-time deadlines. In these systems, schedule analysis plays the role of the model; to verify that timing constraints can be met deterministically (see, e.g. [Halang91]). For soft real-time, the discussion has been rather general and solid techniques are scarce.

This work is different in that it starts from designer intentions expressed in the UCMs, and that it addresses budgeting issues in soft-real-time systems.

3.3 Budget Analysis Road Map

Figure 3.1 illustrates the process for analyzing budgets. The boxes represent design artifacts or libraries used in the analysis, while the arrows show operations carried out either by the analysis, or by some automated transformation. There are seven steps.

3.3.1 Step1: Designer UCM

The designer UCM is a document, which describes the system specification as a set of scenarios, at a suitable level of abstraction, and defines software components and responsibilities. It shows the idea of software designer at some early stage of development.

UCMs are used to capture user (functional) requirements when very little design detail is available, without reference to messages or component states. With UCMs, scenarios are expressed above the level of messages exchanged between components. A software designer creates UCMs and he may include some design information but not all of them. UCM can be composed of paths where responsibilities are not allocated to any component. However, software designers are likely to hide architectural elements such as internal components. In this case, the description of these components, their nature, and some relationships, e.g., components that include sub-components, are required. Communication links between components are required. If they are not included in the map they have to be added. They can be incorporated in the form of a component, static (single scenario) or dynamic (multiple scenarios) stubs.

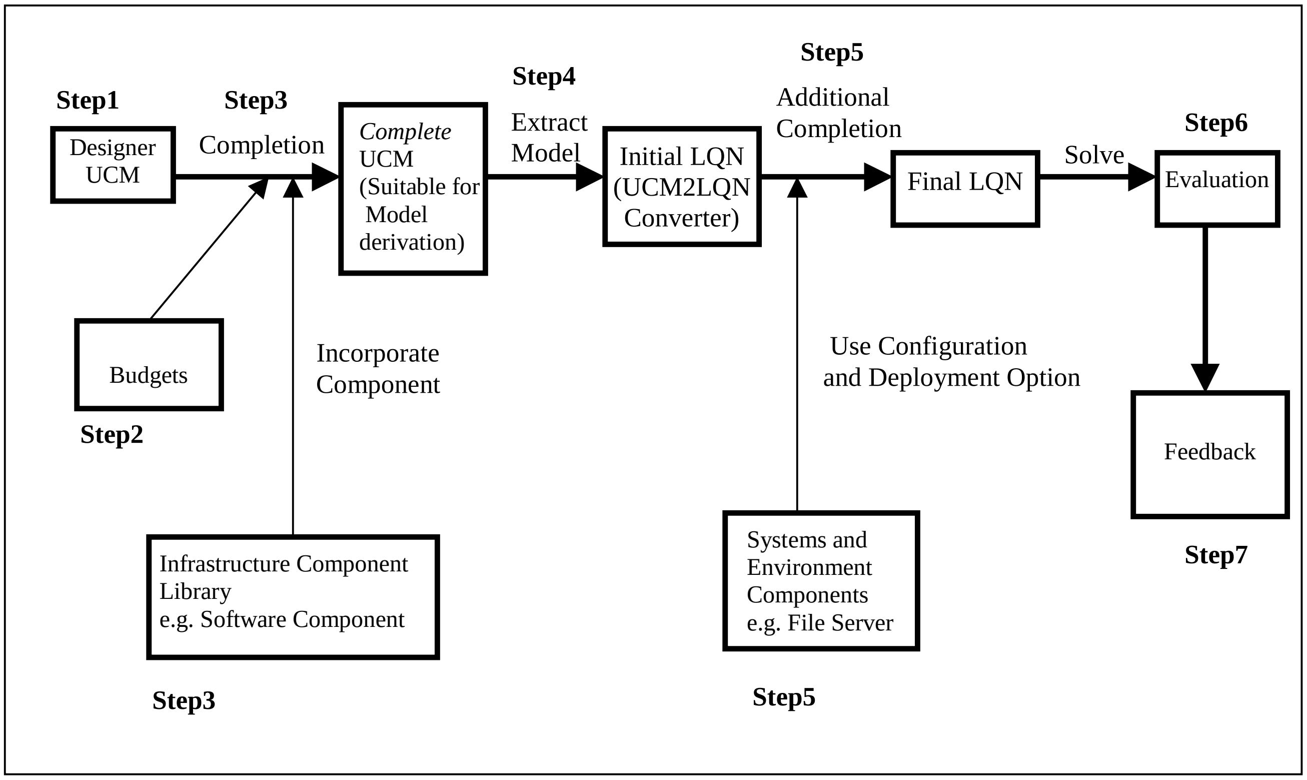

Step1
Designer UCM
Step3
Completion
Complete UCM (Suitable for Model derivation)
Step4
Extract Model
Initial LQN (UCM2LQN Converter)
Step5
Additional Completion
Final LQN
Solve
Step6
Evaluation
Budgets
Step2
Incorporate Component
Infrastructure Component Library e.g. Software Component
Step3
Use Configuration and Deployment Option
Systems and Environment Components e.g. File Server
Step5
Feedback
Step7

Figure 3.1: Budget Analysis Road Map

While the designer UCM may be incomplete in the sense of abstraction, but "completion" must be suggested or defined from somewhere and somehow, sufficiently to allow us to add the missing pieces. For example if a tuple space is to be used for communication, this might be information additional to the Designer UCM (rather than included in it) also the choice of a particular server.

A UCM is not the only possible starting point for budget analysis. Another form of specification, which identifies the scenarios, the components, and the activities for which code is to be developed, could be used. For instance an executable state-machine specification in ObjecTime was considered in [Hrischuk95], and using Specification and Description Language (SDL) in [El-Sayed98]. It might be possible to use a Unified Modeling Language (UML) specification, and a profile for UML has recently been defined to support such an analysis [OMG01]. The advantages of the UCM are that it is a small compact abstract description of the architecture, which can be created before these others and can lead into them [Petriu01a]. The next chapter will describe the idea of Designer UCM in detail before describing the process of "completion".

The designer UCM may or may not define the components and infrastructure that will be included in the final system; these are the subjects of Step 3, the "completions", of road map.

3.3.2 Step2: Demand Budgets

The budgets are values for workload demands by the responsibilities in the UCM. Demands may include CPU operations and storage and communications operations, and may be in units of operation counts, or CPU-seconds on a known processor. Different performance patterns can be described for the users for choosing first budgeted values for the CPU demands or improving these budgeted values for the desired performance. Smith describes performance patterns in [Smith01].

Various performance patterns are described for choosing the initial budget values. These are:

Assumed budgets

If a software system consists of the components, which are new, unfamiliar types or used first time in the budget analysis then CPU demand budgets of such components are assumed or guessed by the design team.

Derived budgets

If the system components are new but of familiar types then the budgeted demand values can be derived from the performance requirements themselves (dividing up the allowed delay into parts allocated to different responsibilities). Or the data for the CPU time demands of the system component is not available then the sensitivities of the performance metrics can be considered as targets to budget the CPU demand for each responsibility.

Reused budgets

If the system components have already used in the some experiments then the CPU demands of the components can be determined, which would be based on the past experiments and results.

If the initial budgets are unable to provide satisfactory performance then budgets can be revised to provide satisfactory performance. The performance patterns for revising the budget are described as under:

Budgets revision

If the desired performance targets are not met then it is required to refine the budgets. It can be done make by making adjustments in the CPU demand of one operation by reducing the CPU demand time and by allowing more time to other operations. Since every operation is time bounded that's why time reduction must be within allowable limit. The example of such systems is mission critical or real time system where the operation CPU demands are time restricted and the operations finish their jobs within the specified time constraint. Therefore, if the budget revisions are not possible in such system then the architecture adjustment can be an alternate option for achieving desired performance goal. Time allowance from one operation to the other operation is acceptable in non-real time systems for the budget revision and refinement where one operation time is reduced and the other operation is allocated more time to finish its job for the desired performance.

3.3.3 Step3: Completion at the UCM level

The software design normally does not fully define the system to be deployed. There may even be multiple deployments in different environments, and with different components. For performance analysis some of these deployments must be defined at least approximately; the amount of detail to be included is a matter of judgment. The missing detail can be supplied at the UCM level, if it arises at particular points in the scenario, or at the LQN level, if it describes a service or infrastructure operation that is used by the system. Often a given “completion” can be added at either level; there is no hard rule.

UCM “completions” are taken from a library of UCM specifications and are added as stubs to the designer UCM. A stub represents an operation with detail defined in a hidden sub-map. Thus the analyst must explicitly indicate where they are to be added. In future it might be possible to add these stubs automatically guided by rules or preferences input by architecture team. Chapter 4 will describe “completion” of an incomplete software specification in details.

3.3.4 Step4: Create the Performance Model

The fourth step of road map is LQN model extraction from the “complete” UCM. The detail of LQN model and available LQN tool are already described in previous chapter.

3.3.5 Step5: Completion of the LQN Model

The LQN model is completed by adding details of the execution environment in the form of hardware as well as software. Examples of “completions” include file servers, protocol stacks, web servers, network latencies, network mechanisms, and processor speeds. These may be defined at this step if it was not done earlier. Chapter 5 will describe LQN Completion Filter. This tool is used to insert network components that handle a call between tasks. The inserted elements are bound to the source and destination tasks of the call, and either to existing processors or to new devices of their own. The LQN Completion Filter has four variations:

- Single Network Component Task (SNCT) filter inserts a single network component LQN task representing a network by replacing single call,
- Network Subsystem (NSS) filter inserts a LQN subsystem by replacing a single call,
- Shared Resource Server Task (SRST) filter inserts a server LQN task by replacing multiple calls, and
- Multicast Network Task (MCNT) filter inserts a single entry LQN task by replacing multiple calls.

3.3.6 Step6: Evaluation

The performance model is solved and the results are compared to the required values. One modality is to use the performance results to verify that, if the budgets are met, the performance will be adequate. A second modality is to derive equipment capacities (such as processor speeds) that, given the budgeted demand values, will allow the performance goals to be met.

3.3.7 Step7: Feedback

Feedback depends on the evaluation results. If they are not satisfactory then some changes are required. If predictions show inadequate performance, one approach is to tighten the budgets until the predictions are in the green zone. Alternatively one could adjust the design in the UCM domain, or the implementation options in terms of “completions” and the environment.

The next section will describe a Simple Call Agent Use Case example, which will walk through budget analysis road map.

3.4 Example: Simple Call Agent Use Case Map (UCM)

3.4.1 Step1: Designer UCM

Figure 3.2 shows the Designer UCM of a Simple Call Agent UCM [Siddiqui00]. It shows two scenarios, one for a Simple Call Agent (SCA) and the other for the Name Service (NS). The first scenario is a call connection from Call Agent A to Call Agent B. x1, x5 and x9 show the responsibilities and operations of the respective components. The second scenario is of Name Service; it is from Call Agent A to Name Service. This UCM is not complete from the point of view of performance model. It lacks some necessary details and component.

3.4.2 Step 2: Budgets

The CPU time demand of each responsibility is chosen as 1.0 msec in beginning of the budgeting process. Since components used in this example are new and unfamiliar types, therefore, sensitivity of system response time and throughput are considered as targets to budget CPU time demand for each responsibility.

3.4.3 Step 3: UCM Completion and Infrastructure Components

The analysis of Designer UCM reveals that the missing component for this scenario might be a communication infrastructure. Communication infrastructure may involve a point-of-point communication, Remote Procedure Call (RPC), Inter Process Communication (IPC), a black board system, or tuple space between Call Agent A and Call Agent B.

In order to generalize communication component insertion, a point-to-point communication infrastructure is considered for this scenario and network delay and protocol agents are introduced between these parties. In Figure 3.3, the UCM is "completed" by adding infrastructure components (protocol agents at Party A and Party B, and a network delay) and their responsibilities, just for the call connection scenario. The protocol agents carry out low-level functions (connection, data checking, flow control). There could be additional completions to describe the message exchanges in greater detail, in both scenarios, but this is sufficient for illustration.

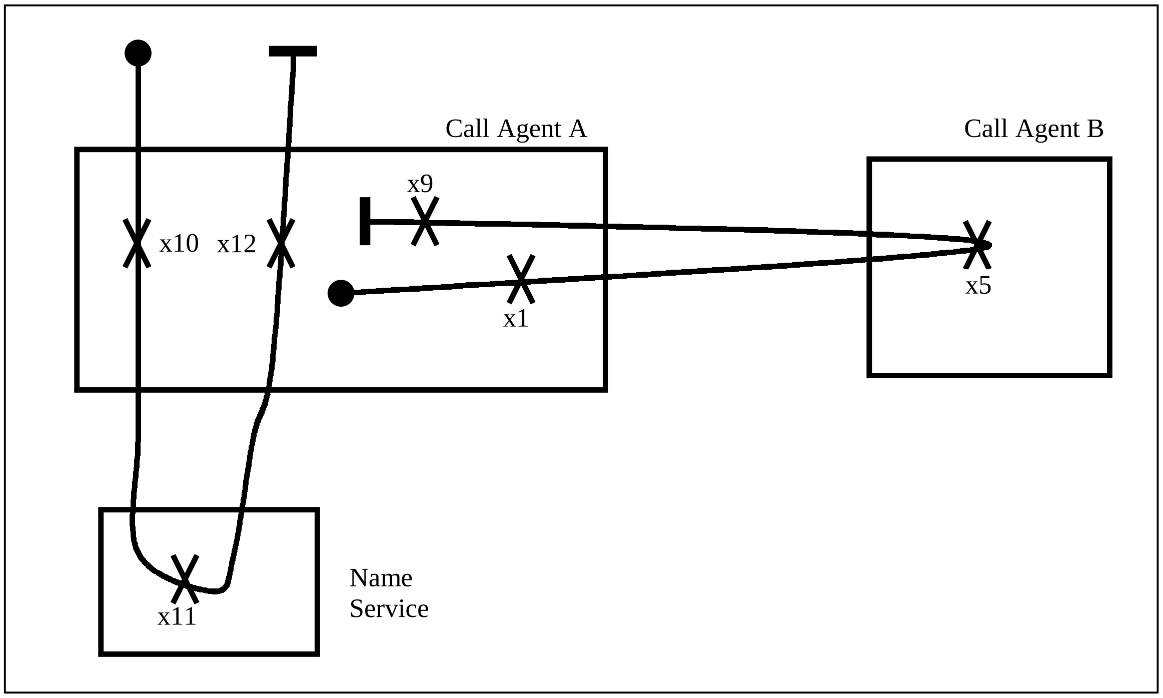


Figure 3.2: Simple Call Agents UCM (Designer UCM)

3.4.4 Step 4: LQN Model of Simple Call Agent

Figure 3.4 shows the performance model of Simple Call Agent, which is extracted, from “complete” UCM. It shows that some components are additional to the information provided by “complete” UCM. These are systems and environment components, which are essential for a performance model of a system.

There are four user tasks representing the external participants in the scenarios, namely “NSUser”, “RemoteUser”, “WLUserA” and “WLUserB”. “NSUser” is a source of name service user requests, “RemoteUser” indicates request of those users who are simply trying to connect from Call Agent A to Call Agent B. “WLUserA” and “WLUserB” have been added to represent sources of competing workload at the nodes executing the scenarios and will be explained in next step.

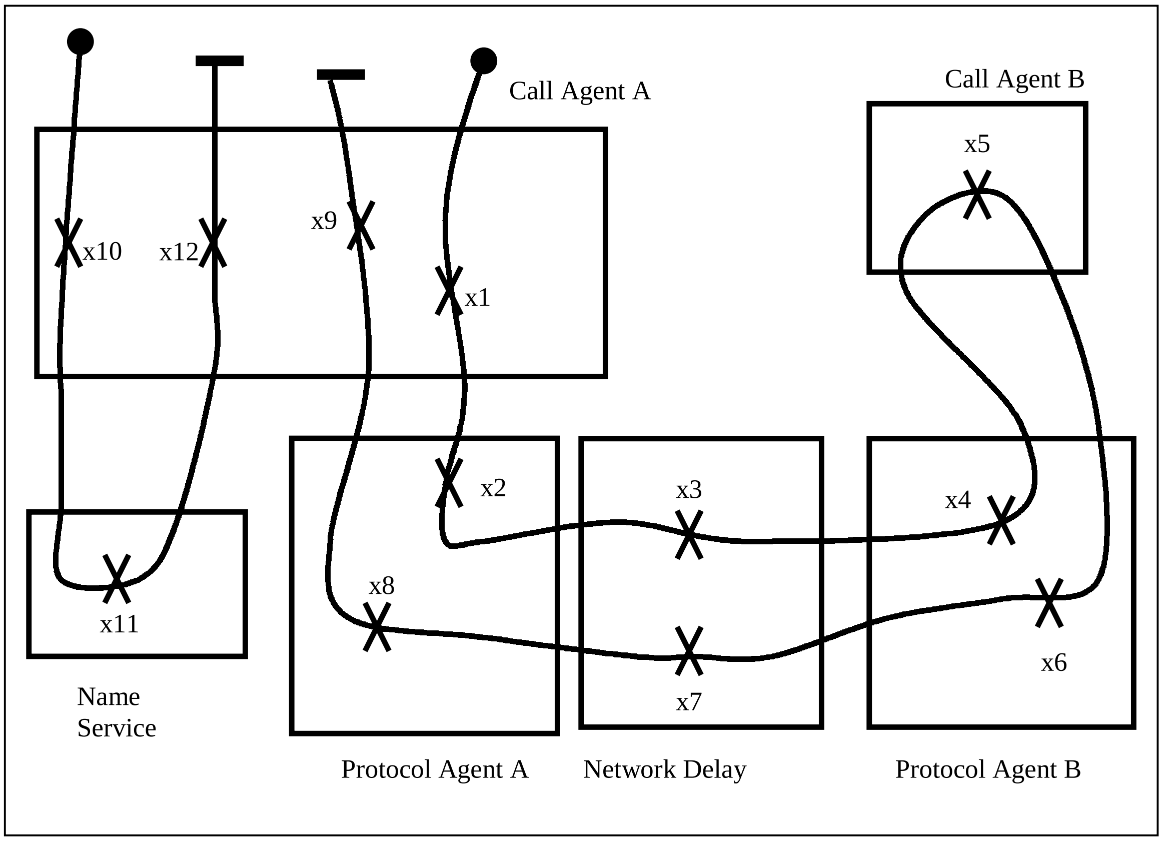


Figure 3.3: Simple Call Agents UCM (Complete UCM)

Call Agent A is represented by an LQN task “CallAgentA”. This task has two entries. “CallAgentAL” entry handles Name Service users and “CallAgentAR” handles requests for call connection to Call Agent B. The two threads of this task are considered for LQN model. The “CallAgentAL” entry also calls the “NameService” task, which processes name service requests. In order to process Name Service request it accesses the disk. The disk or file service is shown as “Disk”.

3.4.5 Step 5: Additional Completions, Environment and Assumptions

“Additional completions” are introduced in the form of environment components. In Simple Call Agent performance model, these environment components are introduced as competing workloads. These workload components are added in the form of competing workload users “WLUserA”, “WLUserB”, “CompetingWLA”, “CompetingWLB, “CresourceA”, and “CresourceB” task.

The “CallAgentAR” entry processes the calling scenario by calling the “ProtocolAgentA” and “NetworkDelay” tasks. At the same time it is competing for the node resource “CResourceA” with “ProtocolAgentA” and “CompetingWLA” tasks.

“CompetingWLA” is an abstract workload, which represents all the competing workload at node A, driven by the source “WLUserA”. It can be adjusted to represent different environments and levels of business of the entire system.

For the example, we need to create budget values for the parameters:

- CPU time demand for each responsibility except those shown in the network delay
- Disk demand (how many accesses) for the name server responsibility x11
- Network latency values as demands for x3 and x7.

In doing this, we are interpreting responsibilities in the UCM as operations. If a responsibility in the UCM is really an abstract representation of work done by several tasks together, it needs to be refined to a point where it is executed by one component. The analysis assumes this has already been done.

Notice that choosing a budget value is different from making a prediction. If the component being budgeted is of a familiar type, the budget value may be a prediction based on experience; if it is a new and unfamiliar type, then the budget value is properly seen as a target or a commitment. The difference between a prediction and a commitment is subtle but important; anyone may refuse to make a prediction if they lack data, but anyone beginning a project is making all kinds of commitments, and this is just one more.

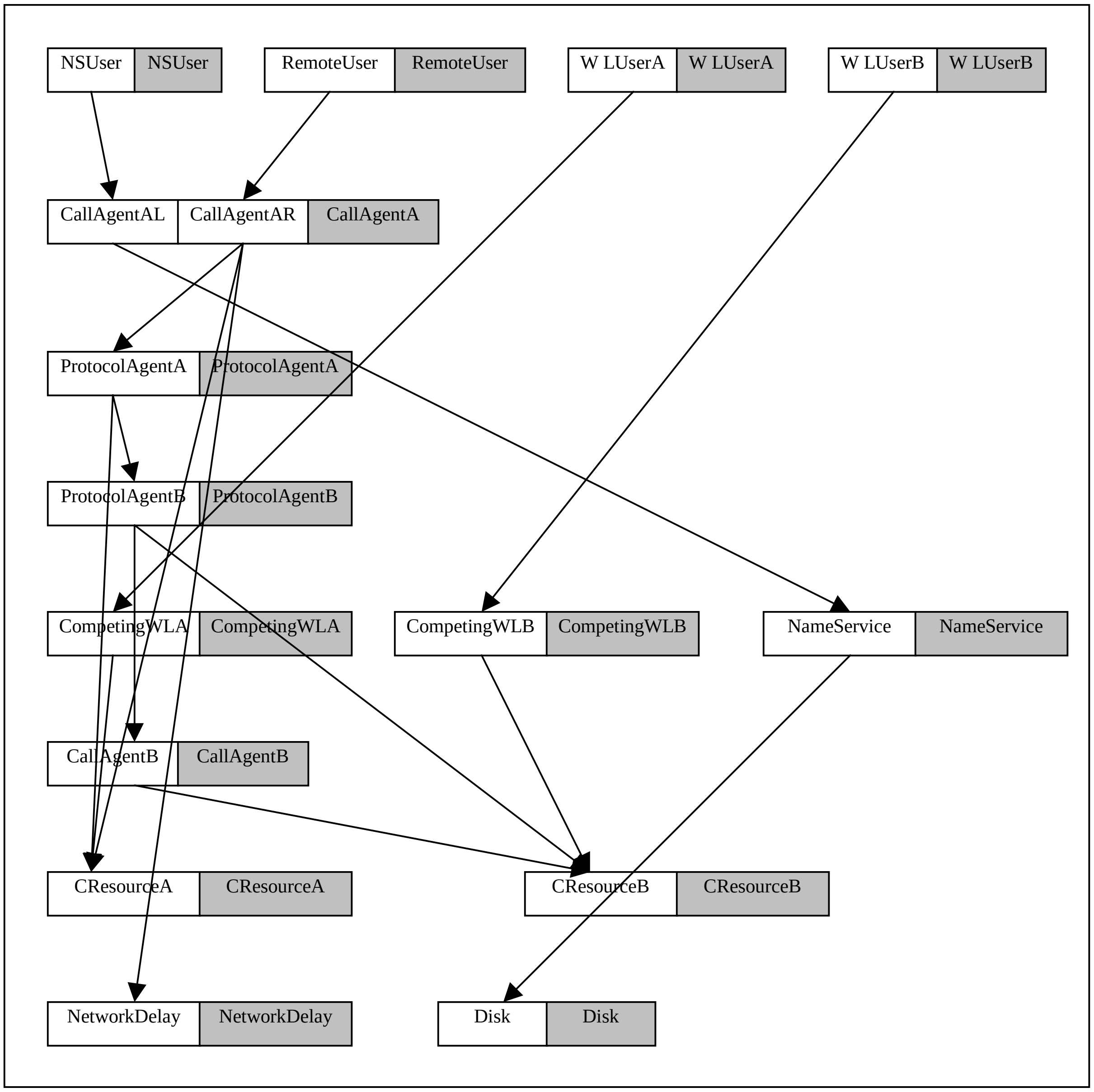


Figure 3.4: LQN Model of Simple Call Agent

3.4.6 Step 6: Experimental Results Evaluation

The LQN analytical model gave the throughput and response time of the remote users, calculated for a range of remote users, for different numbers of name server users. “CallAgentA”and “CallAgentB” tasks are multithreaded. The numbers of threads for both tasks were two. Tools, which exist to make the analysis easier, are the LQNS solver for LQNs, and the SPEX program for sequencing repeated runs over ranges of parameter values. The LQN file of the model is included in Appendix A.The capacity and delay of the system were monitored in term of throughput and response time respectively. Figure 3.5 shows that as the number of name service users increases the performance of the system for call connections decrease. Other runs showed how, when the service time (representing the CPU time budget) of any task increased then the utilization of that task was increased and the remote user performance decreased.

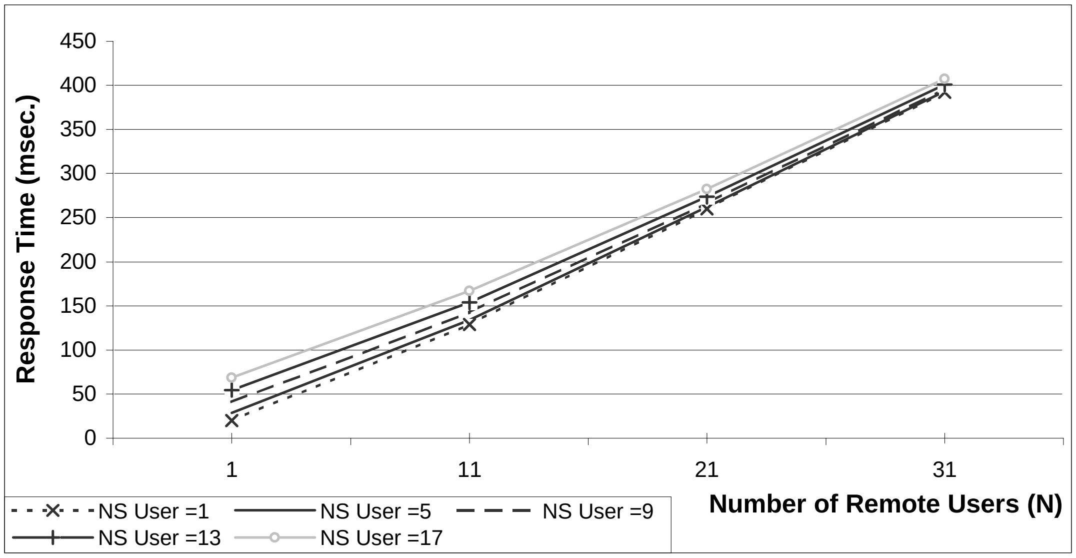


Figure 3.5: Response Time (msec) of Remote Users for Different NS Users

The impact of increased budgets also depends on the location of the task in the layered model. Because, if the task is located at lower layers and its service time is high as compared to the tasks located on the upper layers then this task may cause bottleneck in the system. The bottlenecks of this task also cause bottleneck for upper layers. This effect was observed by increasing the service time of tasks “CallAgentA”, “CallAgentB”, “ProtocolAgentA” and ”ProtocolAgent B”.

Remote Users	NS User =1	NS User =5	NS User =9	NS User =13	NS User =17
1	19.7408	28.5517	40.9504	54.6051	68.7478
11	128.744	133.914	142.47	153.887	167.006
21	260.053	261.985	266.631	273.513	282.212
31	391.934	393.136	396.198	400.923	407.123

Table 1: Response Time (msec)

The throughput was also analyzed and shows the effect on system capacity of changes in numbers of users, in Figure 3.6. The same data is shown in numerical form in Tables 1 and 2.

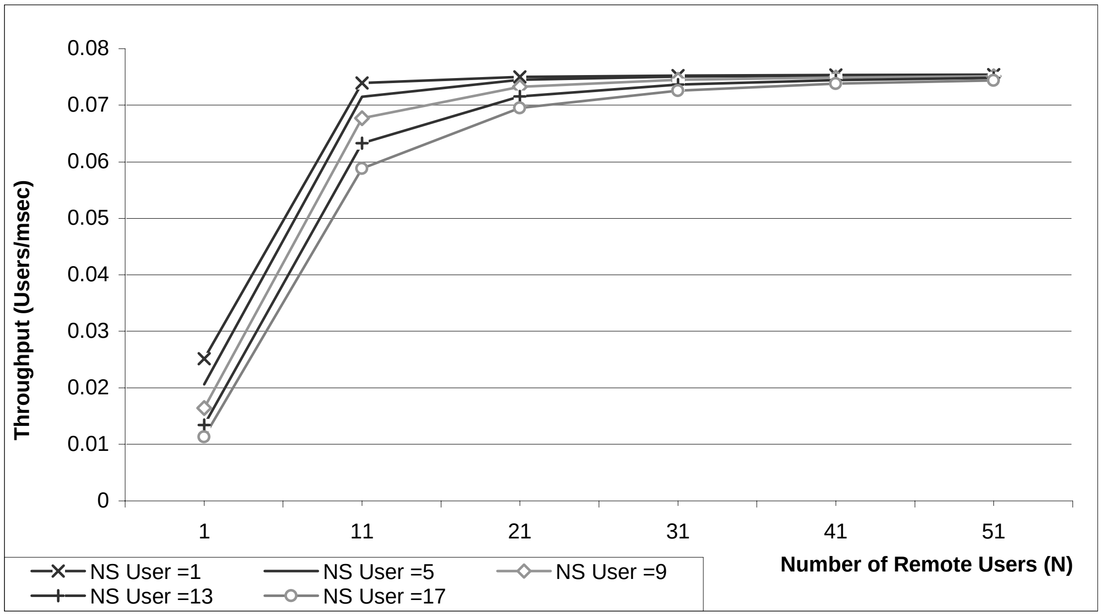


Figure 3.6: Throughputs (Users/ms) of Remote Users for Different NS Users

Remote Users	NS User =1	NS User =5	NS User =9	NS User =13	NS User =17
1	0.025	0.021	0.016	0.013	0.011
11	0.074	0.071	0.068	0.063	0.059
21	0.075	0.074	0.073	0.072	0.069
31	0.075	0.075	0.074	0.074	0.073
41	0.075	0.075	0.075	0.074	0.074
51	0.075	0.075	0.075	0.075	0.074

Table 2: Throughputs (Users/ms)

3.5 Summary

This chapter explains in detail the Budget Analysis Road Map with an example, which highlights the key steps of the performance budgeting approach. These are as follows:

1. Designer UCM as one of the high level specifications of a software design. Any specification can be used a starting point of the performance budgeting road map,
2. Resource demand budgets,
3. “Completion” of an incomplete specification of software design,
4. Creation of the performance model,
5. Additional ”completion” at performance model,
6. Evaluation, and
7. Feedback

4.0 Completion of an Incomplete Software Specification

4.1 Introduction

A software specification does not include all the information, which is necessary for the performance evaluation of a software design, because of its level of abstraction. The information can be some missing software components or operations depending on the performance evaluation requirements and the nature of the system. This thesis uses the term "completion" to mean additional elements, which are required for the performance evaluation of a software system specification, and will be incorporated in an incomplete software specification. Although the addition of the components and operation will result in the modification of software design but it will describe all of the expected components and operations necessary for the performance evaluation.

The performance model requires knowledge of different components and operations of the system. The components can represent software entities such as objects, processes, databases, servers, and networking components (tuple space or blackboard system) as well as non-software entities e.g. or hardware etc. The operations can be CPU operations (associated execution costs) and communications operations, which are required for the workload demands. The chapter will discuss the following issues for the “completion”:

- How should a “completion” be specified?
- Alternatives for the “completion”
- “Completions” at different stages
- Automation for the “completion”
- An example will be discussed in this chapter to elaborate the “completion” and alternatives for the “completion”. The “completion” will be incorporated first and the “additional completion” introduced later if one required for this example.

4.2 How should a “completion” be specified?

“Completion” can be specified at UCM level while the “additional completion” introduced at LQN level. The software designer can insert missing components and operations at UCM level for the “completion”, which are not shown explicitly but can be inferred. These components can be some existing software components, represented in the specification by design stubs and will be used as re-used software components. For example, a “completion” might refer to a missing network component used in communication between different objects. It can be single or multiple components. Network components may not be important for a software design scenario but are required for a performance model. Components and operations can also be incorporated at LQN level. These can be introduced in the form of software components or performance parameter such latency (adding think time in latency) or message overhead for execution demands depending on the requirement of system.

The “completion” can be a replacement of some components in a specification or model with some other components e.g. a plug-in map replace a stub in a UCM specification or a task replacement with subsystem in LQN model.

The “completion” can also be a pattern that can be instantiated or transformed in specification e.g. adapting a plug-in to a use or adapt the scenario to the plug-in in a UCM specification or the communication pattern transformation, which will be described in next chapter.

Since, the “completion” and the “additional completion” may be approximate in both functional and structural terms, or may define workload and demand abstractions, and are for the performance evaluation of a software design therefore they are different from the functional refinements of a specification, elaborations by generative programming [Czarnecki01], or software components in component-based software engineering [Szyperski98].

In this chapter, some examples of the “completion” will be described in UCM domain as well as the “additional completion” in LQN domain and some examples will be described only as the “additional completions” in the LQN domain in next chapter.

At this stage, the “completions” are introduced manually. Since all the tools are not available for the “completion” and the “additional completion” that’s why semi-automation, which refers to manual work and available tools help, is preferable at this stage.

4.3 Alternative “completions” for Inter-component Communication

There can be alternatives for the “completion” of inter-component communication. The selection of missing components and operations is also important from the “completion” and performance model viewpoint. It depends on the scenarios and nature of a software system. It can be understood with a paradigm of two parties communication system shown in Figure 4.1.

4.3.1 Example: Two-Party Communication

Two parties, “Party A” and “Party B”, are interacting each other in a UCM scenario. “Party A” sends a request to “Party B”. “Party B” accepts the request. The responsibilities associated with each party are labeled as “x1” and “x2”. These responsibilities represent generic processing (actions, activities, operations or tasks etc.). This scenario is illustrated in Figure 4.1. This information is not sufficient for two parties to communicate with each other. The communication type is also unknown. It is important to know whether they are located on same machines or different machines. It is desired to choose a suitable a network component between them.

Different type of communications can be possible between these two parties. It can be a point-to-point communication, Inter Process Communication (IPC), Remote Procedure Call (RPC), an Object Request Broker (ORB) [Mowbray97 and OMG95], a blackboard system [Cabri98 and Spruit97], or a tuple space communication. The choice of network components depends on the location of components whether they are located locally or remotely and their interaction type such as synchronous, asynchronous or forwarding.

It can also be anonymous communication i.e. one party does not have to know the other party. If both parties are on local machines then IPC is a good choice, if they are remote machines then RPC is a good choice, if both parties have dissimilar operating systems or hardware then ORB will be a good choice. Tuple space communication will be a suitable choice of communication when both parties are on local or remote machines and interacting anonymously. Tuple space server can be a meeting point for both parties. The choice of tuple space server is also important whether it is centralized or distributed.

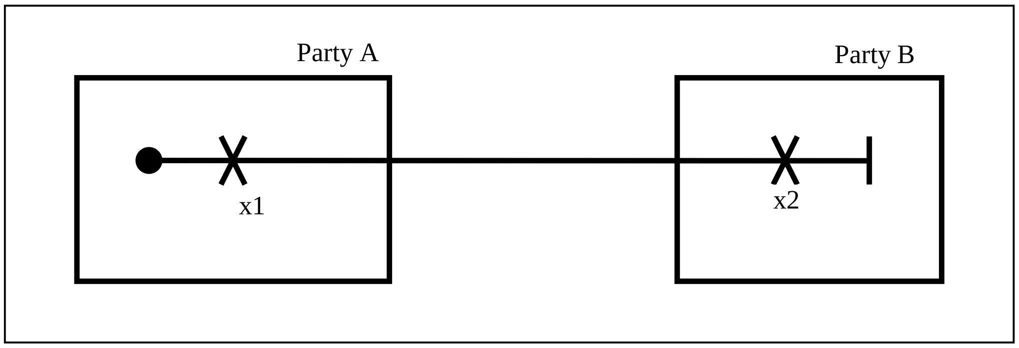


Figure 4.1: Two-Party Communications

4.3.2 Alternative: “Completion” by Simple Network Messaging

Consider simple network messaging case for the “completion”. The network component is added between two parties for the “completion”. It is assumed that parties are located remotely. The root map is shown in Figure 4.2.

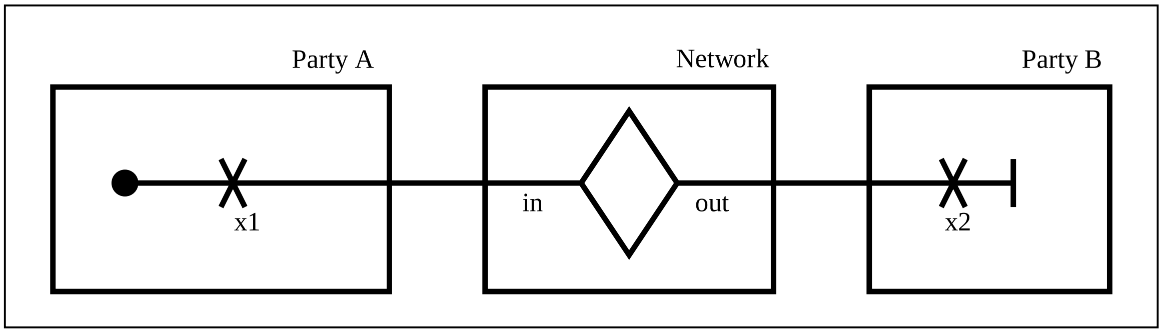


Figure 4.2: “Completion” with network messaging

Network component has a static stub, which represents a sub scenario. The plug-in map associated with this sub map is shown in Figure 4.3.

Sub scenario shows “Protocol Agent A” for “Party A”, “Protocol Agent B” for “Party B” and “Network Delay” component. The choice of protocol is also important for network message system. It is assumed that the protocol agents of both parties are part of network component and this is sufficient for the “completion’ illustration.

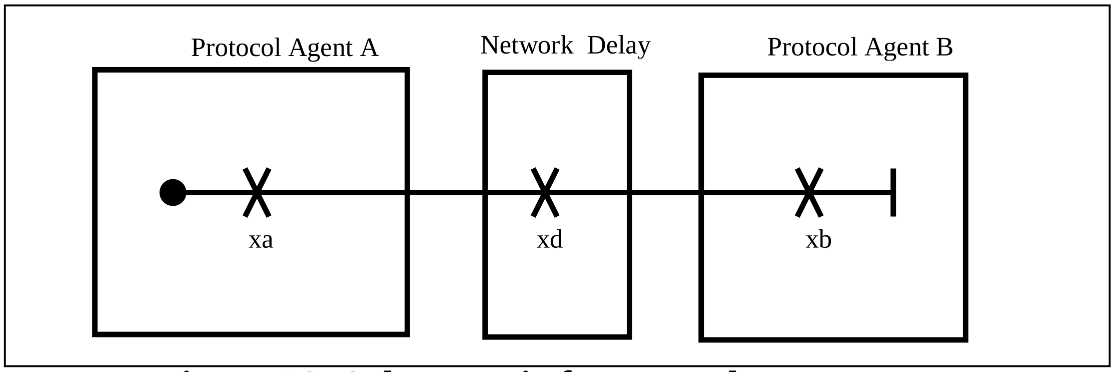


Figure 4.3: Sub-scenario for network component

4.3.3 Alternative: Completion By Tuple Space

Consider tuple space as a suitable communication component for the "completion". Tuple space is added between two parties for the "completion". It is assumed that "Party A" is communicating with "Party B", this is one-way communication and both parties are remote parties. A tuple space server is coordinating communication between these parties. The choice of tuple space server is also important whether it is centralized or distributed. Centralized tuple space is selected for this example. For simplicity, only two of the tuple space operations, **in** and **out**, will be used in this example. The **out** operation is used to place tuple in the tuple space and **in** operation is used to remove tuple from tuple space. **in** is a blocking operation. The other operations like **rd**, **rdp**, **eval** and **in**p are not important for this example. The details of these operations are already described in Chapter 2.

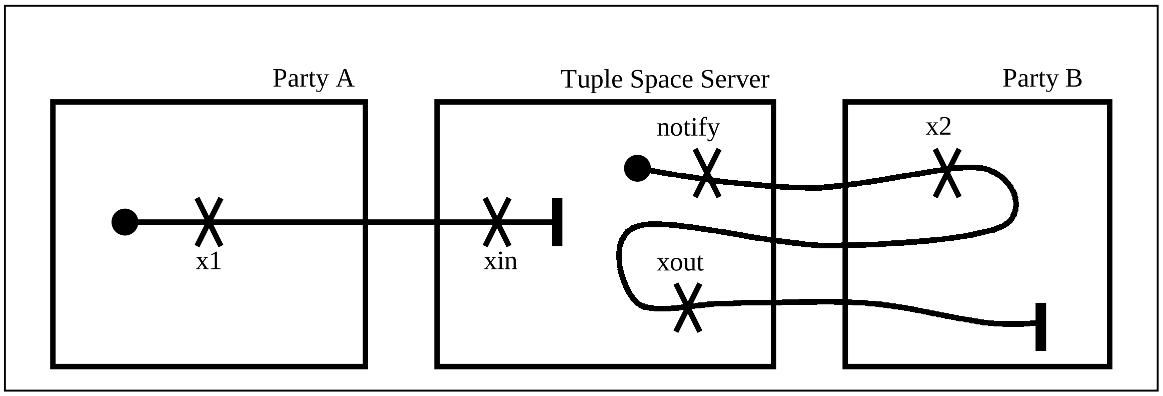


Figure 4.4: Two-Party Communication using tuple space for "completion"

"Party A" places its tuple in tuple space server using **out** operation. The tuple space server notifies "Party B" about the tuple. It is assumed "Party B" would be blocked if the tuple is not available in the tuple space. "Party B" will remove the tuple from the tuple space server using **in** operation. If the tuple found in tuple space then "Party B" will remove it. If the tuple is not in tuple space then "Party B" will not wait for the tuple and will continue its normal execution. When the tuple will arrive in tuple space, tuple space server will notify "Party B". "Party B" will remove the tuple from tuple space.

4.3.4 Alternative: Completion by Remote Procedure Call (RPC)

Consider Remote Procedure Call (RPC) case for the "completion". RPC mechanism is implemented at both parties for the "completion". It is a generalized case for communication while ORB is a special case of RPC, which will be discussed in next section. It is assumed that "Party A" is communicating with "Party B", this is one-way communication and both parties are remote parties. RPC mechanism is coordinating communication between these parties. RPC is a client server mechanism. "Party A" is acting as client and "Party B" is acting as server. The non-blocking RPC mechanism is chosen for this example because "Party A" is not blocked after sending the request to "Party B". Athena RPC at MIT [Arons], Concert-C [Auerbach94] and ASTRA [Ananda91] are examples of non-blocking or asynchronous RPC.

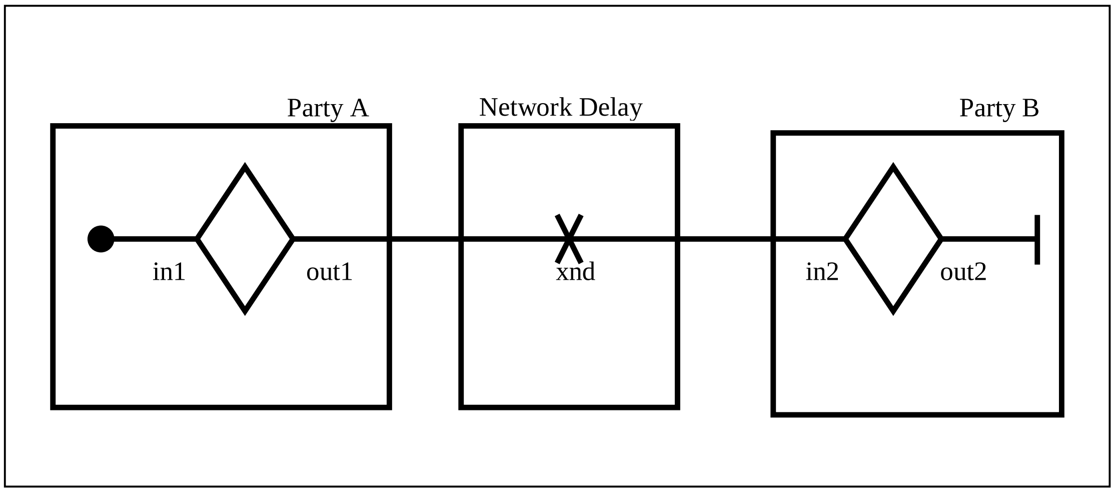


Figure 4.5: Completion with RPC

Before introducing non-blocking RPC for “completion” some components associated with RPC will be discussed.

Client Stub: It performs packing and unpacking of data. Packing refers to the conversion of procedure names and parameters into a message. It is called marshaling of data. Unpacking refers to conversion of message coming from server into results. At link time client is statically bounded to a procedure that spans address space over the network to communicate with the remote server. It takes help from name server.

Client Transport Mechanism: It refers to communication network software. It converts the call into send operation. In non-blocking RPC case the client continues its normal execution after making request to server.

Server Stub: It performs packing and unpacking of data. Packing refers to the conversion of results into message. It is called marshaling of data. Unpacking refers to conversion of message into procedure arguments i.e. un-marshaling of data.

Server Transport Mechanism: It refers to communication network software. It receives message from client.

Server Process: It’s surrogate process that executes the procedure. The process accepts call from appropriate client and makes local call.

Two party-communications example is not completed from the point of view of RPC communication. RPC components of both client and server are not shown in Figure 4.1 “Designer UCM”. RPC components are implemented in the form of static stub at both parties as shown in Figure 4.5.

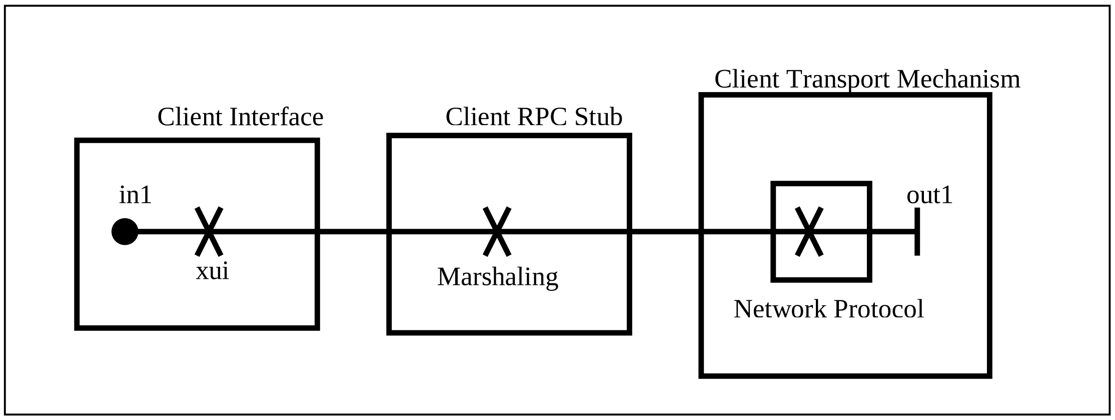


Figure 4.6: Client Scenario at Party A

Figure 4.6 shows the client side RPC sub-scenario, which is introduced as part of “completion” at “Party A”. The associated responsibility with client interface is labeled as “xui”, which shows user interaction with the client side object. The responsibility “marshaling” at stub provides marshaling of data at “Party A”. The third rectangular box shows Client Transport Mechanism and the network protocol associated with it. This provides network communication facility to “Party A”.

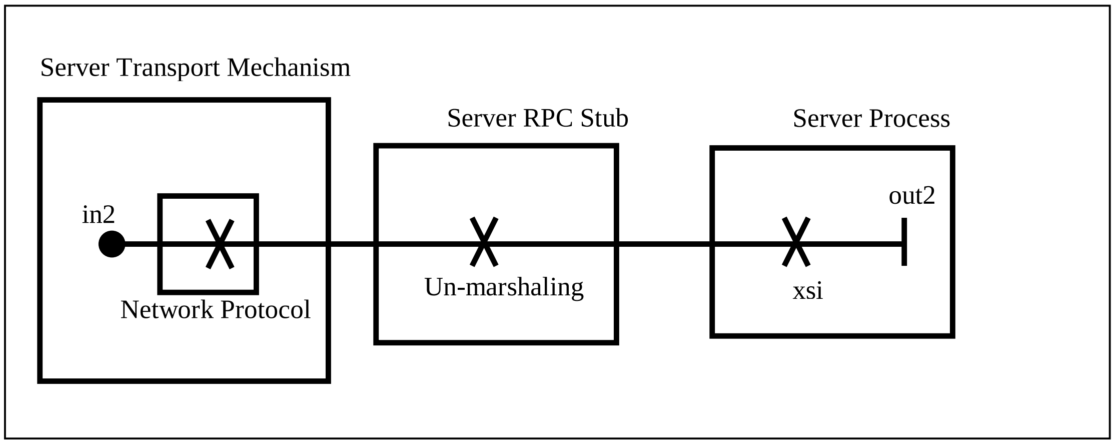


Figure 4.7: Server scenario at Party B

Figure 4.7 shows the server side RPC sub-scenario, which is introduced as a part of “completion” process at “Party B”. The first rectangular box shows Server Transport Mechanism and the protocol associated with it. This provides network communication facility to “Party B”. The responsibility “un-marshaling” at stub provides un-marshaling of data at “Party B”. The associated responsibility with server interface is labeled as “xsi”, which shows client connection with the server side object.

Network delay component introduced as part of “completion” process shown in Figure 4.5 The associated responsibility is labeled as “xnd”.

4.3.5 Alternative: Completion By Object Request Broker (ORB)

Consider an Object Request Broker (ORB) case for “completion” process. ORB is added between two parties for “completion”. It is assumed that “Party A” is communicating with “Party B”, this is one-way communication and both parties are remote parties. An ORB is coordinating communication between these parties. The choice of ORB depends on the implementation and the vender of ORB. For ORB communication it is assumed that "Party A" is acting like as a client and "Party B" is acting like as a server. It is also assumed that both parties are running same vender ORB though it can be from different venders.

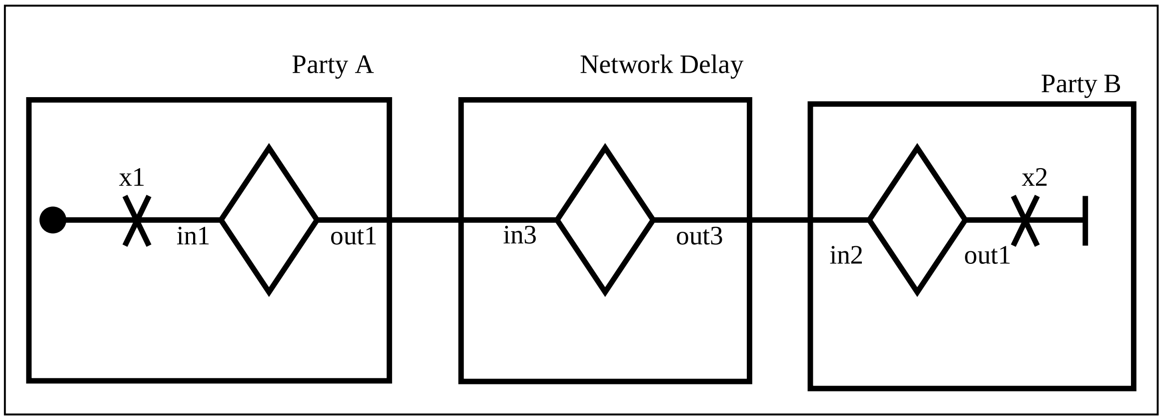


Figure 4.8: "Completion" with ORB

ORB is the infrastructure mechanism standardized by Common Object Request Broker (CORBA). The role of ORB is to unify access to application services, which it does by providing a common object-oriented, remote procedure call mechanism. CORBA provides location transparency i.e. the difference in the location of the objects. Objects can be local to the client i.e. on the same machine; or they can be on the remote machine; or on different platforms or implemented in different languages.

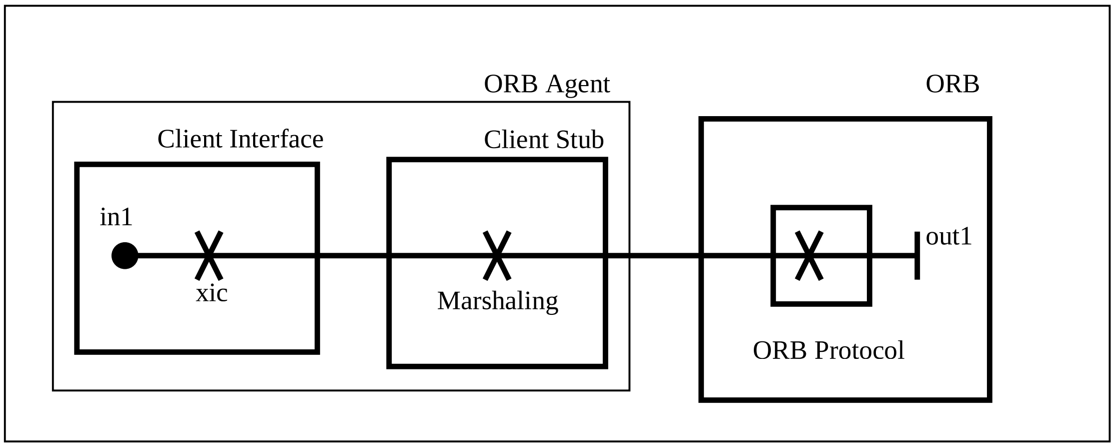


Figure 4.9: ORB Sub-scenario at Party A

Usually ORB is implemented as runtime library. Client and server both have these libraries. Client implements the link to server through communication port using standard ORB protocol. Client calls the stub, marshals the data and send the request to client side ORB. Client side ORB connects server side ORB though communication port using standard ORB protocol.

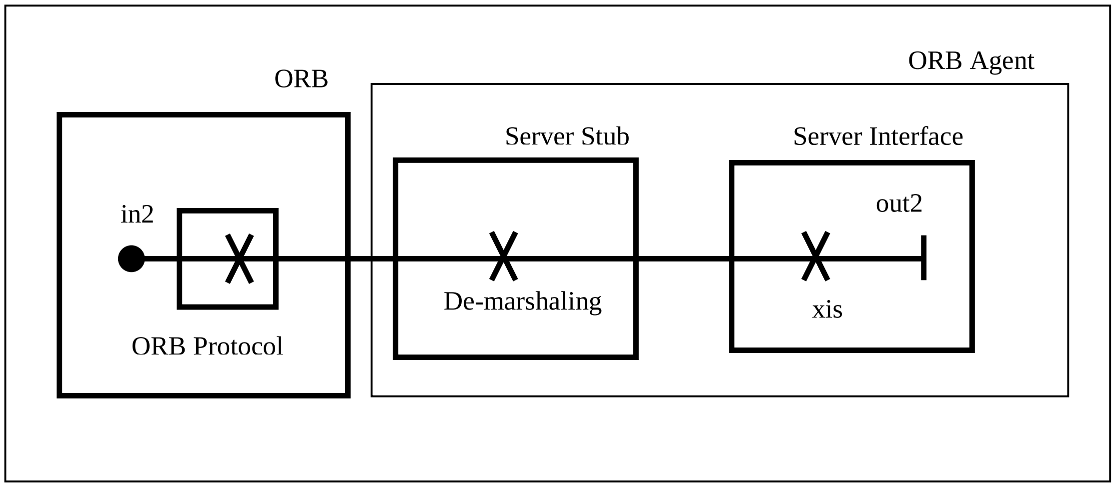


Figure 4.10: ORB Sub-scenario at Party B

Server un-marshals the data, accepts the request and does processing according to the request. If it is synchronous message then server marshals the reply, passes this data to server side ORB that communicates client side ORB, and client un-marshals the data.

Stub provides marshaling of parameter information from language-dependent interface. It translates high-level language parameters to a form for transmitting information across the network.

Interface is a link between user and object on the machine. The static interface is assumed for this purpose. Static interface shows that type of information may be completely specified when the software is written at compile time so that when the stubs are generated, the particular ORB product may use this compile-time information to provide highly optimized marshaling transformation.

The choice of protocols is ORB-implementation dependent, and the client does not have to have explicit knowledge of which protocol is being used.

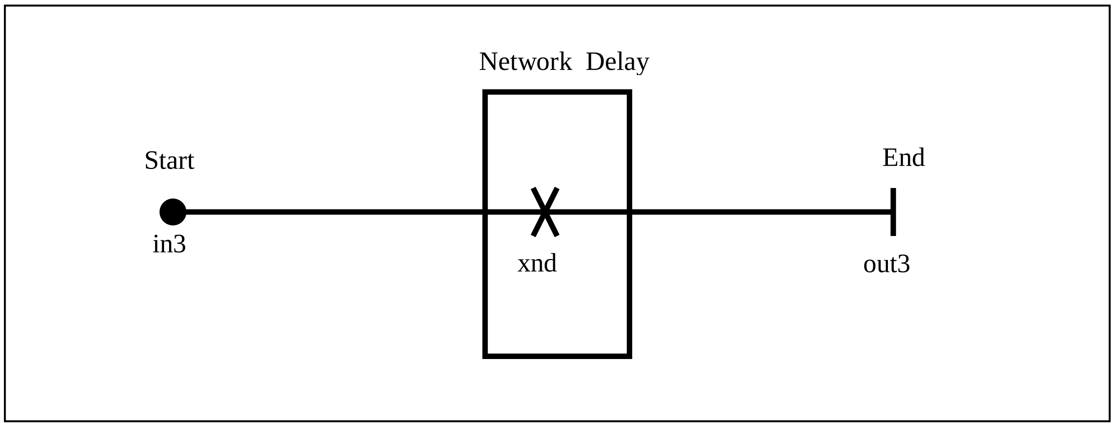


Figure 4.11: Sub-scenario for network delay

Two party-communications example is not completed from the point of view of ORB communication. ORB components of both client and server are not shown in Figure 4.1 “Designer UCM”. ORB component is implemented in the form of static stub at both parties as shown in Figure 4.8.

Figure 4.9 shows the client side ORB sub-scenario, which is introduced as part of the “completion” at “Party A”. The associated responsibility with client interface is labeled as “xic”, which shows user interaction with the client side object. The responsibility “marshaling” at stub provides marshaling of data at “Party A”. The third rectangular box shows an ORB and the protocol associated with it. This protocol will connect “Party A” to “Party B”.

Figure 4.10 shows the server side ORB sub-scenario, which is introduced as a part of the “completion” at “Party B”. The first rectangular box shows an ORB and the protocol associated with it. This protocol will connect “Party B” to “Party A”. The responsibility “un-marshaling” at stub provides un-marshaling of data at “Party B”. The associated responsibility with server interface is labeled as “xis”, which shows client connection with the server side object.

Network delay component is shown in Figure 4.11 as sub-scenario. The associated responsibility is labeled as “xnd”. This is also introduced as part of “completion”.

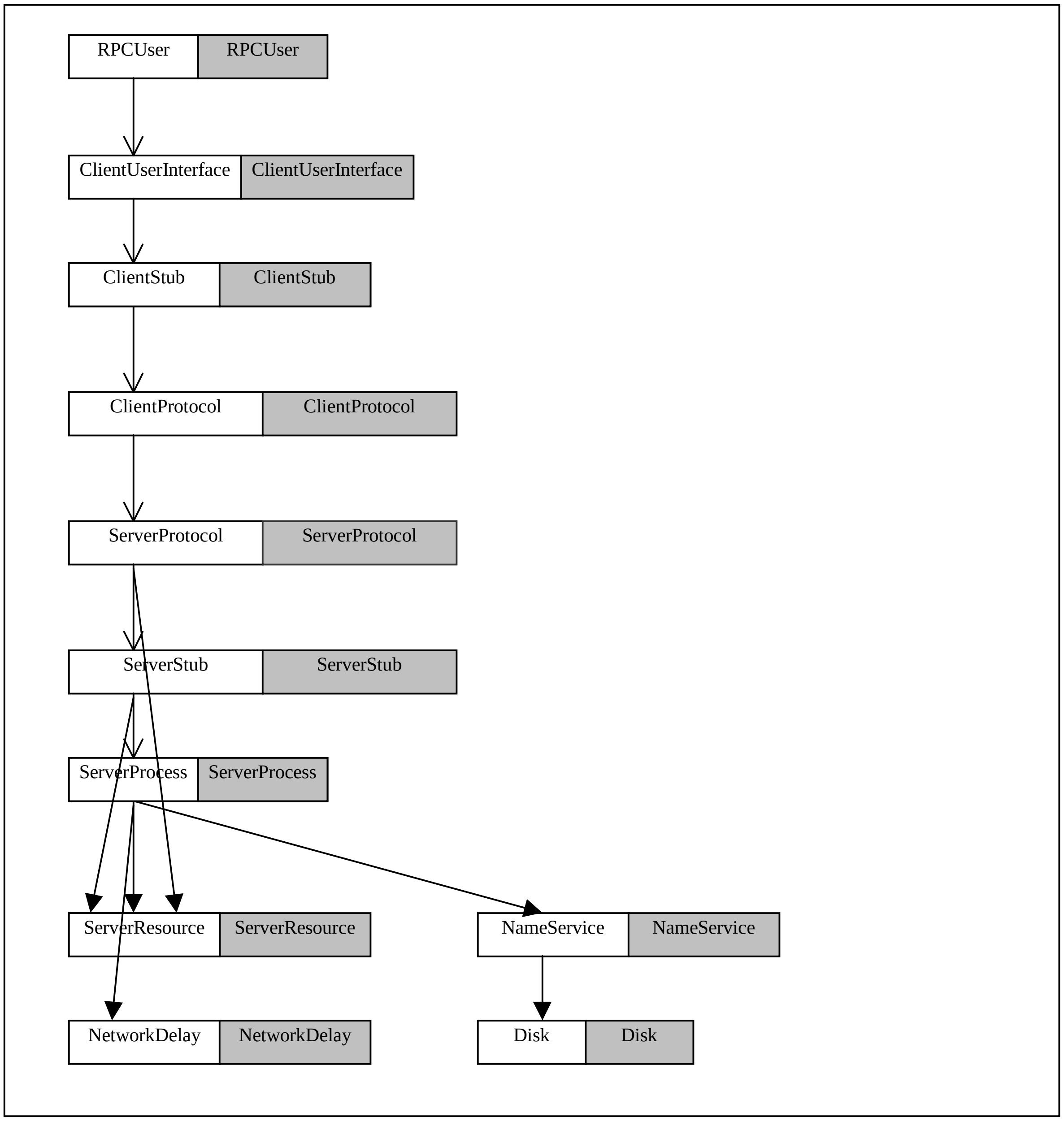


Figure 4.12: LQN Model of RPC

4.4 "Completions" at different stages: Additional 'completion" in LQN domain

The term additional "completion" will be used to refer to the notion that some network components have been inserted at UCM level but the details associated with these components can not be shown. Associated details of network components can be latencies and message overheads, which cannot also be shown at UCM level so it can be incorporated in the form of additional “completion” at LQN level. Latencies can be incorporated in the form the User Think time and message overheads in execution demand. Figure 4.12 shows completion at LQN level.

4.4.1 Example: Additional Completion for RPC

Figure 4.13 shows the LQN model with additional “completion”. Each task has single entry. The detail of each task is as follows:

RPCUser Task: It shows the user task at “Party A” to make an RPC request.

ClientUserInterface Task: It shows the user interface at “Party A”, through which user will make request for the connection.

ClientStub Task: It shows the stub responsibility at “Party A”. It performs marshaling of data.

ClientProtocol Task: It shows the Network Protocol responsibility at “Party A”. It provides communication mechanism for “Party A”.

ServerProtocol Task: It shows the Network Protocol responsibility at “Party B”. It provides communication mechanism for “Party B”.

ServerStub Task: It shows the stub responsibility at “Party B”. It performs un-marshaling of data.

ServerProcess Task: It performs the request of the client at “Party B”.

WLUser C Task: It shows the client side users, which are competing for the resources at client.

CompetingWLC Task: It shows the client side competing workload task. WLUserC accesses this task to the resource.

ClientResource Task: It shows the competing resource at client side

WLUserS Task: It shows the server side users, which are competing for the resources at server.

CompetingWLS Task: It shows the server side competing workload task. WLUserS users use this task to access server side resources.

ServerResource Task: It shows the competing resource at client side

NameService Task: This task shows a name server. A RPC process is associated with name space. If process would like to access a procedure in another name space a special name to address binding is required. Name server aids in this binding by producing the name to process mapping. The RPC server registers each procedure with name server so the client can locate it at call time.

Disk Task: Name Server accesses the disk.

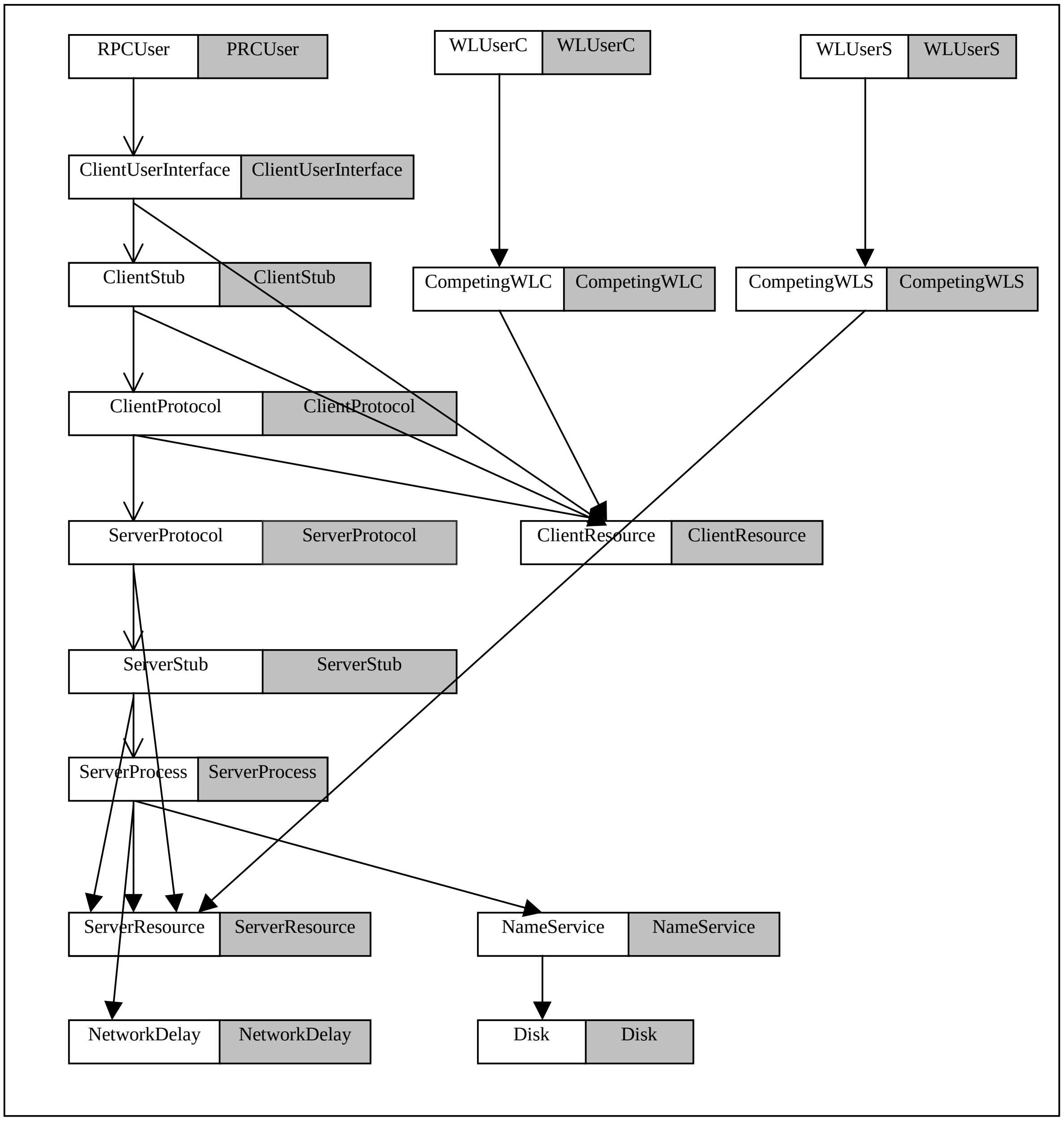


Figure 4.13: LQN Model of RPC

It can be seen from the performance model of RPC example that Name Service task and competing workload tasks are included. The name server is very important for RPC mechanism. This may not be important for a software designer because this information is not important for a RPC mechanism but for performance model it is. The name server accesses the disk, which can affect the performance of the system. The workload users were also not important to show RPC mechanism but they are important for a performance model. Because, there will be some users on client side as well as server side, which will be using both the machines or some other processes will be running at the same time on both machines. These processes will compete for the resources with different RPC processes, which will really affect the overall performance of the system.

In RPC performance model, only network delay was introduced but the execution overheads were not considered. These can also be included as additional “completion” in order to get satisfactory performance result.

4.5 Automation for the “completion”

Different tools can be designed to provide automation for the “completion” at specification level as well as model level. Some of the tools which are already available to introduce “completions” at specification level e.g. UCM Navigator introduces “completions” at UCM level in the form of software component or a UCM plug-in map as a software subsystem. In addition to this, each software component has a performance related window for the information, which includes arrival rates at start points, path choice probabilities, deployment of processes to processors and the workload demands of responsibilities.

Some of the tools, which are already available, insert the “completion” after the model is built e.g. LQN Components Filter and Latency and Message Overhead filter insert the “completions” at LQN level.

These tools do not provide integrated automation for the “completion”. Research and development required in this direction to combine and integrate all the tools to introduce different types of “completions” at specification level as well as model level. All previously available tools can also be combined into UCM Navigator to provide an integrated environment for the “completion”. UCM2LQN Converter is already integrated in UCM Navigator.

This thesis’ contribution for the “completion” is the development of LQN Component Filter tool. It has four variations. This tool introduces additional “completion” in the form of network communication components at LQN level. The mechanism of the design and algorithm will be discussed in next chapter.

4.6 Summarry

This chapter explains in detail one of the important steps of the budgeting road map, the “completion” of an incomplete specification of a software design. The following issues are discussed in this chapter:

1. How should a “completion” be specified?,
2. Alternative “completions” for inter-component communication,
3. “Completions” at different stages, and
4. Automation for the “completion”.

5.0 LQN Component Filter

5.1 Introduction

This chapter will describe the design of LQN Component Filter, which provides automation to the “completion” at model level. The idea of the “completion” is already discussed in previous chapter. There are four variants of the LQN Completion filter. These filters define the “completion” at LQN level. This is done for a call replacement with a network component. It is implemented in form of Single Network Component Task (SNCT) filter, Network Subsystem (NSS) filter, Shared Resource Server Task (SRST) filter and Multicast Network Task (MCNT) filter.

5.2 Motivation for the Design

Distributed and communication software involve many network components. A network mechanism usually involves some latency and delay communication associated with these network components. These network components can be single e.g. a router or multiple e.g. a chain of routers, local e.g. LAN or remote e.g. a RPC client-server connection through Internet, single threaded or multi threaded, depending on the communication types and systems. It is possible to replace a call with a network component by hand but automation will enhance the process of replacement. LQN requires new features such as a call replacement with a particular type of network component by automation.

The definition and type of calls in LQN performance model are discussed in detail in Chapter 2. They represent simple messages from one task to another task (from one entry to another entry) but there are some cases when message passing or simple call representation is not sufficient for a performance model. So, it is required to replace a simple call or a number of calls with one or several network components depending on the type of communication.

5.3 Basic Concept of Network Component Filter

In LQN model, a call is bound with source and destination tasks (source and destination entries) and tasks are bound to processors. These tasks can be on the same processor or separate processors depending on the system requirements.

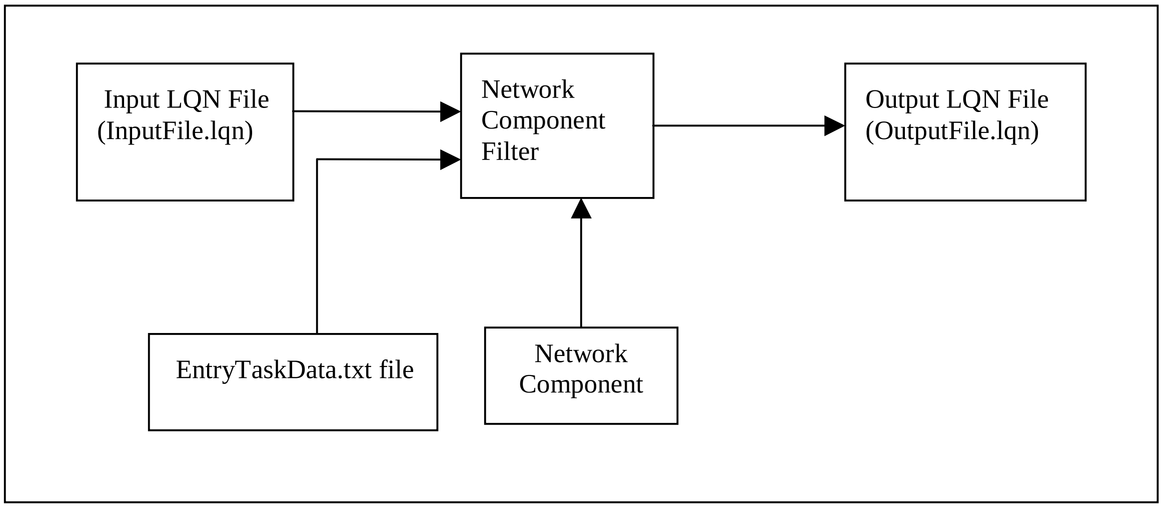


Figure 5.1: Network Component Filter Design

The basic concept of Network Component filter is such that it reads an input LQN file, which defines the model of a software system where a call will be replaced by a network component. It can be a replacement of single call or multiple calls depending on the definition of “completion”. The second argument of the filter is a text file called EntryTaskData.txt file, which defines a set of pairs of entries and tasks. In LQN model, a call is associated with an entry and each entry is associated with a task. Therefore, in EntryTaskData.txt file, first entry is a source entry with an associated task where call is originated and second entry is a destination entry with an associated task where call is terminated. The general syntax of the file is shown below.

<Source Entry> <Source Task> <Destination Entry> <Destination Task>

The third argument of the filter is a network component. A network component can be represented by a single network task or a network subsystem in LQN model depending on the component definition. Therefore, third argument is either the name of the task, if the network component is a single task, or a file name representing a network subsystem. It replaces the desired call with network component and generate output file. The design of the filter is shown in Figure 5.1.

Many network component filters can be designed to achieve different types of “completions” at network components level. Three particular network filters design will be discussed in next sections, which will define different types of “completions”. The sections will cover simple Single Network Component Task (SNCT) filter, Network Subsystem (NSS) filter, Shared Resource Server Task (SRST) filter, and Multicast Network Task (MCNT) filter.

5.4 Single Network Component Task (SNCT) Filter

5.4.1 Motivation For The Design

There are some cases in which a single call represents a single network component. This component might share a processor with some other tasks, which exist in the input LQN file or it might have its own separate processor, which will be incorporated with this task in the input LQN file during the execution of the filter. So, it is required to design such a filter, which will read a set of pairs of source entry with the associated task, destination entry with the associated task, and a processor if the component share the processor, which exists in the input LQN file. It can be a replacement of single call or multiple calls depending on “completion” definitions. For multiple call replacement, all the replaced network components can be of similar or dissimilar types. The example of the former is many routers with same configurations having their own separate processors. In the later case some of the network components would be sharing some processors, which exist in the input LQN file e.g. a modem and some of them would have their own separate processors inserted by the filter e.g. routers. Single Network Component Task (SNCT) filter defines such a “completion” for a distributed communication software system in the form of single network component task.

5.4.2 Design Of The Filter

The design of Single Network Component Task (SNCT) filter assumes that each task will have one or multiple entries and each single network component to be incorporated will share a processor exist in the input LQN model or have a separate processor.

The design of SNCT filter is based on Network Component filter design shown in Figure 5.1. The SNCT filter also accepts three arguments from the user from command line. The first two arguments are the name of two separate files, and the third argument is used to specify network component task name. The first argument is the input LQN file, which is already described in section 5.3. The second argument is a file called EntryTaskDataFile.txt with an added <Processor name> field. The modified syntax of the file is shown below:

<Source Entry> <Source Task> <Destination Entry> <Destination Task> <Processor name>

<Processor name> is the name of the processor, which is optional. If processor name is defined in the EntryTaskData.txt file then it is assumed that this processor is one of the processors, which exist in the input LQN model and the single component will share this processor with other tasks. If processor name is not defined then the filter will insert a separate processor in the input LQN file for the single network component during the execution.

LQN model does not allow two tasks having a common name. The name of each task should be unique. Since, network component is incorporated at run time and it might be possible after inserting one network component in a model the user would like to insert another component, therefore, network component name is specified as third argument on the command line to provide uniqueness to network component task. This will differentiate network component task with other components in a model. Thus the complete command line has the following syntax.

```
$java SNCT model.lqn EntryTaskData.txt NetworkTaskName
```

If there might be a case when it is required to replace number of calls with number of network components at the same time and each network component must be distinguished from other network components. Index will be used with each network component name to differentiate it from other network components. For example, if it is required to replace three calls with three network components e.g. three routers and the task name entered from command line is ‘nt’ then network components to be incorporated will be ‘nt0’, ‘nt1’, and ‘nt2’. Moreover, if the names of processors are not defined in the TaskEntryData.txt file then each network component will have a separate processor since each router is a stand-alone device having its own processor; therefore, in this case, respective processors will be ‘pnt0’, ‘pnt1’, and ‘pnt2’.

5.4.3 SNCT Filter Algorithm

This section provides the algorithm for the SNCT filter for the replacement of synchronous, asynchronous and forwarding calls. Since only one call will be replaced by single task therefore the type of the call is not important. The role of the task is important whether the task will share a processor of the input LQN file or will have a its own separate processor.

The execution steps of the SNCT filter are described as under:

1. The SNCT filter reads the input LQN file, EntryTaskData.txt file and the name of the single task modeling the network and checks the validity of both files and the presence of the LQN task representing a network component in the input LQN file.
2. (a) If another task is already present with the same name in the input LQN file then the filter will generate an exception and halt.

   (b) The relation of the entry and the associated task in the EntryTaskData.txt file is verified for each entry and task pair in the file whether the entry belongs to this associated task. If not then the filter will generate an exception and halt.

   (c) The validity of the call between source and destination entries, defined in the EntryTaskData.txt file, is determined. If there is no such call in the input LQN file then the filter will generate an exception and halt.

(d) If a processor is defined with the pair of entries and the associated tasks (source and destination) in the EntryTaskData.txt file then the validity of the processor is determined in the input LQN file. If no such processor is available in the input LQN file then filter will generate an exception and halt.

3. The filter creates the processor if necessary
4. A LQN task, representing the network, is created with a default delay (network service time) of one unit and will generate output LQN file.

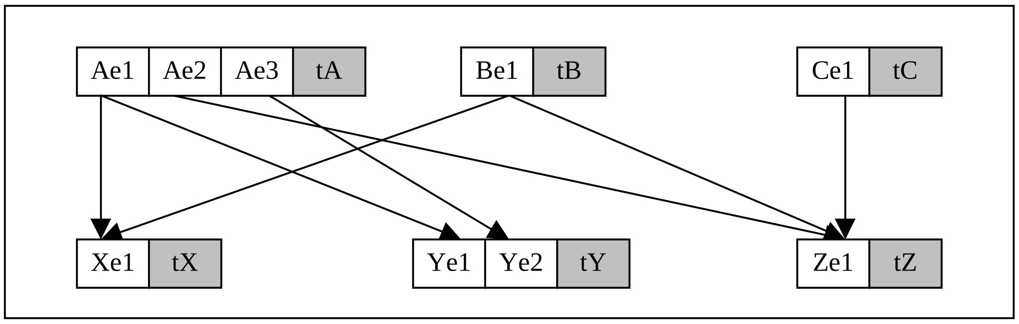


Figure 5.2: Input LQN Model

5.4.4 Example

An input LQN model is shown in Figure 5.2. Each box represents a task in which gray part represents the name of the task and white part represents the entry or entries of the task. Each task has its own processor. All the tasks are interacting synchronously. It is required to replace all the calls between entries Ae1 (associated with task tA) and Xe1 (associated with task tX), Ae3 (associated with task tA) and Ye2 (associated with task tY), and Ce1 (associated with task tC) and Ze1 (associated with task tZ) with single network components.

The following command is executed to incorporate single network task.

```
<khs@sunbird>java SNCT SingleEntryTaskData.txt Nt
```

The output LQN model is shown in Figure 5.3. Three calls are replaced by three network component tasks Nt0, Nt1, and Nt2.

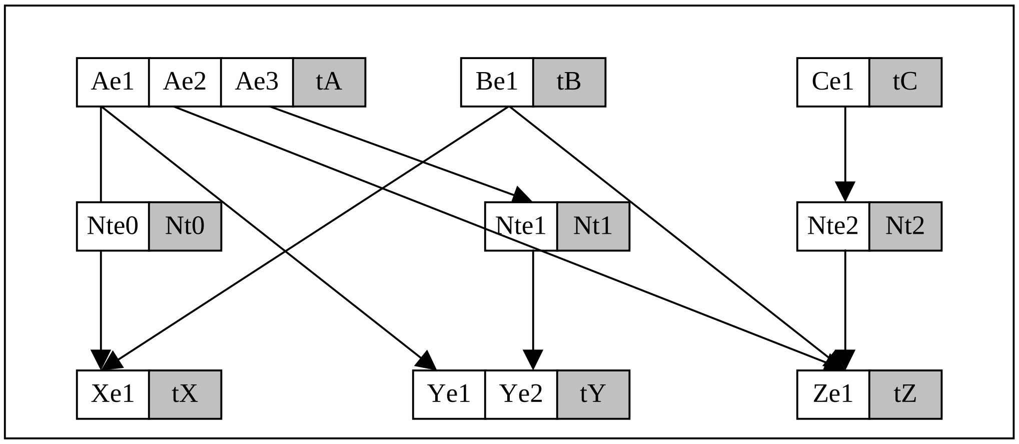


Figure 5.3: Output LQN Model

5.5 Network Subsystem (NSS) Filter

5.5.1 Limited Goals For The Design

There are some cases in which a single call is handled by multiple network components, which are the parts of a network subsystem. These multiple network components have associated network delay and execution over heads which might affect the over all performance of a distributed and communication software. Each component may have its own processor or sharing a processor with some other tasks. For example a network subsystem may consist of replicated tasks having a separate processor for each task e.g. a series of routers connected between a client and server and some other tasks with shared processors e.g. an ORB agent etc. So, it is required to design such a filter, which will replace a single call with a network subsystem.

In SNCT filter design, it was assumed that similar or dissimilar type of network components would be incorporated during a multiple calls replacement and each network component would replace a call while in NSS filter design a network subsystem replaces a single call with multiple replicated and shared tasks.

5.5.2 Design Of The Filter

The design of NSS filter is based on the design of network component filter shown in Figure 5.1. In addition to the three arguments described in Network Component filter design, NSS filter uses a fourth argument on command line. The first three arguments are the names of three separate files and the fourth argument is the name of the processor shared by some tasks of the network subsystem and the input LQN file. The first argument is an input LQN file, the second argument is the EntryTaskData file, and the third argument is network subsystem LQN file. The input LQN file and the EntryTaskData file are already described in section 5.3 while the network component LQN file is used to specify a network subsystems consisting of multiple network tasks. It is already mentioned in the previous section that a network subsystem may consist of some replicated tasks and some shared processor tasks. A shared processor task is distinguished from other tasks by using three letters 'KHS' at the end of task name in the network subsystem LQN file. If a processor name is specified in the fourth argument and the processor exists in the input LQN file then the shared processor task will share the processor, which is defined as the fourth argument on the command line, with other tasks of the input LQN file otherwise the NSS filter will generate an exception and halt. Thus the complete command line has the following syntax.

```
$java NSS model.lqn EntryTaskData.txt SubsystemModel.lqn SharedProcessor
```

In this filter, the type of the call, which will be replaced by network subsystem, is important. The call will or will not affect the calls inside the network subsystem. When a network subsystem "completion" is defined in a software system then this subsystem maintains the type of call between the sender and the receiver of the replacing call. For example, if a call, which is to be replaced by network subsystem, is synchronous then sender gets the reply from receiver. When the network subsystem replaces the call then sender gets the reply from the receiver not from any of the tasks of network subsystem. An example explains this effect in section 5.5.4.

A network subsystem may consist of one or multiple tasks. These tasks can be on a separate processor or on common processor depending on the design of network subsystem. The type of call between each pair of tasks may be similar or dissimilar. For example, a network subsystem consists of three tasks nt0, nt1 and nt2. It is assumed that the call type between each pair of tasks is asynchronous. In similar call type case, nt0 will be interacting with nt1 asynchronously and nt1 will be interacting with nt2 asynchronously. In dissimilar call type case, nt0 will be interacting with nt1 asynchronously and nt1 will be interacting with nt2 synchronously or vice versa.

5.5.3 NSS Filter Algorithm

This section provides the algorithm for the NSS filter for the replacement of synchronous, asynchronous and forwarding calls. The validity check for the set of pair of entry and the associated task (source and destination), call between source and destination for the EntryTaskData.txt file described for the SNCT filter are same for the NSS filter the only difference is the processor definition. In NSS filter, the shared processor is defined as the fourth argument instead of defining it in the EntryTaskData.txt file. Three letters "KHS" are appended with the name of the task in the network subsystem file which will share the processor with the other task of the input LQN model. If the fourth argument is not defined at the time of the execution, the NSS filter will generate an exception.

In NSS filter, the call to be replaced between sender and the receiver in the input LQN file important. Since the type of the call will affect the calls inside the network subsystem, which will replace this call therefore, each call replacement will be described separately.

1. Asynchronous Call Replacement

If the call, which is to be replaced, is an asynchronous call then there will be no affect on network subsystems calls. The NSS filter will simply plug-in the network subsystem in the input LQN file by introducing asynchronous call between the sender task and the first task of the network subsystem and between the last task of the network subsystem and the receiver task. The filter will generate an output LQN file.

2. Synchronous Call Replacement

If the call, which is to be replaced, is a synchronous call then the NSS filter will introduce a synchronous call between the sender task and the first task of the network subsystem and the call between the last task of the network subsystem and the receiver task will be converted according to type of the network subsystem because the synchronous call replacement affects the calls inside the network subsystem. The effect of the call can be described as under:

- If the call between each pair of the tasks of a network subsystem is also synchronous then there will be no affect of replacing call on network subsystem calls. The NSS filter will simply plug-in such a network subsystem in the input LQN model by introducing a synchronous call between last call of the network subsystem and the receiver task and generate an output LQN file.
- If the call between each pair of tasks of the network subsystem is asynchronous then these asynchronous calls will be replaced by forwarding calls. For such a network subsystem the NSS filter will convert all the asynchronous calls of the network subsystem into forwarding calls and introduce a forwarding call between the last task of the network subsystem and the receiver task. The filter maintains type of interaction between the sender and the receiver and the receiver gets the reply from the receiver.
- If there is a forwarding call interaction in the network subsystem then the NSS filter will introduce a synchronous call between the last task of the network subsystem and the receiver task. After the insertion of the network subsystem the sequence of interactions between the sender task and the receiver task can be described as follows. The sender will send the message and wait for the reply to the first task of the network subsystem, which will send this message to the second task and wait for the reply. This message will be forwarded to the last task of the network subsystem, which will send this message to the receiver and wait for the reply. The receiver will reply to the last task of the network subsystem. The last task will reply to the first task of the network subsystem, which will eventually reply to the sender.

3. Forwarding Call Replacement

If the call, which is to be replaced, is a forwarding call then the NSS filter will introduce a forwarding call between the sender and the first task of the network subsystem and a forwarding call between the last task of the network subsystem and the receiver. The effect of the call can be described as under:

- If the call between each pair of tasks of a network subsystem is also forwarding then there will be no affect of replacing call on network subsystem calls. The NSS filter will simply plug-in network subsystem in the input LQN file and generate output LQN file.
- If the call between each pair of tasks of a network subsystem is asynchronous then the NSS filter will convert all the asynchronous calls to the forwarding calls and generate an output LQN file.
- The filter will generate an exception if there is a synchronous call in network subsystem.
- The filter will generate an exception if a forwarding call is followed by an asynchronous call in network subsystem.

5.5.4 Example (See Also Appendix Fi)

To understand the effect of a call, which will be replaced by a network subsystem, consider a simple Input LQN Model shown in Figure 5.2. It is required to replace the call between the entries Ae3 (tA) and Ye2 (tY). The Input LQN File is included in Appendix G1 and the EntryTaskData.txt file in Appendix G2.

A network subsystem consists of three single entry replicated tasks and a shared processor task, which is distinguished by, letters ”KHS”. The network subsystem file is included in Appendix G3.

Synchronous Call Replacement

Since the call, which is to be replaced, is a synchronous call therefore the interactions between the tasks inside the network subsystem will be affected. Three cases are described as under:

Case 1: The call between each pair of tasks of a network subsystem is synchronous therefore will be no effect of replacing call on network subsystem calls. The network subsystem is included in Case S1 of Appendix F2. The output LQN model is included in Case S1 of Appendix F2. When the fourth argument is defined as processor p3 then the filter replaces the shared task processor np2 with p3. The output LQN file is included in Appendix G4.

Case 2: The call between each pair of tasks of network subsystem is asynchronous and these asynchronous calls will be replaced by forwarding calls. The network subsystem is included in Case S2 of Appendix F2. The source entry Ae3 will interact with the entry of first task of network subsystem synchronously and all other entries in network subsystem will forward this call to the destination entry and the destination entry Ye2 will reply to the source entry Ae3. The output LQN model is included in Case S2 of Appendix F2.

Case 3: A forwarding call network subsystem is included in Case S3 of Appendix F2. The output LQN model is included in Case S3 of Appendix F2.

Forwarding Call Replacement

To understand the effect of a forwarding call, which will be replaced by a network subsystem, consider a simple input LQN model included in Appendix F3. The possible forwarding call replacement cases are described as under:

Case1: All the tasks are interacting asynchronously in the network subsystem and all the calls will be converted to forwarding calls. The network subsystem and output LQN model are included in Case F1 of Appendix F4.

Case2: All the tasks are forwarding the request and there will be no effect. The network subsystem and output LQN model are included in Case F2 of Appendix F4.

5.6 Shared Resource Server Task (SRST) Filter

5.6.1 Motivation For The Design

There are some cases in which a many clients are interacting with one server. The server may have the capability to serve unlimited number of client requests. Server handles each client request independently with a separate thread. For example, this might be a web server or a tuple space server. Shared Resource Server Task (SRST) filter introduces such a “completion” in LQN model. In this type of “completion”, a call is replaced by an entry of a shared resource (threaded) server LQN task. If there are number of calls and it is required to replace these calls with a threaded server task then a single entry of this server will replace a single call.

5.6.2 Design Of The Filter

The design of SRST filter is based on the design on network component filter shown in Figure 5.1. It accepts three arguments from user on command line. The first two arguments are files names, and third argument is a server task name, which provides uniqueness to server task name. The details of input LQN file and EntryTaskData.txt file are discussed in section 5.2. Thus the complete command line has the following syntax.

```
$java SRST model.lqn EntryTaskData.txt ServerTaskName
```

5.6.3 SRST Filter Algorithm

This section provides the algorithm for the SRST filter for the replacement of synchronous, asynchronous and forwarding calls. The type of the call is not important because each call will be replaced by the single entry of the share resource server task. Since, this is a server task therefore it is assumed that it will have a separate processor.

The execution steps of the SRST filter are described as under:

1. The step1 to 4 described in section 5.4.3 for the SNCT filter are same for the SRST filter as well.
2. After the validity of the input LQN file, EntryTaskData.txt file and task name, the relation between entry and the associated task (source and destination), and presence of the call the filter will incorporate a multiple entries shared resource server. Each entry of the task will replace a call. The SRST filter will insert a separate processor with the task in the input LQN file and will generate output LQN file.

5.6.4 Example

Tuple space server task can be incorporated as a shared resource task. This network component is used when tuple space communication is used among different tasks. It is assumed that there is one centralized tuple space server, which is coordinating communication among between many tasks.

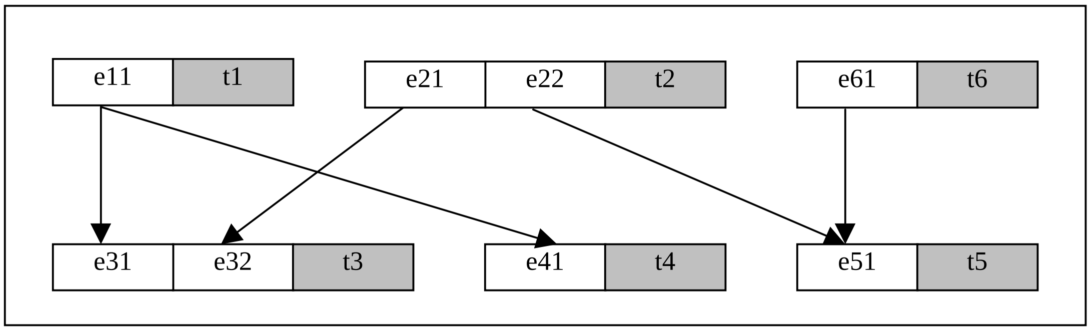


Figure 5.4: Input LQN Model

Consider an LQN model shown in Figure 5.4 as input LQN model. It is required to introduce tuple space server task in the model. Each call will be replaced by each entry of tuple space server task. The output model is shown in Figure 5.5.

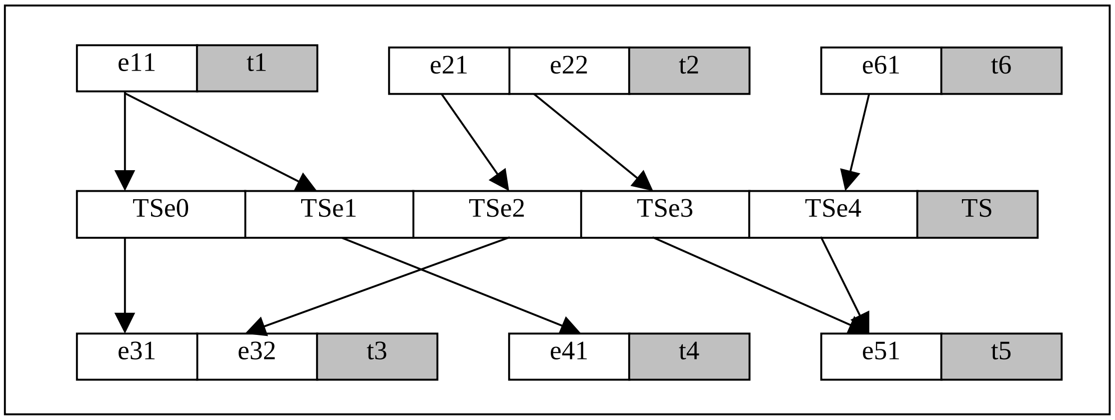


Figure 5.6: Output LQN Model

5.6.5 Design Limitations

- If a user would like to replace more LQN calls with a server task which is already incorporated in a LQN model. New entries cannot be incorporated in server task once the server task is created. This can be done by hand but automation does not support it.

- Multiple server tasks cannot be incorporated at a time for distributed server systems. In order introduce the idea of distributed server tasks; the filter program will be executed as many times as the number of server tasks are required.

5.7 Multicast Network Task (MCNT) Filter

5.7.1 Motivation For The Design

There are some cases in which the information is sent from one or more senders to a set of other receivers e.g. a multicast network. Multicast Network Task (MCNT) filter introduces such a "completion" in LQN model. In this type of "completion", a call is replaced by single entry LQN task. If there are number of calls and it is required to replace these calls with a single entry LQN task.

5.7.2 Design Of The Filter

The design of MCNT filter is based on the design on network component filter shown in Figure 5.1. It accepts three arguments from user on command line. The first two arguments are files names, and third argument is single entry LQN task name, which provides uniqueness to task name with the other tasks of input LQN model. The details of input LQN file and EntryTaskData.txt file are discussed in section 5.2. Thus the complete command line has the following syntax.

```
$java MNCT model.lqn EntryTaskData.txt MulticastNetowrkTaskName
```

5.7.3 MCNT Filter Algorithm

This section provides the algorithm for the MCNT filter for the replacement of synchronous, asynchronous and forwarding calls. Since multiple calls will be replaced by the single entry of the task therefore the type of the call not important. It is assumed that this task will have a separate processor.

The execution steps of the MCNT filter are described as under:

1. The step1 to 4 of the SNCT filter described in section 5.4.3 are same for the MCNT filter.
2. After checking the validity of the input LQN file, the EntryTaskData.txt file and the task name, the relation between entry and the associated task (source and

destination), and presence of the call, the filter will incorporate a single entry task. The MCNT filter will insert a separate processor with the task in the input LQN file and will generate output LQN file.

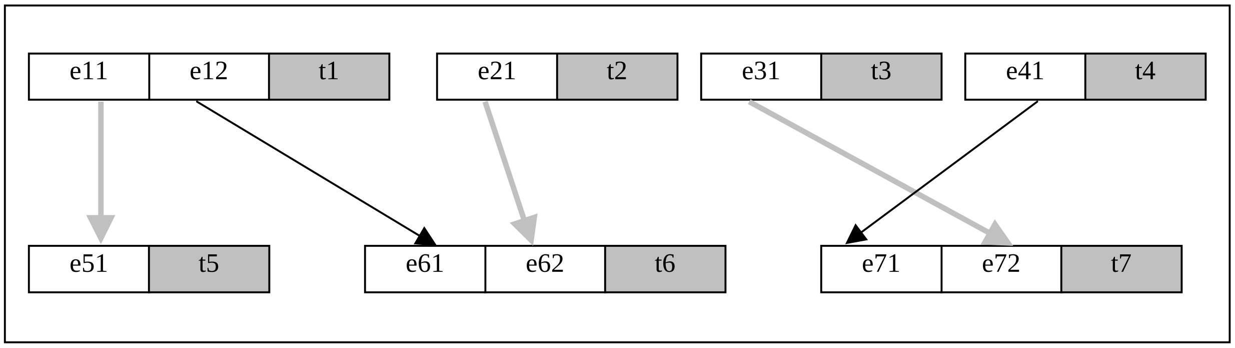


Figure 5.7: Output LQN Model

5.7.4 Example

Consider an LQN model shown in Figure 5.7 as the input LQN model. It is required to introduce a multicast network task in the model. The calls, which will be replaced by single entry multicast network (MCast) task, are shown with gray colors. The output model is shown in Figure 5.8.

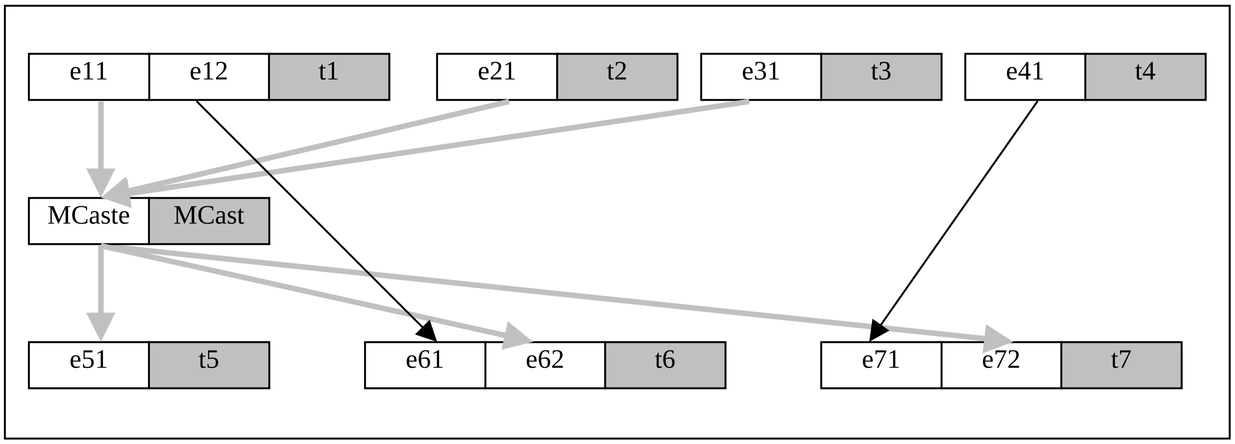


Figure 5.8: Output LQN Model

5.8 Future Work

All the filter designs discussed in this chapter support call replacement for a call between source and destination entries, not the call between a source activity and a destination entry. So, the designs can be extended for call replacement between source activity and destination entry.

5.9 Summarry

This chapter demonstrated the idea of an automated tool for the “completion”, LQN Component Filter, which filter insert the “completion” at LQN level only. The LQN Component Filter has four variations and the design and implementation of each filter is described with a example. These are as follows:

1. Single Network Component Task (SNCT) filter,
2. Network Subsystem (NSS) filter,
3. Share Resource Server Task (SRST) filter, and
4. Multicast Network Task (MCNT) filter.

6.0 Tuple Space Measurements and Modeling

6.1 Introduction

This chapter describes the measurements done on a particular vendor specific tuple space called J-Spaces. J-Spaces Technologies Ltd. releases J-Spaces as The J-Spaces Platform 1.0. The J-Spaces Platform 1.0 is the first commercial implementation of the JavaSpaces specification. JavaSpaces is a powerful Jini service specification. In order to understand J-Spaces measurements environment, brief descriptions of Jini network technology, JavaSpaces and The J-Spaces Platform 1.0 will be discussed. Measurements results are analyzed using *PerfAnal* [Mayers00].

6.2 Jini Network Technology

Jini network technology provides simple mechanisms, which enable devices to plug together to form a community. Each device provides services that other devices in the community may use. These devices provide their own interfaces, which ensures reliability and compatibility. Sun Microsystems introduces this network technology. The details of Jini Specification are described in [Oaks00, Arnold00, Edwards00, and Arnold99a].

Jini technology uses a lookup service with which devices and services are registered. When a device plugs in, it goes through an add-in protocol, called discovery and join-in. The device first locates the lookup service (discovery) and then uploads an object that implements all of its services' interfaces (join).

6.3 JavaSpaces

JavaSpaces is an integral part of Jini1.1 Starter Kit of SUN Microsystems Inc. [Waldo98]. JavaSpaces is a powerful Jini service specification that provides a simple, yet powerful infrastructure for building distributed applications. The JavaSpaces specification defines a reliable distributed repository for objects, along with support for distributed transactions, events and leasing. Applications are viewed as a group of processes in JavaSpaces programming model, cooperating via the flow of objects into and out of "spaces” [Arnold99b].

JavaSpaces is based on the concept of tuple spaces first described in 1982 in the Linda programming language and system, originally propounded by Dr. David Gelerntner at Yale University [Roberts92, Foster95, Walkner95, Gelerntner97, Dasgupta98 and Matloff99]. The public-domain Linda system is a coordination language for expressing parallel processing algorithms without reference to any specific computer or network architecture and provides inter-process coordination via virtual shared memories, or tuple-spaces, that can be accessed associatively.

The tuple-space model is especially useful for concurrent algorithms. Although JavaSpaces technology is strongly influenced by the Linda system, it differs from it in several ways, like Java's richer typing, object orientation, subtype matching and transactional support spanning multiple spaces, leasing and events.

6.3.1 JavaSpaces API

The JavaSpaces API is very simple and elegant, and it provides software developers with a simple and effective tool to solve coordination problems in distributed systems, especially in domains like parallel processing and distributed persistence. The developer designs the solution as a flow of objects rather than a traditional request/reply message based scenario. Combined with the fact that a Java Space is a Jini service, thus inheriting the dynamic nature of Jini, JavaSpaces is a good model for programming highly dynamic distributed applications.

Each object, which implements the interface net.jini.core.entry.Entry, corresponds to a tuple. The public instance variables of these objects are the components of the tuple. The JavaSpaces API consists of 4 main method types discussed in [Waldo98 and Arnold99b]:

- Write() : writes an entry to a space.
- Read() : reads an entry from a space.
- Take() : reads an entry and deletes it from a space.
- Notify() : registers interest in entries arriving at a space.

JavaSpaces enables full use of transactions, leveraging the default semantic of the Jini Distributed Transactions model. This enables developers to build transactional-secure distributed applications using JavaSpaces as a coordination mechanism. Since Jini transactions can span more than one Jini Service, and a JavaSpace instance is a Jini Service, a transaction may span more than one space.

The API itself provides non-blocking versions, where a read() or take() operation may take a maximum timeout to wait before returning to the caller. This is very important for applications that cannot permit themselves to block for long times or in the case that the space itself is in some kind of a deadlock.

JavaSpaces also makes extensive use of Jini leases, as it mandates that entries in the space be leased (and thus expire at a certain time unless renewed by a client). This prevents out-of-date entries, and saves the need for manual cleanup administration work.

6.4 The J-Spaces platform 1.0

The J-Spaces Platform [JSpaces] was designed initially for enterprise-scale mission-critical applications that require the extensive use of the JavaSpaces API.

A J-Spaces platform is an intelligent connectivity network service. It consists of a server or a set of servers, which are located somewhere in the network, and allows applications to share and exchange information in a transparent and spontaneous manner. Upon demand, the space server allocates a space (distributed shared memory), into which the user can start sending information objects. The information objects in the space can be shared with other applications. Unlike other solutions, the space doesn't need to know anything about the information it stores, or about the identity and location of the sender and receiver. The application, on the other hand, can use the space immediately without any need for specific integration or modifications.

6.4.1 J-Space server

J-Spaces server is the actual entity (JVM) that has a J-Spaces container instance among other Jini services. The J-Spaces server provides the ability to control whether or not to launch in-memory Jini services. By default all the services, including the HTTP daemon, RMID, reggie, Jini Lookup Service (LUS) and mahalo (Jini Transaction Service) are launched.

The LUS is an active-able service. A client will first access the activation service (RMID) and the RMID will provide them with the reference for the actual LUS. This is done only once in the life cycle of client/service communication. After the client gets a reference to the LUS from the RMID it access the LUS directly not through the RMID.

J-Spaces supports dynamic code loading of its proxy classes. This means that when a client gets a reference to the space its application needs to know nothing about its underlining implementation. The proxy loads dynamically all the relevant classes it requires from the HTTP server and this is the reason why HTTPD is required in J-Spaces Platform. A proxy is not a server it is an object, which contains the client implementation of the space and can be obtained via the LUS.

6.5 Measurements Using *PerfAnal* Tool

The measurements results are analyzed using a tool called *PerfAnal*. Measurements are done using Java Development Kit (JDK) 1.3.0_01 version on Windows NT 4.0 platform.

PerfAnal is selected for the following reasons:

- It is easy to use and provides reasonable results
- The measurement tool provided by JDK is called *hprof*. *hprof* is actually an agent, which can be helpful to provide information about CPU time, heap usage etc. by using JVM profiler. It is difficult to get and analyze the results from *hprof*. *PerfAnal* reads the results from a file coming from *hprof*, it analyzes data and provides a graphical display for CPU time of threads, methods and lines on behalf of *caller* and *callee*.
- *PerfAnal* is used for analysis when process is terminated that’s why it does not occupy too much memory when process is executing.

This tool has some drawbacks. These are as under:

- It inherits drawback of *hprof.*
- It does not provide elapsed time (wall clock time). The time it takes for a task to finish its job.
- Concurrent measurements are not possible. *PerfAnal* is used when process is terminated that’s why it is not possible to monitor a process during its execution. It is not possible to get the snapshot of some interesting point during process execution.

- CPU time is measured in tick, so it is required to calibrate the tick for of machine and JDK.

6.6 A Brief Description of Master/Workers Example

This example is used to do measurements on J-Spaces. Asaf Kariv, the Vice President R&D of J-Spaces Technologies Limited, developed this example.

The example is based on a Master and Workers processes. Master process writes an integer on the space. Workers read this integer from the space. Each worker finds the divisor of the integer from a specified range of integers allocated by Master and writes the result back to the space. Master takes this result from the space and determines whether the result is a prime number.

The sequence of operation in terms of J-Spaces can be described as follows: Workers wait for the data from the Master. The Master writes the data on the space and the space notifies the Workers proxy about that event and handles the Workers proxy actual data. This causes Workers to return from their blocking read operations and continue to write its result and the same goes to the Master. The space does not block for this amount of time, only the Master and Workers proxies. The space woke up the proxies via a callback that it triggers for those clients.

This example consists of five Java classes. The descriptions of three core classes are described in separate sub sections.

Main.java class

Main.java class sets RMI Security Manager. It gets references to the space and transaction services. It calls Master.java class and displays the result test whether the result is a prime number.

Master.java class

This class is responsible for splitting the task of checking a prime candidate into sub tasks, gathering the partial results and returns the final result.

The following steps are involved to determine whether a given integer is prime. Master.java class

- creates a new transaction

- breaks task into several tasks and write them to space in one transaction
- commits transaction
- creates and starts worker threads wait for any result. If the result indicates that is not a prime number, return false. If all results indicate a prime return true. It creates a new transaction
- finds divisor whether the candidate is prime number

Master.java class uses two classes CheckIfPrimeTask.java and CheckIfPrimeResult.java.

CheckIfPrimeTask object encapsulates a task entry in space. It writes the result as a CheckIfPrimeResult object. CheckIfPrimeResult object encapsulates the result of a CheckIfPrimeTask entry.

Worker.java class

A worker is a thread that is responsible for taking CheckIfPrimeTask tasks from a space, performing these tasks and put back a CheckIfPrimeResult objects. Worker.java class

- creates a new transaction
- takes a task from space
- checks result
- commits transaction

6.7 Scenario for the Measurement

The measurements are done on Windows NT 4.0 Workstation operating system using Sun Utlra 5 machine through SunPC card. The scenario for the measurements is simple and consists of the following steps:

- The J-Spaces Platform Installation: The installation of the J-Spaces Platform set up all the services, including the HTTP daemon, RMID, reggie, Jini Lookup Service (LUS) and mahalo (Jini Transaction Service) because they are embedded in the J-Spaces Platform.
- The Installation Verification: Upon completion of the installation it is necessary to run the J-Spaces Server. The J-Spaces Server is installed by default in “C:\J-

SPACES>". The server is run from Windows NT console as well from "Start" menu. The utilities are configured to work on the default space named "JavaSpaces" which is created upon installation.

- Testing the J-Spaces Server: Before running the J-Spaces Server it is recommended to check the parameters in the setenv.bat file. To run the server the JSpacesServer.bat file is executed and the following result is obtained on the console:

```
C:\J-SPACES\bin>setenv


C:\J-SPACES\bin>JSpacesServer
Verifing license key...OK
J-Space server starting...
C:/J-Spaces\lib\jaxp.jar
C:/J-Spaces\lib\parser.jar
C:/J-Spaces\lib\JSpaces.jar
C:/J-Spaces\lib\JSpaces-dl.jar
C:/J-Spaces\lib\jini-core.jar
C:/J-Spaces\lib\jini-ext.jar
C:/J-Spaces\lib\jini-core.jar
C:/J-Spaces\lib\reggie-dl.jar
C:/J-Spaces\lib\mahalo-dl.jar
C:/J-Spaces\lib\jini-examples-dl.jar
C:/J-Spaces\lib\space-examples-dl.jar
Web server is up...
RMID is up and running...
/reggie-dl.jar requested from sunbird-pc.sce.carleton.ca:3073
Reggie is up...
/mahalo-dl.jar requested from sunbird-pc.sce.carleton.ca:3079
Mahalo is up...
...
```

- To verify a successful installation run the ping.bat utility and the following result is obtained on the console:

```
C:\J-SPACES\bin>ping sunbird-pc JavaSpaces -r localhost
Looking for sunbird-pc container...
Started to READ from <JavaSpaces> space with
TIMEOUT: FOREVER
BUFFERSIZE: 100
ITERATIONS: 5
AVERAGE TIME = 118 milliseconds
```

- Measurement with the Master/Worker example: The run.bat, which is a script file, contains the following command and options to record traces of the example for J-Spaces measurements. The run.bat file contains the following:

```
@call ../../bin/setenv.bat
@java -Xrunhprof:cpu=samples,depth=12,thread=y,file=f150.txt
-Djava.security.policy=%POLICY%
-classpath../../lib/jini-core.jar;../../lib/jini-ext.jar;
classes;../utilities/classes primes.Main localhost 150
```

  where,

  1. -Xrunhprof is an option used to record traces of a Java program and file=f150.txt is an ASCII file where the traces are recorded.
  2. classes;../utilities/classes primes.Main localhost 150 is an option where the name of a java class file, of which measurements are obtained, is specified.

- After recoding the traces in f150.txt file the measurements results are analyzed using *PerfAnal* program, which is written in Java and is a separate program, which is not included in J-Spaces Platform. The following is a command to execute the *PerfAnal* program.

```
C: \PerfAnal>java -cp pa.jar com.macmillan.nmeyers.PerfAnal f150.txt
```

6.8 Master/Workers Example Performance Model

The performance model of Master/Workers example is essentially a queuing model expressed in layered queuing, a driver task called "RefTask1" stimulating the execution of operations of software entities. The performance model is shown in Figure 6.1.

The configuration that was modeled and analyzed for the measurements was one Master and 12 Workers. Master registers service to Lookup Service (LUS task) using RMID Server (RMID task) and Transaction Manager (TM task) and places data on the space (Jspaces task) using HTTP Daemon (HTTPD task).

Operation Name	CPU Demand (ms) per operation
Read	3.92
Write	3.92
Take	3.92

Table 3: Measured CPU Demands for Operations

Workers read the integer from the space using same sequence of operations, find the divisors of the integer and write the results on the spaces. Masters takes the results from the space. The LQN file of J-Spaces model is included in Appendix B.

The CPU demands measured and calculated for each entry of operation and component (server) in milliseconds is shown in Table 3 and 4 respectively.

The measured CPU demands show that it would take 12.25 ms for Master or Worker to do a read, write or take operation in J-Spaces.

Component Name	CPU Demand (ms) per operation
RMID	131.5
TSM	8.33
LUS	31.83
HTTPD	56.42

Table 4: Measured CPU Demands for Components

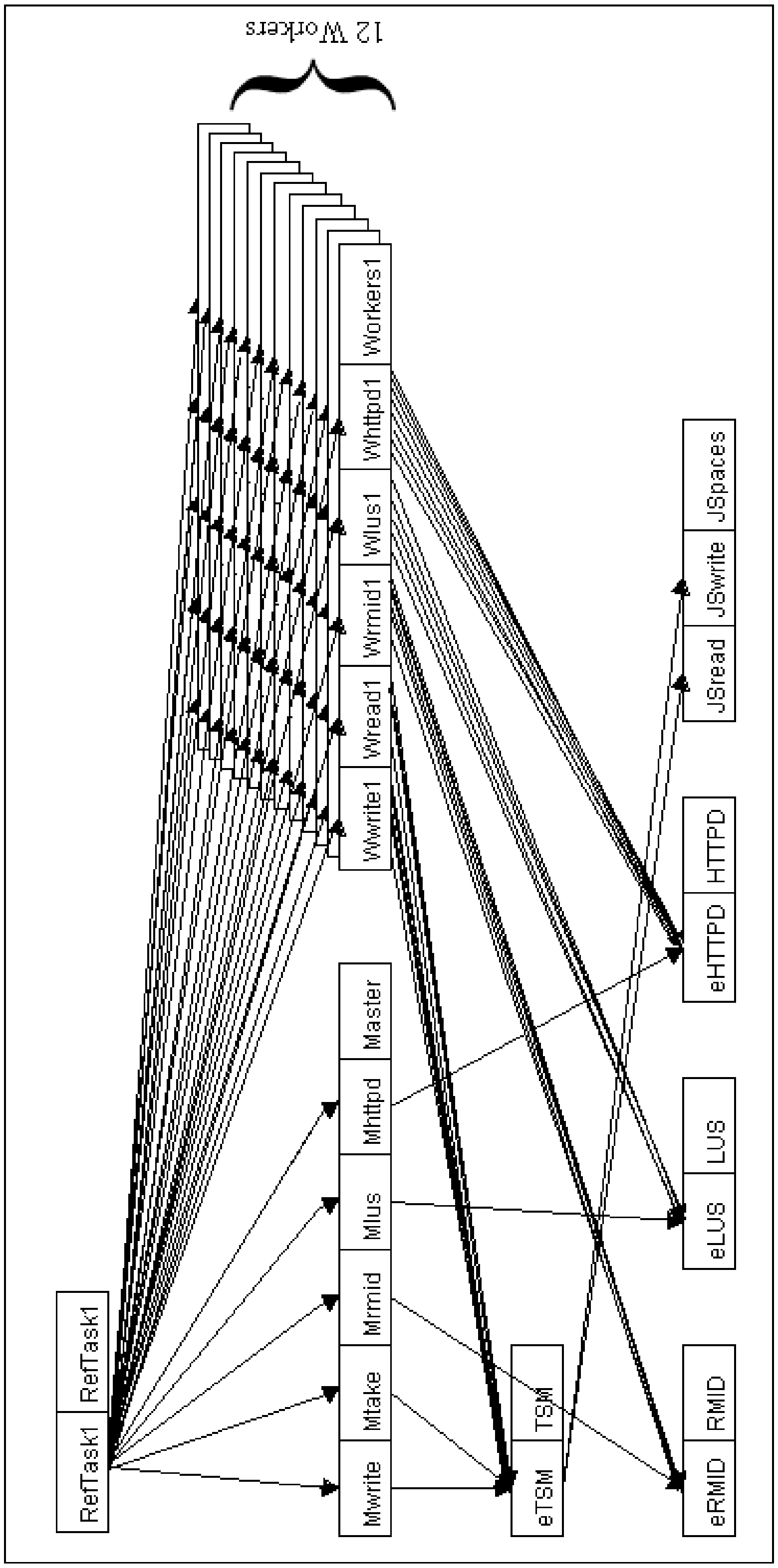


Figure 6.1: LQN Model of J-Spaces

6.9 Model Results

The simulation model gave the throughput, response time and processor utilization of Workers, calculated for a range of Workers, for different numbers RefTask1 users.

Figure 6.2 shows the curves of the response time (msec) for different number of RefTask1 users for a range of Workers. The range of Reftask1 users do not affect the repose time of the system as much as the different number of the workers. The response time of the system increases with number of workers. The same data is shown in numerical form in Table 5.

Figure 6.3 shows that as the number of Workers increases the performance of the system decreases. The host processor utilization was 97.17% for 12 workers when throughput was around 0.00143 response cycles for 1 RefTask1 users. The bottleneck of the system is RMID Server, when the utilization of RMID server was 52.55% the over all processor utilization of system was 97.8%.

The throughput and host processor utilization are shown on graphs in Figure 6.3 and 6.4. The same data is shown in numerical form in Table 6 and 7.

The results are obtained with 95% confidence level (error ranges from and 1% to 10%).

6.10 Summary

This chapter describes the measurements and modeling of a commercially available tuple space, J-Spaces, with an example. It describes the following:

1. Jini Network Technology overview,
2. JavaSpaces overview,
3. J-Spaces platform 1.0 necessary details for the measurement,
4. Measurements using PerfAnal tool, and
5. Brief description of Master/Workers example, scenario for the measurement, performance model for the Master/Workers example and the model results.

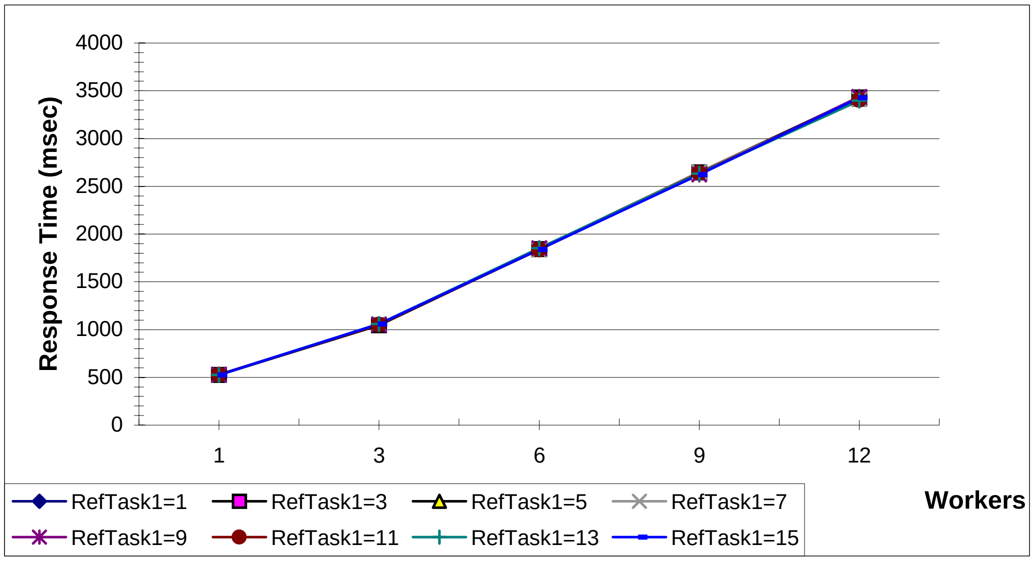


Figure 6.2: Response Time (msec) to the Number of Workers

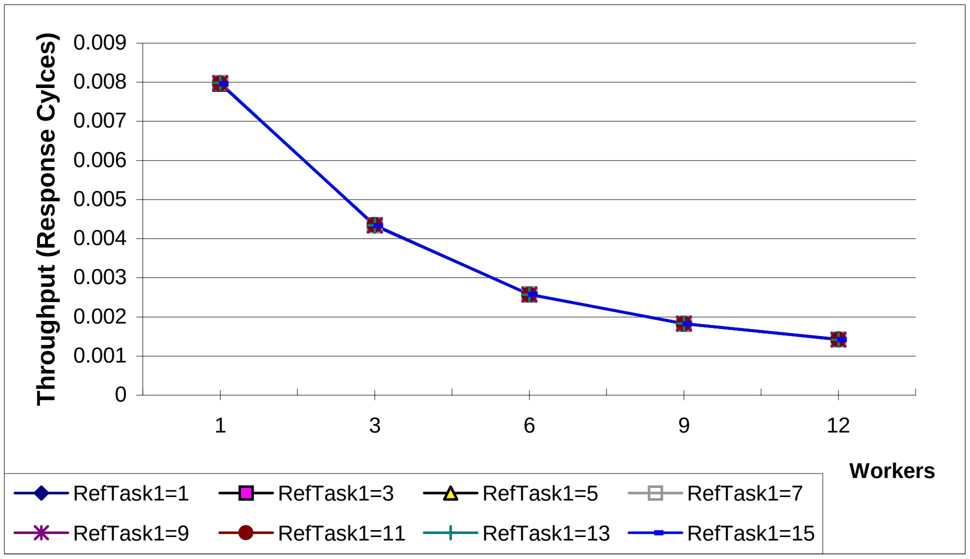


Figure 6.3: Throughput (Response Cycles) to the Number of Workers

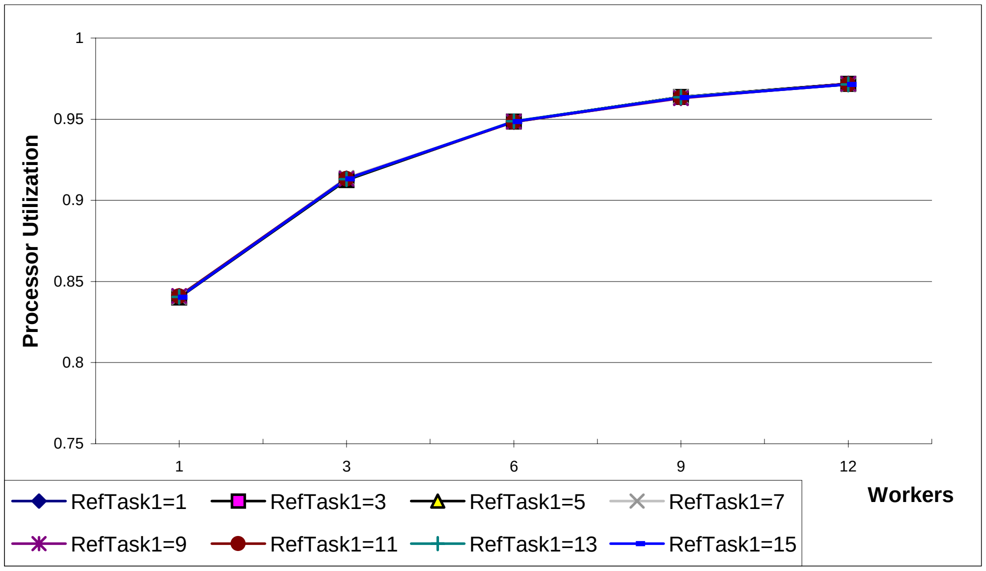


Figure 6.4: Host Processor Utilization to the Number of Workers

Workers	RefTask1							
	1	3	5	7	9	11	13	15
1	526.058	526.243	526.8	527.619	528.018	527.478	526.603	526.242
3	1055.54	1044.8	1052.83	1057.42	1052.71	1051.55	1057.09	1057.04
6	1840.01	1842.61	1845.41	1837.15	1852.35	1840.62	1854.15	1839.57
9	2632.12	2645.87	2647.81	2648.9	2624.1	2638.84	2631.71	2622.91
12	3433	3431.41	3417.62	3410.1	3438.37	3404.59	3398.54	3425.99

Table 5: Response Time (msec) to the No. of Workers

Workers	RefTask1							
	1	3	5	7	9	11	13	15
1	0.00798	0.00796	0.00797	0.00799	0.00797	0.00797	0.00798	0.00796
3	0.00433	0.00434	0.00434	0.00433	0.00434	0.00435	0.00434	0.00433
6	0.00257	0.00258	0.00258	0.00258	0.00257	0.00257	0.00257	0.00257
9	0.00183	0.00183	0.00183	0.00183	0.00183	0.00183	0.00183	0.00183
12	0.00143	0.00142	0.00142	0.00142	0.00142	0.00142	0.00142	0.00142

Table 6: Throughput (Response Cycles) to the No. of Workers

Workers	RefTask1							
	1	3	5	7	9	11	13	15
1	0.84008	0.84009	0.84066	0.84057	0.84113	0.84082	0.84050	0.84033
3	0.91345	0.91256	0.91318	0.91386	0.91315	0.91329	0.91308	0.91323
6	0.94844	0.94854	0.94845	0.94841	0.94858	0.94842	0.94866	0.94849
9	0.96324	0.96376	0.96365	0.96371	0.96295	0.96332	0.96349	0.96323
12	0.97179	0.97178	0.97190	0.97137	0.97173	0.97144	0.97140	0.97146

Table 7: Processor Utilization to the No. of Workers

7.0 Case Studies

7.1 Introduction

This chapter will demonstrate performance budgeting approach by presenting two case studies; one describing Distributed Hand-off Protocol UCM design and one describing Plain Old Telephone System (POTS) UCM design.

Two case studies are described to demonstrate different aspects of the usability of the approach. Distributed Hand-off Protocol UCM design case study involves all the necessary steps mentioned in budget analysis road map, the usability of LQN Component filters and tuple spaces. Plain Old Telephone System (POTS) UCM design case study involves only the budgeting of the CPU demands for the system. For POTS UCM design, it is assumed that the available UCM is a “complete” UCM and LQN model is extracted from UCM2LQN tool mentioned in Chapter 2.

7.2 Case Study A - Distributed Hand-Off Protocol UCM Design

7.2.1 Overview

This case study describes a UCM design of a distributed hand-off protocol, based on a tutorial example by Gunter Mussbacher at Mitel Networks, which illustrates a particular style of drawing UCMs, to teach designers the UCM notation [Mussbacher01]. The original UCM example has been simplified and used as Designer UCM for the case study.

7.2.2 Step1: Designer UCM

- Figure 7.1 shows a Designer UCM for a distributed hand-off protocol. The protocol described in this specification is used to coordinate the execution of some arbitrary duty by two tasks, Process_A and Process_B, in a distributed environment.

In Figure 7.1, the scenario is shown in three stages:

- Party_A initiates the execution of the duty, which is forwarded to Process_A via Device_A and Main_Controller, with a confirmation to Party_A.

- Later, Party_A initiates a request to hand off the duty to Process_B, which goes via Device_A and the Main_Controller to Process_A, and then to Process_B. Process_B interrogates Party_B. Party_B forwards a confirmation via Device_B and Main_Controller to Process_B, which then begins to execute the duty.
- While doing so it sends a synchronous storage request to Process_Backup, which carries out the request and replies back to Process_B.

Responsibilities in the UCM are interpreted for performance analysis as operations.

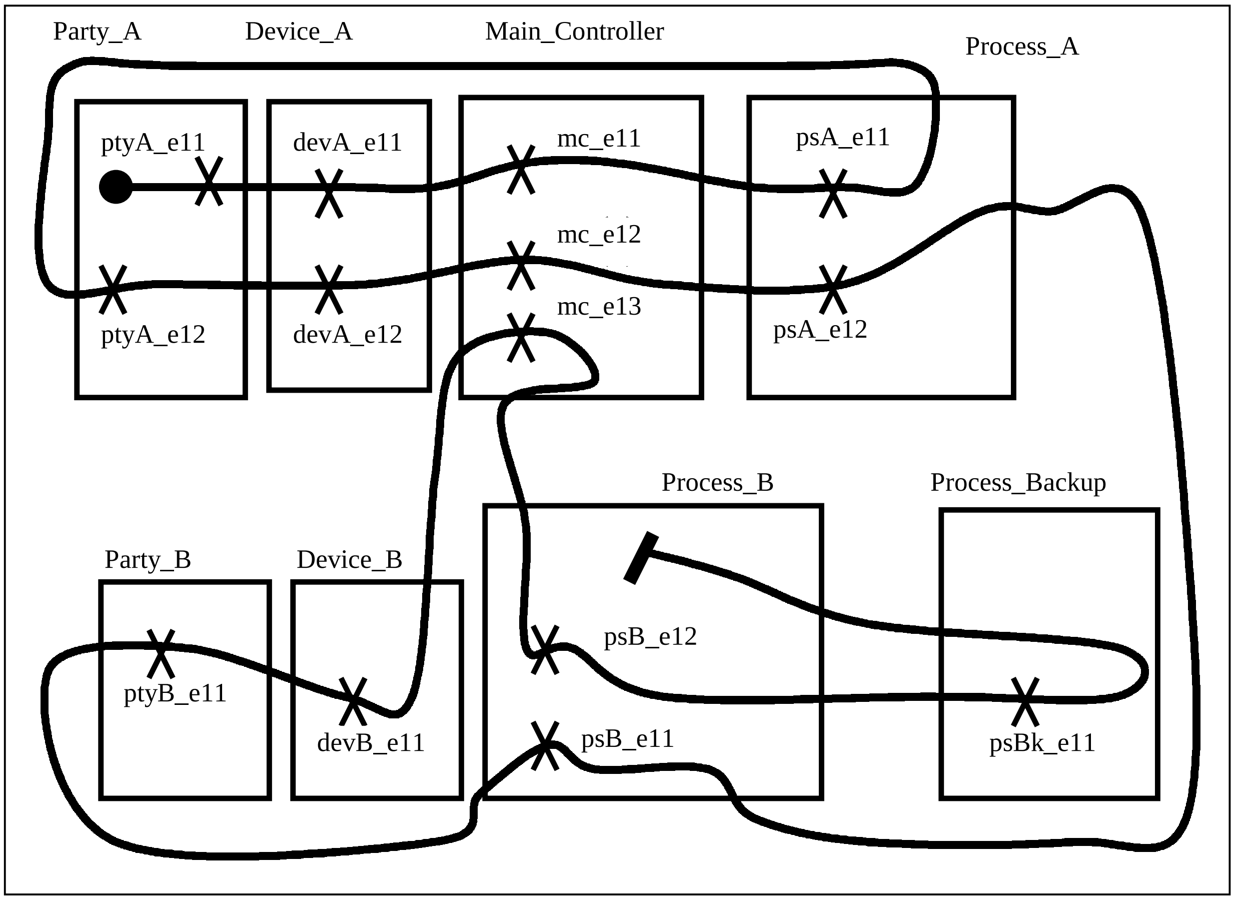


Figure 7.1: Distributed Hand-Off Protocol UCM (Designer UCM)

7.2.3 Step 2: Budgets

Since, this is an artificial example, the execution demands of the responsibilities have been arbitrarily chosen as unity, also the sensitivity of system response time and throughput are considered as targets to budget CPU time demand of each responsibility.

7.2.4 Step 3: UCM Completion and Infrastructure Components

The Designer UCM shown in Figure 7.1 does not describe the communication infrastructure between different communicating processes. Communication infrastructure might consist of protocol stacks, middleware, etc. Here, it was decided to use a tuple space communication mechanism, which is already described in Chapter 2. Other communications such as protocol stacks are needed but will not be shown here, for simplicity of explanation. The tuple space communications is provided by a server task, which handles and stores messages.

In this example, it is assumed that there are anonymous communications between Process_A and Process_B, and Process_B and Process_Backup. Tuple space provides anonymous communication feature among different communicating processes, as described in Chapter 2. Figure 7.2 shows a distributed tuple space server with two server processes. The process Tuple_Space_A provides communications between Process_A and Process_B, Tuple_Space_B process provides communication between Process_B and Process_Backup. All other processes are using address based socket communications.

The scenario illustrates that Process_A performs an “out” operation and places data in Tuple_Space_A, and Process_B performs an “in” operation, which removes it. Process_B and Party_B forward this request to Device_B. Device_B replies back to Main_Controller, which sends asynchronous message to Process_B. Process_B performs “out” operation and places tuple in Tuple_Space_B. Process_Backup removes tuple from Tuple_Space_B by performing “in” operation, it performs a backup operation and replies back to Process_B through Tuple_Space_B. The number shown in small gray box along UCM path provides sequence of operation for UCM scenario.

7.2.5 Step 4: LQN Model of Distributed Hand-off Protocol

The LQN model of Distributed Hand-off Protocol Complete UCM was generated by the LQN2UCM converter. This converter maps every task of UCM to an LQN task, every crossing to an entry, and every responsibility to an LQN activity. Figure 7.3 shows the model, with many activities suppressed. In Figure 7.3, every rectangle represents a task, with a gray color box showing the task name. White boxes besides this are the entries of the task. Boxes shown in a second row below the task are activities. The entry, which contains these activities, is just above the first activity from the left. The messaging arrows with a filled head represent synchronous service calls while the other arrows represent asynchronous service calls. The dashed line arrows with a filled head represent forwarded service calls.

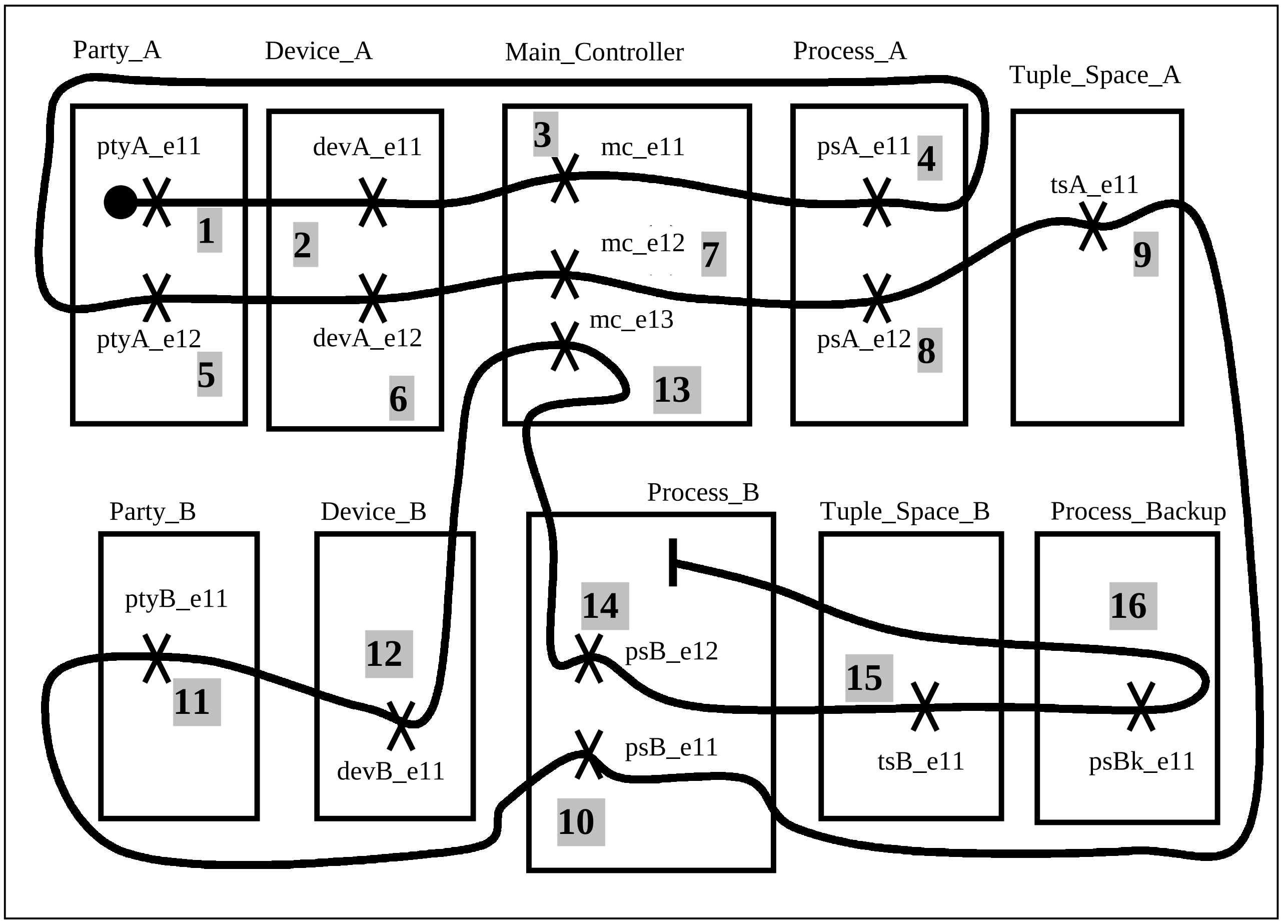


Figure 7.2: "Completion' of Figure 2 with Tuple Space Communication

The Figure 7.3 includes four new tasks, Party_ A_User, ts_CriticalSection, Cs_opA and Cs_opB, which have been introduced as additional 'completions" which will be described in next step.

Figure 7.3 shows that the scenario begins with:

1. the Party_A_User task initiates the scenario by making a synchronous request through entry ptyA_User to entry ptyA_tae of the Party_A task.
2. the activity PtyA_e11 handles this request and makes a synchronous request to entry dev_e11 of the Device_A task.
3. this request is forwarded by mc_e11 entry of Main_Controller to psA_e11 entry of Process_A, which replies back to ptyA_tae entry. Eventually, Party_A task replies back to Party_A_User task.

The scenario continues as Party_A task makes an asynchronous request to Device_A through ptyA_e12 activity. devA_e12 makes an asynchronous request to mc_tae entry of Main_Controller. This request is handled by mc_e12 activity of mc_tae entry. It makes a synchronous request to psA_e12 entry of Process_A task. Tuple_Space_A, Process_B, and Party_B forward this request to Device_B, which replies back to Main_Controller.

Finally Main_Controller initiates an asynchronous request through mc_e13 activity to psB_e12 entry of Process_B task, which requests Process_Backup for back up operation through Tuple_Space_B task. Process_Backup performs a backup operation and replies back to Prcoess_B.

The number shown in gray color box under each task provides mapping of UCM path in LQN model.

7.2.6 Step 5: Additional Completions, Environment and Assumptions

Additional "completions" are introduced in the form of environment components i.e. Party_ A_User, ts_Central, Cs_opA and Cs_opB tasks. All of these tasks are added as to achieve suitable performance model. Party_A_User task, which is a reference task, will initiate the scenario. Since, communication infrastructure is based on a distributed tuple space server with two server processes therefore a centralized tuple space server is required to synchronize these processes.

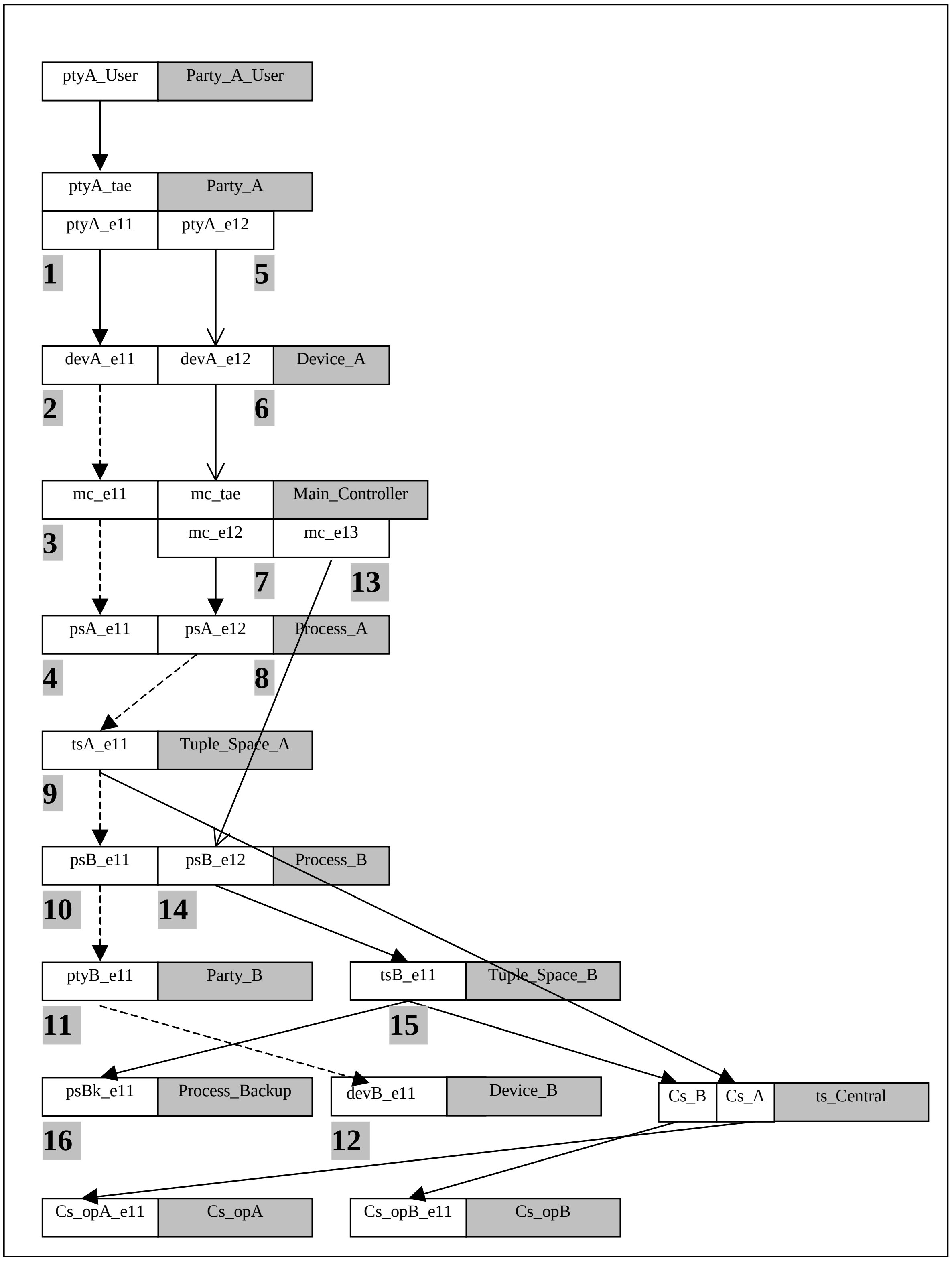


Figure 7.3: Distributed Hand-off Protocol Layered Queuing Network (LQN) Model

ts_Central task, which is acting as centralized server for distributed tuple space servers, will provide synchronization for Tuple_Space_A process through Cs_opA task and Tuple_Space_B process through Cs_opB task.

To follow the analysis a little further, the performance model was created with workload values, and the following assumptions:

- The CPU demands were considered as unity for all entries and activities in the beginning
- The probabilities for the path traversing were considered as 100 % (i.e. value '1' from one entry of a task to entry of another task).
- All the requests for **in** and **out** operations of tuple space are handled by one entry.
- All the tasks are on separate processors.

7.2.7 Step 6: Experimental Results Evaluation

The Distributed Hand-off Protocol LQN model is solved by ParaSRVN simulator. Figure 7.4 shows the model predictions for the delay of the system, giving the sensitivity of response time vs. different number of users for the "Full Demand" or base case (labeled FDMC, with 1 msec for all demands) and for other budget values. The number of threads of the Main_Controller, denoted as m, was also varied with values 1, 2, 6 and 10. Thus, for instance, FDMC(m=1) is a base case with single Main_Controller thread and CPU demands of 1.0 for all the entries and activities. The LQN file of FDMC(m=1) is included in Appendix C1.

The sensitivity of response times and throughputs vs. different number of users for the "Full Demand" with different threads of Main_Controller are shown as FDMC(m=2), FDMC(m=6), and FDMC(m=10) in Figure 7.4 and 7.5 respectively and same data is shown in numerical form in Table 8(a) and 9(a) respectively.

The results are obtained with 95% confidence level (error ranges from and 1% to 5%). Other values of demands were modeled with 10 threads (m = 10):

- the MC(0.05, 0.15, 0.15) curve is obtained with the CPU demand value of 0.05 msec for the entry mc_e11, 0.15 msec for the activity mc_e12, and 0.15 msec for the activity mc_e13 of Main_Controller. The CPU demands of rest of the system are 1.0 msec.

- the DVD(0.1, 0.1) curve is obtained by budgeting the CPU demands to 0.1 msec for the entries of Device_A and Device_B, the CPU demand values for the rest of the system are same as described for the MC(0.05, 0.15, 0.15) curve.
- the PABD (0.1, 0.1) curve is obtained by budgeting the CPU demand values of 0.1 msec for the entries of Process_A and Process_B, the CPU demand values for the rest of the system are same as described for the DVD(0.1, 0.1) curve.
- the PAB (m = user) curve is for multiple threads in Process_A and Process_B, equal to the number of users) with budget CPU demands described for the PABD (0.1, 0.1) curve.

Users	Main_Controller (d=1.0, 1.0, 1.0) Device_A (d=1.0, 1.0) Device_B (d=1.0, 1.0) Process_A (d=1.0, 1.0) Process_B (d=1.0, 1.0) Other system Components (d=1.0)		
	Main_Controller Threads (m=2)	Main_Controller Threads (m=6)	Main_Controller Threads (m=10)
1	10.518	7.936	6.825
31	21.951	12.976	8.021
61	64.308	25.569	12.600
91	263.789	77.915	29.326
121	689.314	258.030	108.101
151	1061.060	594.883	336.250
181	1542.470	882.566	594.886
211	1927.360	1177.000	875.489
241	2336.310	1509.870	1136.060

Table 8 (a): Response Time (msec) With Full CPU Demands

The sensitivity of response times and throughputs vs. different number of users for the “budgeted demands” with different 10 threads of Main_Controller are shown as MC(0.05, 0.15, 0.15), DVD(0.1, 0.1), PABD (0.1, 0.1), and PAB (m = User) in Figure 7.4 and 7.5 respectively and same data is shown in numerical form in Table 8(b) and 9(b) respectively.

Users	Main_Controller Threads (m=10)			
	MC(0.05, 0.15, 0.15) Main_Controller (d=0.05, 0.15, 0.15) Device_A (d=1.0, 1.0) Device_B (d=1.0, 1.0) Process_A (d=1.0, 1.0) Process_B (d=1.0, 1.0) Other system Components (d=1.0)	**DVD(0.1, 0.1)** Main_Controller (d=0.05, 0.15, 0.15) Device_A (d=0.1, 0.1) Device_B (d=0.1, 0.1) Process_A (d=1.0, 1.0) Process_B (d=1.0, 1.0) Other system Components (d=1.0)	**PABD (0.1, 0.1)** Main_Controller (d=0.05, 0.15, 0.15) Device_A (d=0.1, 0.1) Device_B (d=0.1, 0.1) Process_A (d=0.1, 0.1) Process_B (d=0.1, 0.1) Other system components (d=1.0)	**PAB (m = User)** Main_Controller (d=0.05, 0.15, 0.15) Device_A (d=0.1, 0.1) Device_B (d=0.1, 0.1) Process_A (d=0.1, 0.1) Process_B (d=0.1, 0.1) Other system components (d=1.0) Party_A Thread (m=User) Party_B Thread (m=User)
1	6.388	5.610	4.417	3.122
31	7.090	5.993	4.928	3.650
61	9.670	8.805	6.948	4.314
91	22.374	19.993	13.359	6.144
121	112.250	105.530	47.920	11.100
151	331.773	321.029	200.679	25.578
181	597.021	593.844	428.956	78.589
211	847.206	846.439	664.890	202.736
241	1138.340	1112.520	914.523	372.236

Table 8 (b): Response Time (msec) with Budgeted CPU Demands

The LQN files of MC(0.05, 0.15, 0.15), DVD(0.1, 0.1), PABD (0.1, 0.1), PAB (m = User) are included in Appendices C2, C3, C4, and C5 respectively.

From the graphs we can see that the performance starts to degrade around 90 simultaneously active users, for m = 1, but that with 10 threads in the Main Controller this is improved to about 120 users. The detailed inspection of the model outputs confirm that the Main_Controller is a software bottleneck, which requires multi-threading, but the improvement is slight beyond m=10. Main controller demand reduction has only slight effect; device demand reduction has a modest effect; Process_A and Process_B demand reductions have a large effect, and Process_A and Process_B multithreading has a very large effect, roughly doubling the capacity when m=10 for the Main_Controller. The sensitivity of the throughput was also analyzed and shows the effect on system capacity of changes in numbers of users. We can see that throughput saturates after 151 users for MC(0.05, 0.15, 0.15), DVD(0.1, 0.1), and PABD (0.1, 0.1) and after 211 users for PAB (m = User). The same data is shown in numerical form in Tables 8(b) and 9(b).

Users	Main_Controller (d=1.0, 1.0, 1.0) Device_A (d=1.0, 1.0) Device_B (d=1.0, 1.0) Process_A (d=1.0, 1.0) Process_B (d=1.0, 1.0) Other system components (d=1.0)		
	Main_Controller Threads (m=2)	Main_Controller Threads (m=6)	Main_Controller Threads (m=10)
1	0.0010	0.0010	0.0010
31	0.0304	0.0305	0.0306
61	0.0575	0.0595	0.0601
91	0.0720	0.0847	0.0883
121	0.0718	0.0957	0.1092
151	0.0732	0.0949	0.1130
181	0.0712	0.0960	0.1135
211	0.0721	0.0970	0.1126
241	0.0722	0.0959	0.1129

Table 9 (a): Throughput (Users/msec) with Full CPU Demands

Users	Main_Controller Threads (m=10)			
	MC(0.05, 0.15, 0.15) Main_Controller (d=0.05, 0.15, 0.15) Device_A (d=1.0, 1.0) Device_B (d=1.0, 1.0) Process_A (d=1.0, 1.0) Process_B (d=1.0, 1.0) Other system Components (d=1.0)	**DVD(0.1, 0.1)** Main_Controller (d=0.05, 0.15, 0.15) Device_A (d=0.1, 0.1) Device_B (d=0.1, 0.1) Process_A (d=1.0, 1.0) Process_B (d=1.0, 1.0) Other system Components (d=1.0)	**PABD (0.1, 0.1)** Main_Controller (d=0.05, 0.15, 0.15) Device_A (d=0.1, 0.1) Device_B (d=0.1, 0.1) Process_A (d=0.1, 0.1) Process_B (d=0.1, 0.1) Other system components (d=1.0)	**PAB (m = User)** Main_Controller (d=0.05, 0.15, 0.15) Device_A (d=0.1, 0.1) Device_B (d=0.1, 0.1) Process_A (d=0.1, 0.1) Process_B (d=0.1, 0.1) Other system components (d=1.0) Party_A Thread (m=User) Party_B Thread (m=User)
1	0.0009	0.0010	0.0010	0.0010
31	0.0308	0.0307	0.0307	0.0308
61	0.0602	0.0604	0.0606	0.0610
91	0.0890	0.0891	0.0896	0.0903
121	0.1088	0.1094	0.1156	0.1198
151	0.1134	0.1146	0.1258	0.1470
181	0.1135	0.1136	0.1265	0.1678
211	0.1143	0.1141	0.1268	0.1759
241	0.1127	0.1141	0.1258	0.1759

Table 9 (b): Throughput (Users/msec) with Budgeted CPU Demands

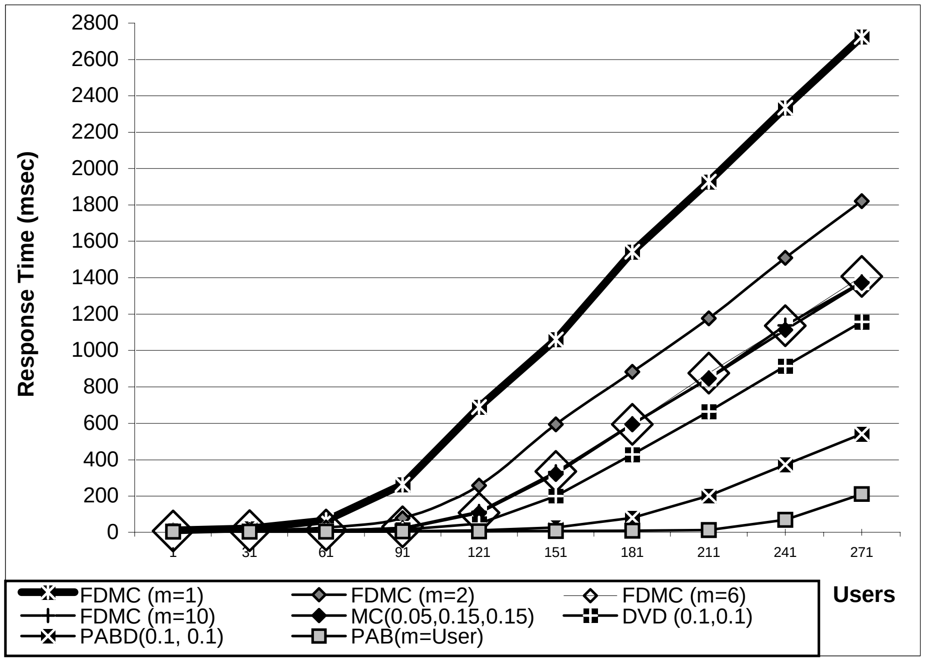


Figure 7.4: Sensitivity of Response (msec) Time to the Number of Users

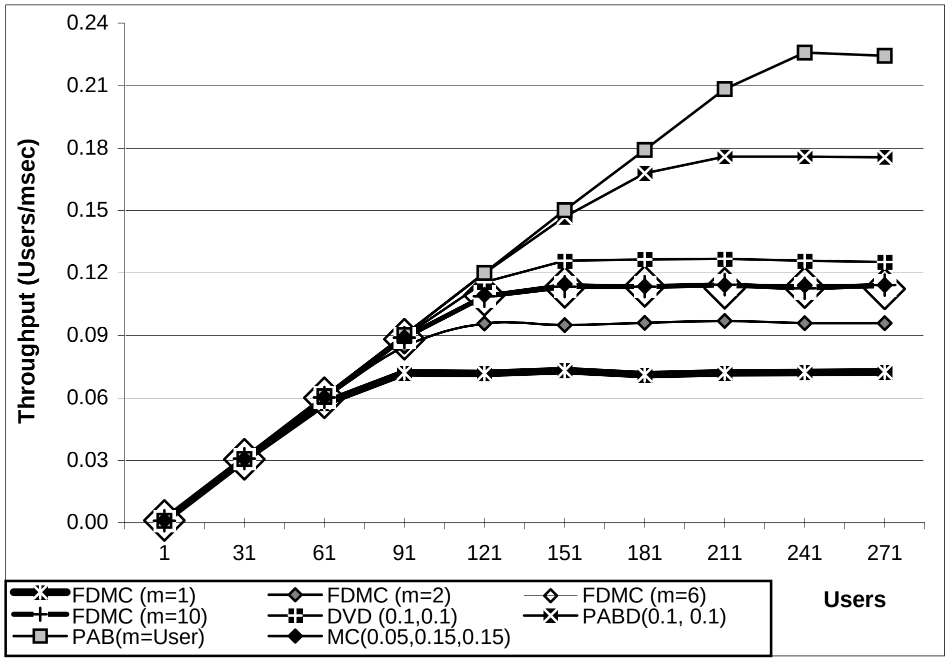


Figure 7.5: Sensitivity of Throughputs (Users/msec) to the Number of Users

7.3 Case Study B – Plain Old Telephone System (POTS) UCM Design

7.3.1 Overview

This case study describes Plain Old Telephone System (POTS) call connection UCM design example, which provides a base UCM design for traditional telephony system. Daniel Amyot originally developed this UCM in[Amyot99b and Amyot00].

7.3.2 Step1: Designer UCM

The root map of POTS UCM Design is shown in Figure 7.6. The scenario consists of five rectangular boxes labeled as SCP, OS, Switch, Orig, and Term, referring to UCM components and three diamond-shaped objects PreDial, PostDial, and Billing, referring to static stubs.

The system components are described as under:

- Orig component refers to caller's telephone set
- Term component refers to callee's telephone set
- Multiple copies of Orig indicate multiple user connections
- Switch provides connectivity to both
- Operations System (OS) component provides billing facility to system
- Service Control Point (SCP) component is used to processes IN features but this component is not used in POTS scenario.

The stubs of UCM path are described as under:

- PreDial stub refers to features that are activated before the number is dialed.
- PostDial stub refers to features that are activated after the number is dialed.
- The Billing has a straight path connecting its input and output. The path has a single responsibility to log the start time of the connection between the caller and callee.
- PreDial stub has a default plug-in that merely connects the input and output paths. PostDial map has provides more functionality. It has associated plug-in map shown in Figure 7.7. The PostDial plug-in either connects caller to callee or notifies the caller that callee is busy.
- ProcessCall stub refers to features dealing with making a connection; the associated plugin map is shown in Figure 7.8. NumberDisplay and ProcessBusy

stubs have default plug-in without any responsibilities. Former displays the caller's number while the later has features associated with the callee being busy.

- ProcessCall plug-in checks whether the callee is idle. If the callee is idle then its status is changed to busy and the process of making the connection is started otherwise the process of notifying the caller, that the callee is busy, starts. The starting point of a call is bound to IN1 input while the idle and busy end points are bound to stub's OUT1 and OUT2 outputs respectively.

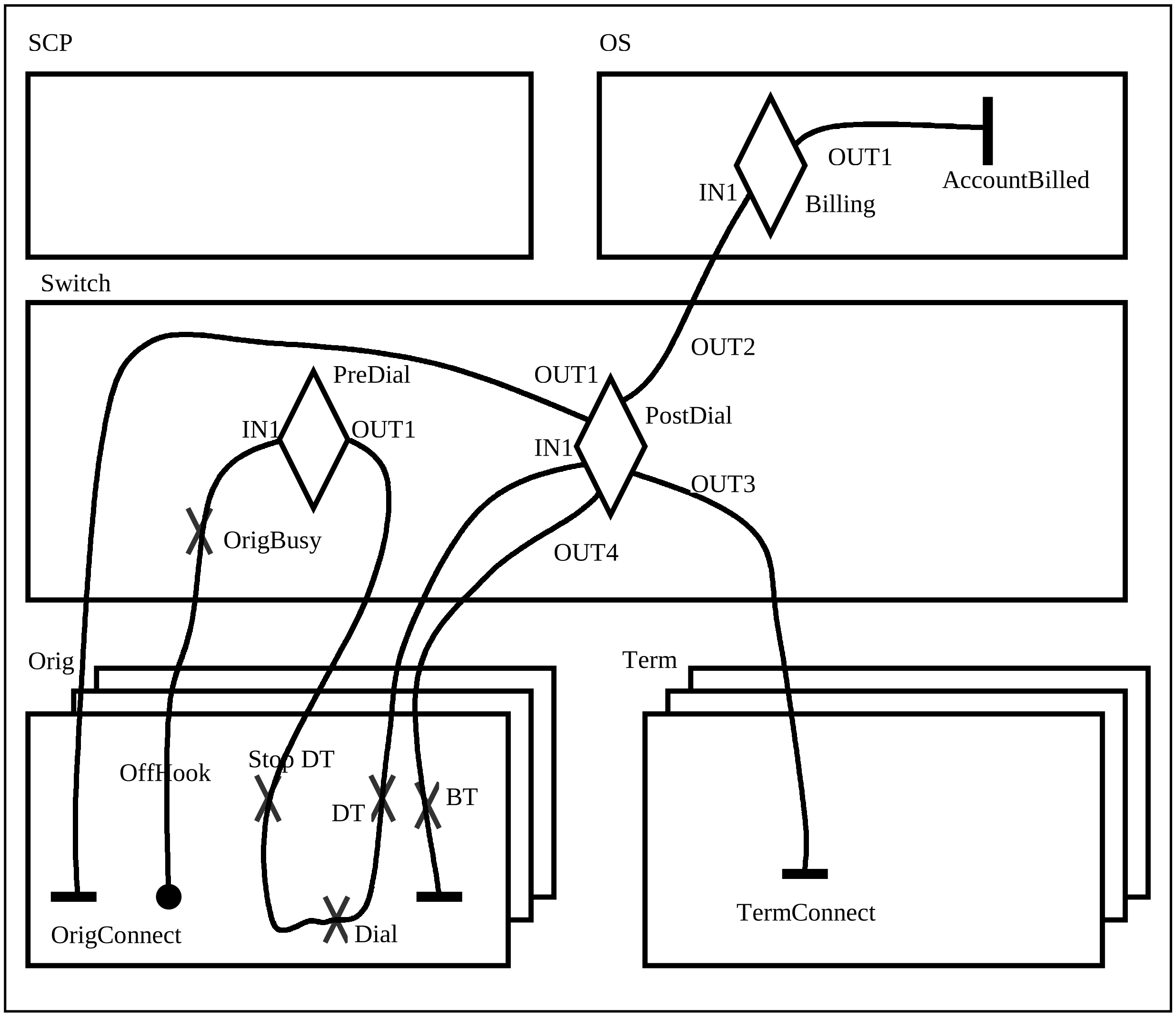


Figure 7.6: UCM Root map for POTS example

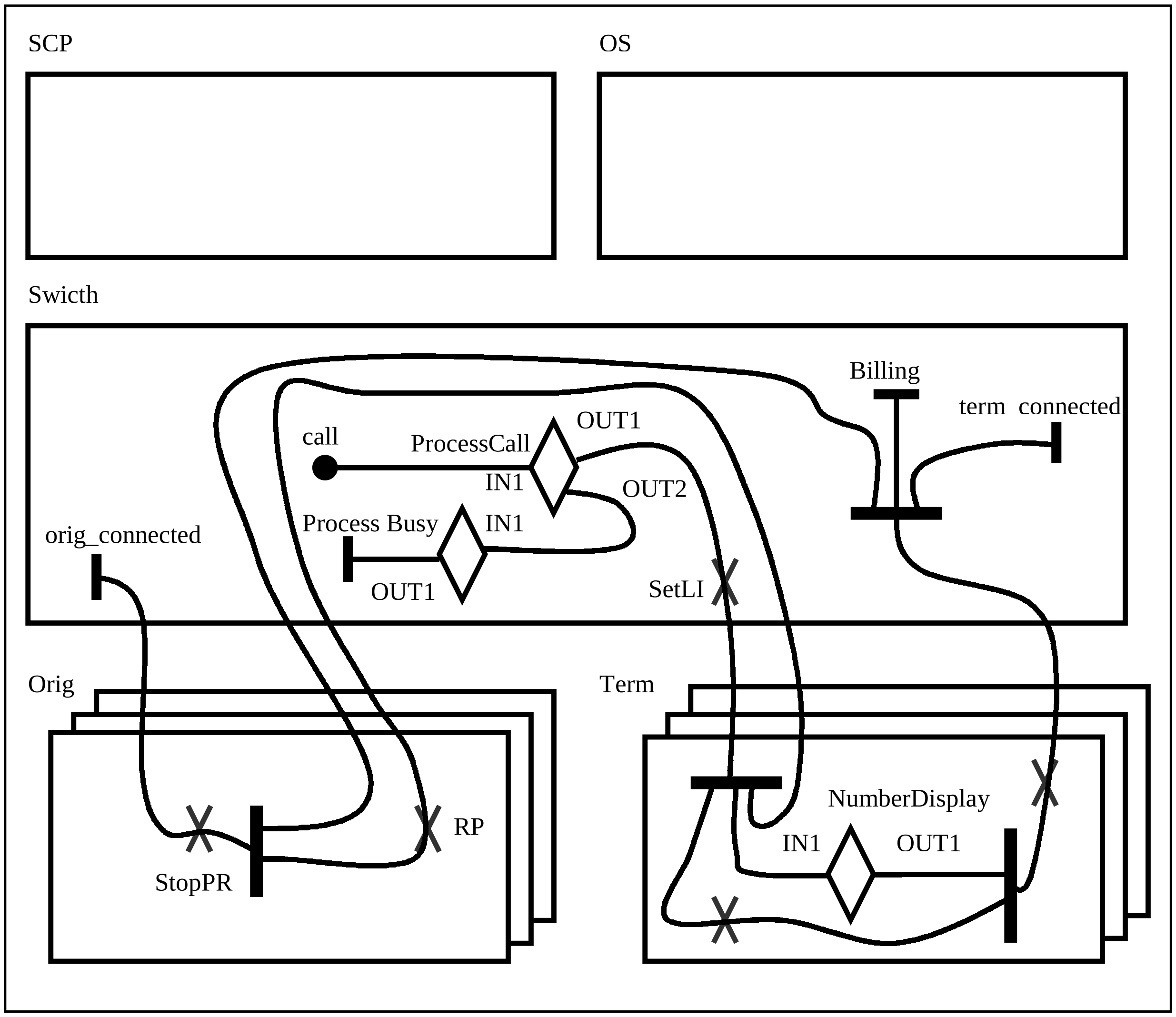


Figure 7.7: PostDial plug-in map

When a user attempt to make a call through POTS then two scenarios are possible. The call can either be setup successfully or indicate that the callee is busy. In successful call connection scenario, the caller picks up the receiver, the switch notes that the caller is busy and caller gets a dial tone. The caller then dials the callee number and dial tone stops. The switch checks the status of callee and finds that the callee is currently idle, and stores the caller number as the callee last incoming number. The callee gets the ring and caller gets the remote ringing tone. The callee picks up the receiver and rings stops. The caller remote ring tone also stops and the billing details are recorded by the operations system and connection is established.

For an unsuccessful call connection scenario, if the switch checks the status of callee and finds that the callee is busy then the caller gets a busy tone and connection is not made.

7.3.3 Step2: Budgets

The CPU time demand of each responsibility of the Switch is chosen 0.25 msec and for the rest of system responsibilities 1.0 msec in beginning of the budgeting process. Though the components used in this example are familiar types but due to lack of availability of data for CPU time demands, sensitivity of system response time and throughput are considered as targets to budget CPU time demand for each responsibility.

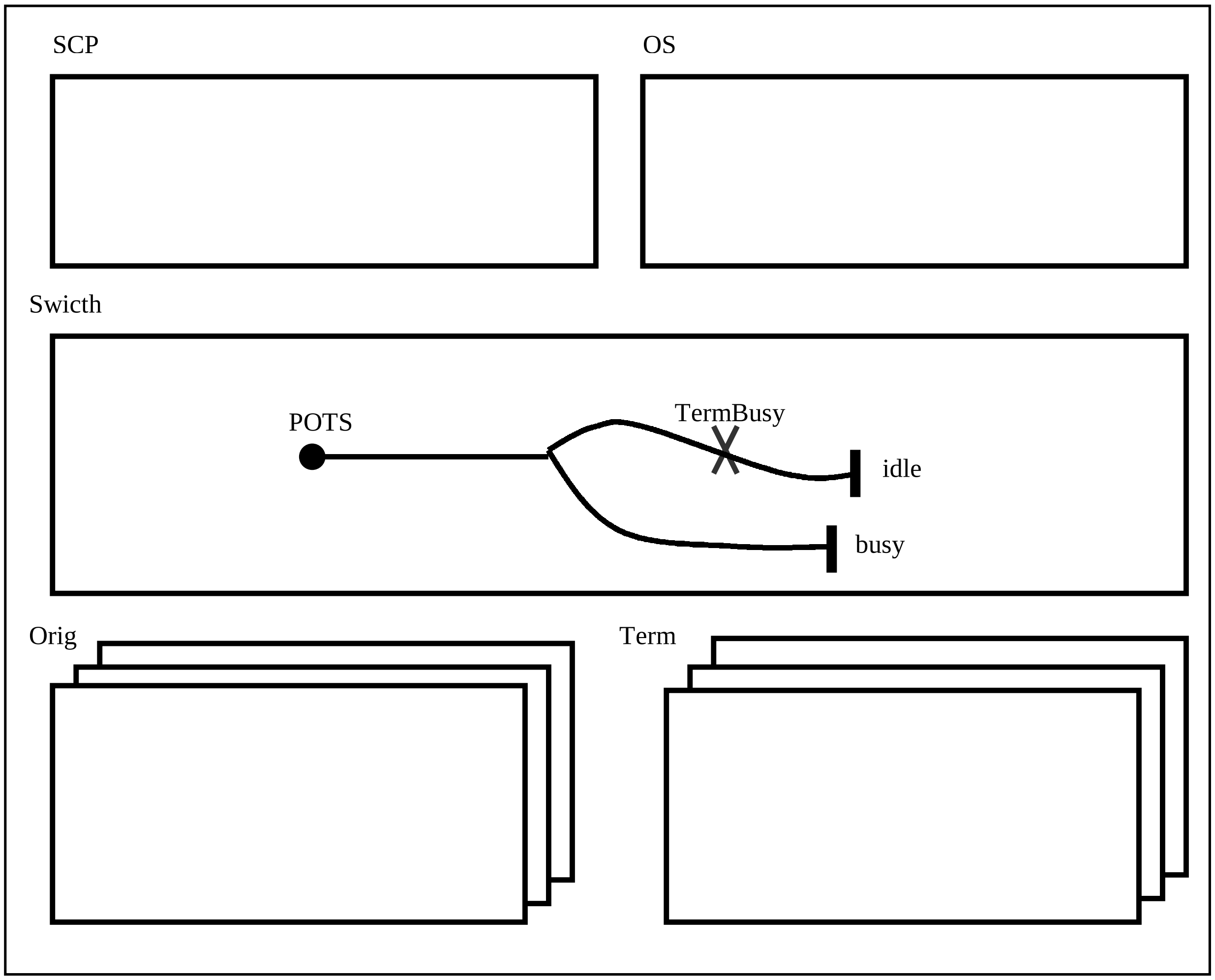


Figure 7.8: ProcessCall Plug-in map

7.3.4 Step 3: UCM Completion and Infrastructure Components

POTS UCM Design is taken from Dorin B. Petriu thesis [Petriu01a]. He experimented developed a generative tool, LQN2UCM converter, for LQN model extraction. It is assumed that POTS UCM Design is a “complete” UCM for model extraction.

7.3.5 Step 4: POTS LQN Model

The UCM2LQN converter generated a large number of default activities in order to create activity connections and calling relationships that correspond to the UCM input. Every fork and join leads to the creation of as many default activities as there are path segments leading into or away from the given fork and join. Thus an AND fork with one input and three output branches will lead to the creation of four default activities to represent it. The POTS UCM design features two AND forks shown in Figure 7.7 in Term and Switch, two AND join shown in Figure 7.7 in Orig and Term, and one OR fork shown in Figure 7.8 in Switch.

The LQN model for POTS shown in Figure 7.9 as jLQNDef graphic, with many activities suppressed. The Figure 7.8 includes one new task, RefTask1, which has been introduced as additional “completion” which will be described in next step. The LQN file for POTS is included in Appendix A.

The starting point of the scenario is RefTask1 task, which sends a synchronous message to Orig task. It indicates that the receiver is picked up and RefTask1 task waits for the reply from Orig task. Orig task interacts with the Switch synchronously.

Orig task first requests a dial tone, then that a connection be established with the other party whose number is dialed, and finally that the connection be enabled so the two parties can talk. The Switch in turn first makes a call to the Term process to create the call. After the callee picks up, Switch sends an asynchronous message to Term process to confirm that the two parties can now talk to each other. Switch also sends an asynchronous message to log the start time of the connection to OS.

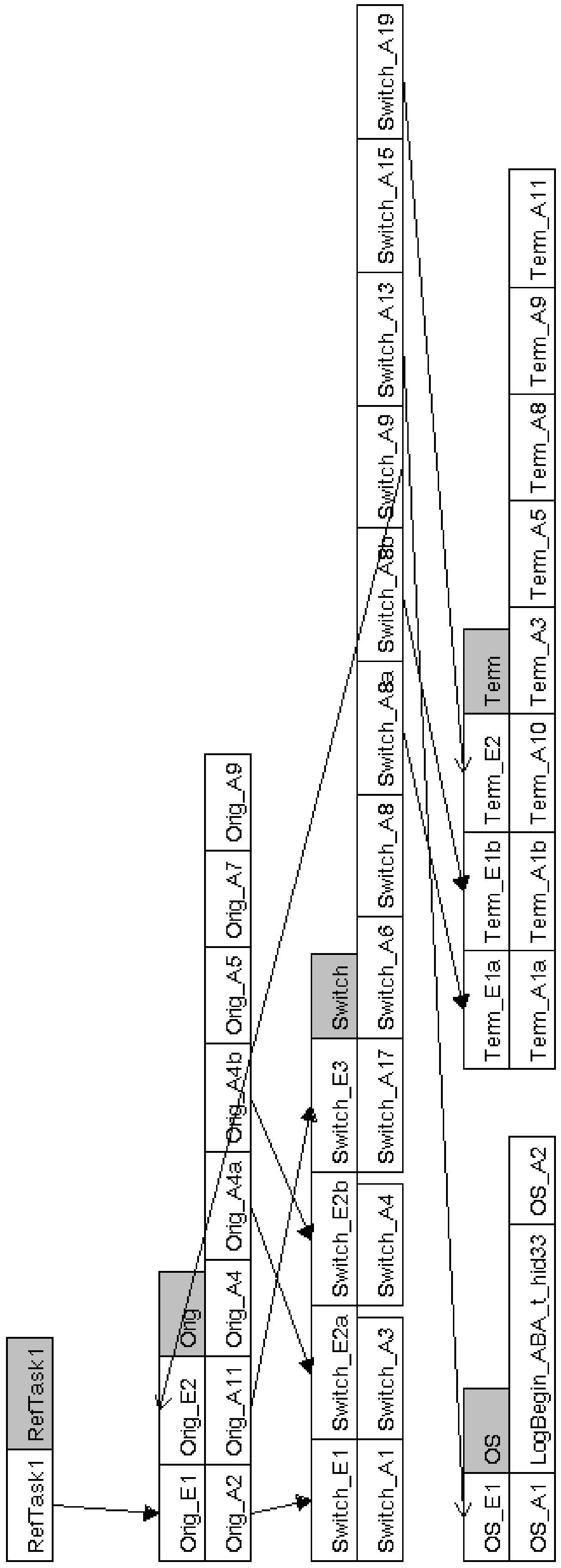


Figure 7.9: POTS reduced Layered Queuing Network (LQN) Model

7.3.6 Step 5: Additional “Completions”, Environment and Assumptions

The UCM2LQN converter introduced additional “completion” in the form of environment components i.e. RefTask1 and one common processor for all the tasks in extracted LQN model. RefTask1 task and processor are added to achieve suitable performance model. RefTask1 task, which is a reference task, will initiate the scenario.

Since, the end of the UCM path did not come back to the start point and the system had open arrivals, the UCM2LQN converter derived it as asynchronous interaction between RefTask1 task and Orig task. The sensitivities of response time (mean delay) and throughput (capacity) of overall system can only be observed when RefTask1 waits for the reply from Orig, therefore, the asynchronous call between RefTask1 and Orig is replaced with synchronous. Performance model was created with workload values and the following assumptions:

- RefTask1 and Orig tasks interact synchronously
- The number of Orig and Term tasks are equal to the number of RefTask1 tasks
- The Switch and the OS share a common processor while all the other tasks have their own separate processors
- The CPU demands were considered as unity for all entries and activities in the beginning
- The probabilities for the path traversing were considered as 100 % (i.e. value ‘1’ from one entry of a task to entry of another task)

The model is solved by ParaSRVN simulator, SPEX tool made the analysis easier for sequencing repeated runs over ranges of parameter values. The model gave the sensitivity of throughput and response time of overall system for a range of RefTask1, Orig and Term tasks, and for different budgeted CPU time demands of system components.

7.3.7 Step 6: Experimental Results Evaluation

The predictions of the simulation model for delay (the sensitivity of average response time) and capacity (the sensitivity of throughput) of overall system are plotted to the number of call connections i.e. different number of RefTask1 users for the budgeted CPU demand values with exponential scale in Figures 7.10 and with logarithmic scale 7.11. The Figures 7.12, 13 and 15 are the plots of processor utilization for the Switch, the Orig and the Term with exponential scale. The simulation results showed that multiple Switch threads degraded system performance. So, only one switch thread was used for all the simulation models. The case Switch (CPUd=0.25 msec) is considered as the base case. In this case, the CPU demands for all the activities of the Switch are 0.25 msec and for the rest of the system 1 msec. The LQN file of Switch (CPUd=0.25 msec) is included in Appendix D1. Other values of budgeted demands with single Switch thread and CPU demand of 0.25 msec of Switch.

- Orig (CPUd=0.5 msec) curve is obtained by budgeting the CPU demands to value 0.5 msec for all the activities of Orig, the CPU demand values for the rest of the system are same as described for the Switch (CPUd=0.25 msec) curve.
- Term (CPUd=0.1 msec) curve is obtained by budgeting the CPU demands to 0.1 msec for all the activities of Term, the CPU demand values for the rest of the system are same as described for the Orig (CPUd=0.25 msec) curve.

Case Orig (CPUd=0.5 msec)

From the graphs we can see that the performance is somewhat better when the CPU demand of Orig was budgeted to 0.5 msec with the CPU demand of the Switch of 0.25 msec and the rest of system of 1.0 msec. This case is plotted as Orig (CPUd=0.5 msec). Detailed inspection of the model outputs shows that the processor utilization of the switch reached to maximum 51.8% for 500 users. The mean delay and the throughput of the system were 95.258 msec and 0.193 Users/msec respectively for 500 users. The LQN file of Orig (CPUd=0.5 msec) is included in Appendix D2.

Case Term (CPUd=0.1 msec)

From the graphs we can see that the performance starts to degrade when the CPU demand of term was budgeted to 0.1 msec with the CPU demand of the Switch of 0.25 msec, Orig of 0.5 msec and the rest of system of value 1.0 msec. This case is plotted as Term (CPUd=0.1 msec). Detailed inspection of the model outputs shows that the processor utilization of the switch reached 84.0% at 500 users. The LQN file of Term

(CPUd=0.1 msec) is included in Appendix D3. The same data is shown in numerical form in Tables 10, 11, 12, 13 and 14.

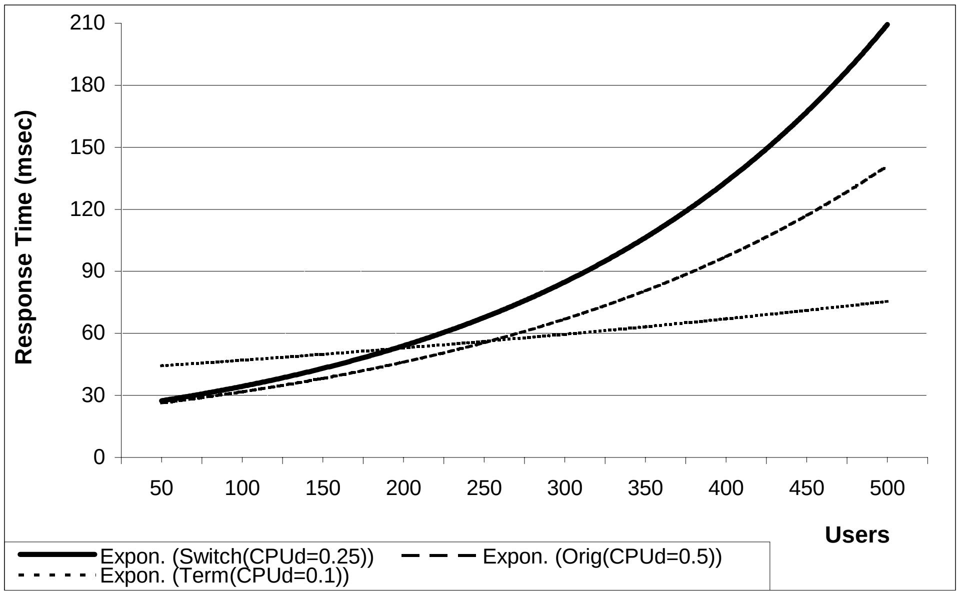


Figure 7.10: Sensitivity of Response Time (msec) to the No of RefTask1 Users

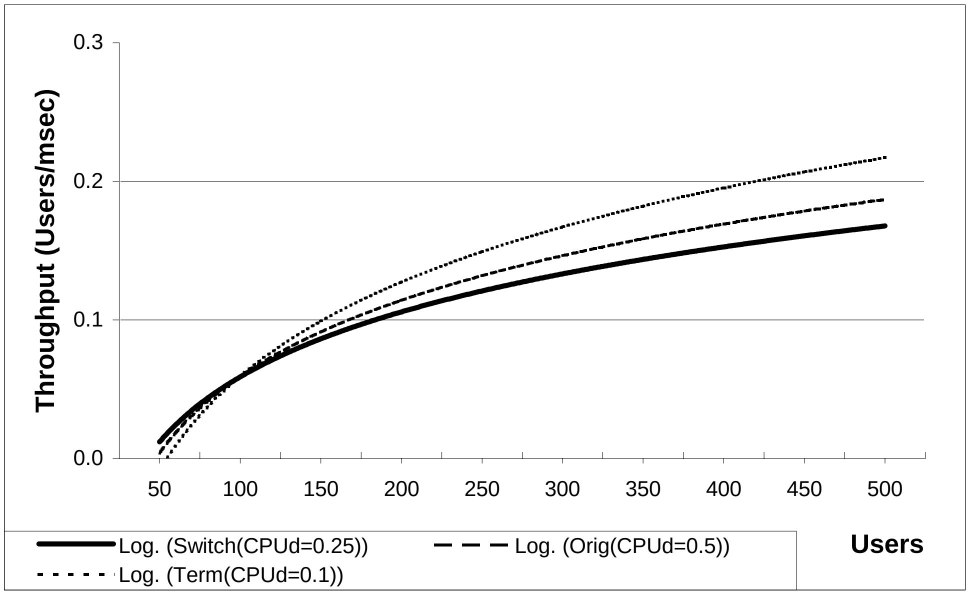


Figure 7.11: Sensitivity of Throughputs (Users/msec) to the No of RefTask1 Users

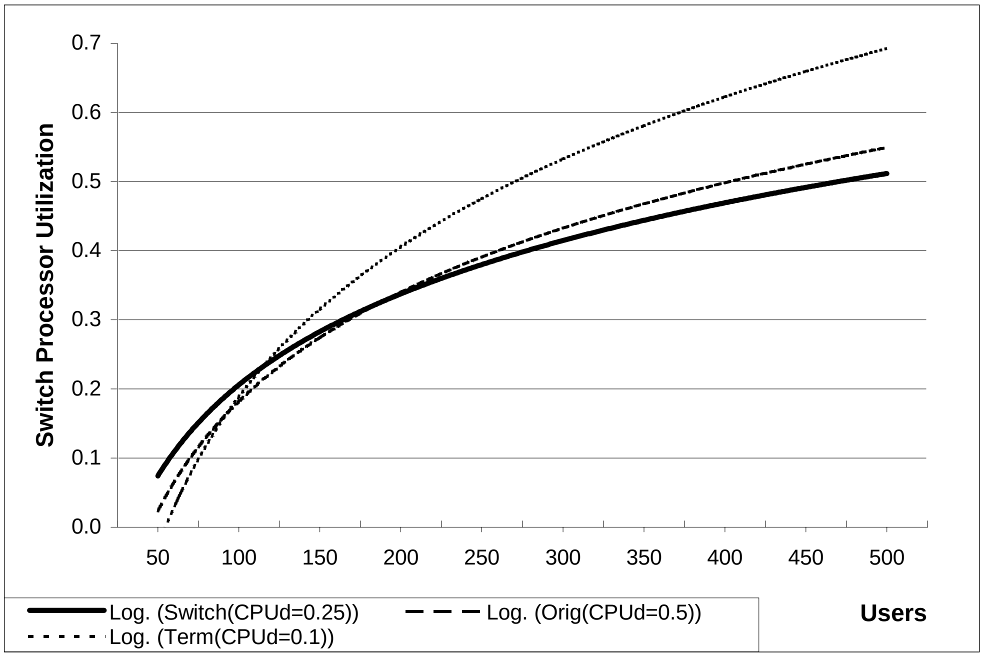


Figure 7.12: Switch Processor Utilization to the No of RefTask1 Users

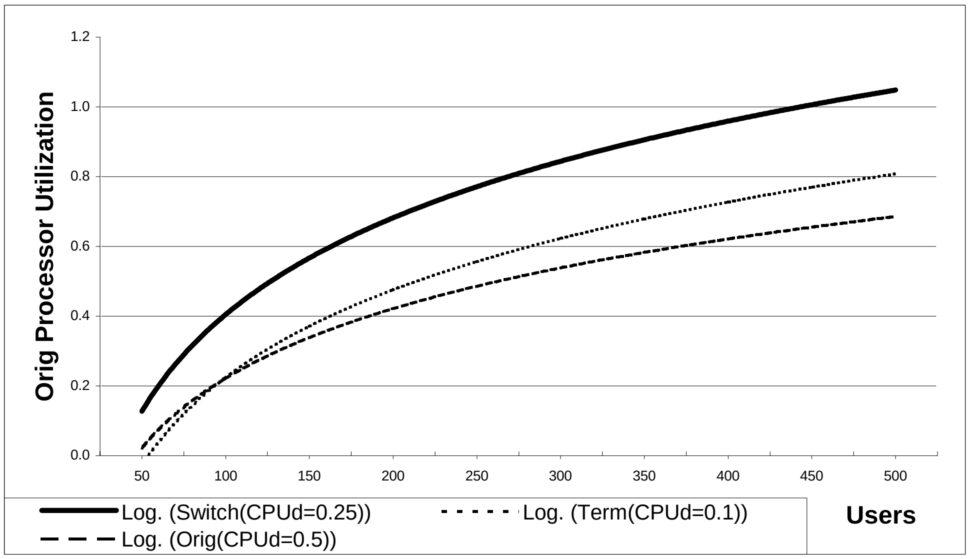


Figure 7.13: Orig Processor Utilization to the No of RefTask1 Users

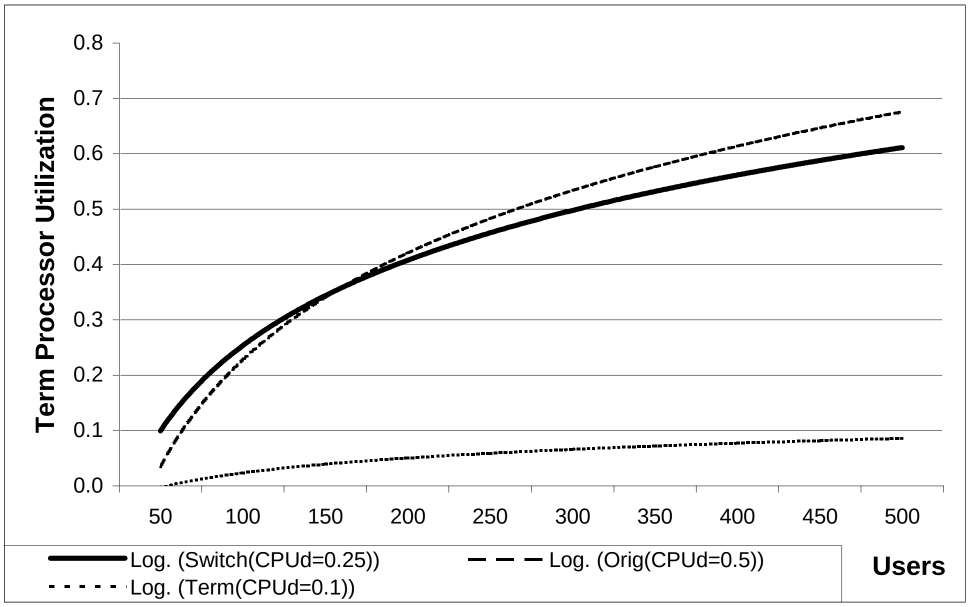


Figure 7.14: Term Processor Utilization to the No of RefTask1 Users

User	Switch(CPUd=0.25 msec)	Orig(CPUd=0.5 msec)	Term(CPUd=0.1 msec)
50	26.567	51.952	92.393
100	28.981	4.223	28.340
150	46.187	106.062	44.447
200	57.955	56.493	35.113
250	70.674	49.039	47.341
300	94.426	101.169	83.227
350	104.711	97.045	69.533
400	147.914	102.024	70.317
450	157.298	115.066	60.966
500	186.520	95.258	87.266

Table 10: POTS Response Time (msec)

User	Switch(CPUd=0.25 msec)	Orig(CPUd=0.5 msec)	Term(CPUd=0.1 msec)
50	0.025	0.029	0.034
100	0.048	0.048	0.042
150	0.080	0.069	0.079
200	0.104	0.104	0.103
250	0.101	0.127	0.135
300	0.153	0.137	0.145
350	0.139	0.172	0.179
400	0.162	0.178	0.196
450	0.165	0.186	0.221
500	0.166	0.193	0.264

Table 11: POTS Throughputs (Users/msec)

User	Switch(CPUd=0.25 msec)	Orig(CPUd=0.5 msec)	Term(CPUd=0.1 msec)
50	0.110	0.085	0.124
100	0.170	0.105	0.137
150	0.237	0.263	0.218
200	0.342	0.294	0.317
250	0.365	0.362	0.386
300	0.465	0.506	0.461
350	0.479	0.519	0.626
400	0.470	0.508	0.676
450	0.483	0.524	0.658
500	0.492	0.518	0.840

Table 12: Switch Processor Utilization

User	Switch(CPUd=0.25 msec)	Orig(CPUd=0.5 msec)	Term(CPUd=0.1 msec)
50	0.191	0.099	0.142
100	0.345	0.152	0.172
150	0.475	0.296	0.262
200	0.660	0.374	0.373
250	0.785	0.455	0.452
300	0.933	0.592	0.560
350	0.968	0.634	0.685
400	0.976	0.648	0.802
450	0.991	0.652	0.796
500	0.991	0.672	0.961

Table 13: Orig Processor Utilization

User	Switch(CPUd=0.25 msec)	Orig(CPUd=0.5 msec)	Term(CPUd=0.1 msec)
50	0.137	0.110	0.016
100	0.240	0.140	0.016
150	0.300	0.330	0.030
200	0.389	0.350	0.039
250	0.408	0.445	0.047
300	0.550	0.610	0.060
350	0.565	0.650	0.072
400	0.585	0.649	0.089
450	0.559	0.629	0.081
500	0.619	0.640	0.103

Table 14: Term Processor Utilization

7.4 Budget Revision

Sensitivity analysis provides the variation in the output of a model qualitatively or quantitatively to different sources of variation. It can be used as a guide for the revision of budgets when the budgeted resource demands are unable to meet the performance targets. It might be possible that the analysis results reveal the bottleneck of the system, which points to most profitable change in the budgeted values. The values of the resource demand can be selected in such a manner that it would reduce the bottleneck of the system and provide the desired performance. If the budgeted values are unable to meet the satisfactory performance then the architecture adjustment can be done as another option for the desired performance goal.

7.5 Conclusions

Performance budgeting approach for an incomplete software design specification has been presented, using Distributed Hand-Off Protocol and Plain Old Telephone System (POTS) UCMs case studies. The approach presented the idea of “completion” for an incomplete software specification and budgeting of CPU time demands for different system components. Several simulation models are developed and analyzed for the sensitivity of average response time as mean delay and the sensitivity throughput as the capacity of overall system. The experimental results are presented, involving tables and graphs for budgeted CPU time demands of different systems components.

In POTS, the Term demand reduction has a large effect and Orig budget demand reductions have a small effect to the overall of the performance of system; OS demand reductions have no effect on the performance of the system. Further performance improvement might be possible with architecture adjustment, which will be discussed in next chapter.

In Distributed Hand-Off Protocol, the “completion” was introduced in the form of tuple space at UCM level. It could be inserted at LQN level by using *The LQN Component Filter*, which would significantly reduce the effort by introducing “completion” as tuple space as single network component LQN task. The inserted components are bound to the source and destination tasks of the call, and either to existing processors or to new devices of their own. The semi-automated model building significantly reduced the effort to construct a model for from an high-level specification of a software design. The approach should easily scale to larger systems.

7.6 Summary

This chapter describes two industry relevant case studies which walk through various steps of budget analysis road map. The are as follows:

1. Distributed Hand-Off Protocol UCM design covers most of the steps of the road map and
2. Plain Old Telephone System (POTS) UCM design is a “complete’ specification. It covers the road map steps from the model extraction through an automated tool to the result evaluation.

8.0 Architecture Adjustment

8.1 Introduction

This chapter presents an approach to deal with performance shortfalls by adjusting the system design. The approach depends on the evaluation results. If the results are not satisfactory then some changes are required. If predictions show inadequate performance, one approach is to tighten the budgets until the predictions are in the green zone. Alternatively one could adjust the design in the UCM domain, or the implementation options in terms of “completions” and the environment. Architecture adjustment can be suggested at software level as well as hardware level. Here, the adjustments will be presented by modifying the completions representing the system environment in the Plain Old Telephone System (POTS) example described in previous chapter.

8.2 Architecture Adjustment in POTS

8.2.1 POTS Architecture

It is assumed that POTS performance results evaluated in previous chapter show that the budgeted CPU time demands of components are unable to achieve satisfactory performance due to the fact that the Switch and the OS were sharing a processor.

POTS call connection UCM design is based on real world telephone system. It is not feasible to suggest adjustments in POTS UCM design because it provides the base UCM of overall telephony system. The necessary adjustments are can be provided in POTS LQN model. The model reveals that POTS architecture consists of five tasks including a Switch, which acts like a server task. It provides call connectivity to caller (originator) and callee (receiver). The Switch hardware architecture details are beyond the scope of this research but it is possible to suggest and demonstrate architecture adjustments in the LQN environment in POTS for desired performance targets.

8.2.2 POTS Architecture Adjustment Options

Numbers of options are available to make changes in POTS LQN model. One possibility is to make some changes in activity connections. If, there are number of activities in a given task and the operation does not depend on sequence the activities then by introducing parallelism in activities connection might improve the performance of the system. POTS call connection depends on sequence of operation for a particular call and all the activities operations are inter-related with each other.

8.2.3 Processors Adjustment in POTS Model

Another possibility is to make hardware level adjustments in the model by introducing new hardware. Although hardware adjustment is expensive, tradeoff should be made between system performance and the cost. Since there is only processor in POTS LQN model and more processor can be introduced to achieve the desired performance.

The POTS LQN model described in previous chapter reveals that there is only one processor allocated to all the tasks and it becomes a bottleneck when number of users increases to a certain limit. So, it is desired to remove this bottleneck by introducing a separate processor for each task.

Following assumptions are made for the new model after budgeting all the values of CPU demands for different components:

- All the tasks are on separate processors
- The CPU time demands of value
    - "1.0 msec" for all entries and activities of the OS
    - "0.5 msec" for all the entries and activities of the Orig
    - "0.25 msec" for all the entries and activities of the Switch
    - "0.1 msec" for all the entries and activities of the Term

The model is solved by ParaSRVN simulator using SPEX for sequencing repeated runs over ranges of parameter values. The model gave the sensitivity of throughput and response time of overall system for a range of RefTask1 and for different budgeted CPU time demands of components with new hardware.

8.2.4 Results Evaluation

The comparison of sensitivity analysis graphs of response time to the number of users for the systems with shared and separate processors are shown in Figure 8.1 and numerical data for above graphs is given in Table 15. For the system with shared processor, OS and Switch share a processor. Dark curves in Figure 8.1 represent system with the shared processor whereas the system with separate processors is represented by light curves. These curves are plotted with exponential scales.

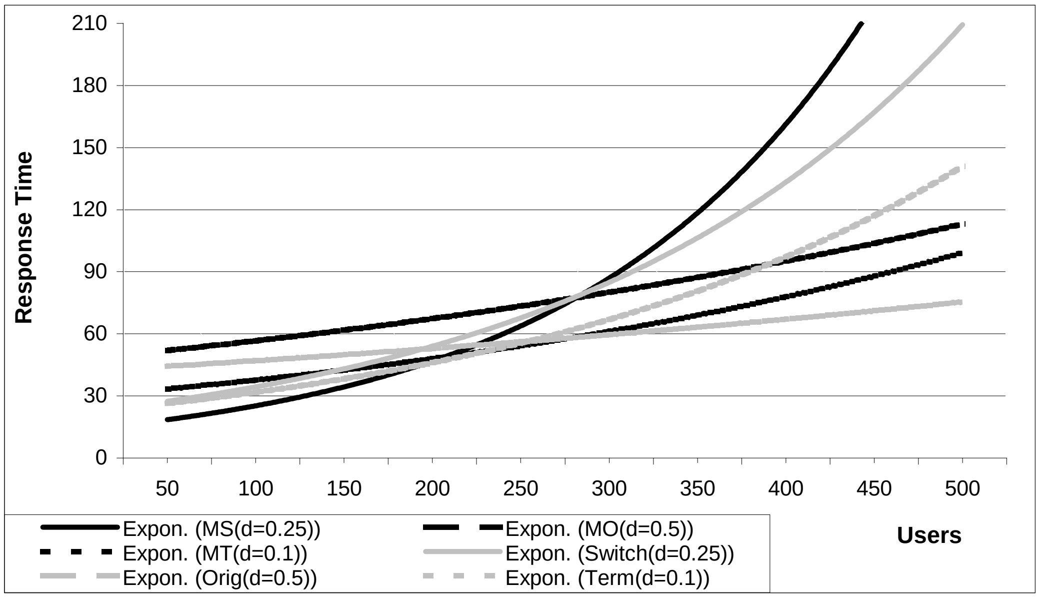


Figure 8.1: Sensitivity of Response Time to the No of Users

Table 15 is divided in three main columns: Users, Separate Processors and Shared Processors. Each sub column shows mean delays values for different budged CPU demand values of the Switch, the Orig, the Term, and the OS. As predicted, the response times for the system with separate processors are reduced compared to the response times of the system with shared processor. The maximum percentage improvement is 63.08% for 50 users when the Switch, the Orig, the Term and the OS have the budgeted CPU demands of values 0.25 msec, 0.5 msec, 0.1 msec, and 1.0 msec respectively. The LQN files of separate processor systems are included in Appendices E1, E2, E3. The percentage improvement is greater for a lighter load because of small number of users.

Users	Separate Processors			Shared Processor		
	Switch(d=0.25) OrigO(d=1.0) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) Orig(d=0.5) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=0.1) OS(d=1.0)	Switch(d=0.25) OrigO(d=1.0) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=0.1) OS(d=1.0)
50	2.29	74.68	20.91	26.57	51.95	92.39
100	100.70	43.80	35.89	28.98	4.22	28.34
150	42.79	77.89	67.22	46.19	106.06	44.45
200	128.21	46.71	50.04	57.95	56.49	35.11
250	57.80	65.49	47.36	70.67	49.04	47.34
300	100.99	98.38	81.10	94.43	101.17	83.23
350	126.54	78.05	101.55	104.71	97.05	69.53
400	156.14	64.25	63.12	147.91	102.02	70.32
450	205.71	130.33	83.30	157.30	115.07	60.97
500	174.81	139.65	76.89	186.52	95.26	87.27

Table 15: POTS Mean Delays with processor adjustment

The comparison of sensitivity analysis graphs of throughputs to the number of users for systems with shared and separate processors were also analyzed and shown in Figure 8.2 with logarithmic scale and numerical data for the mentioned graphs is given in Table 16.

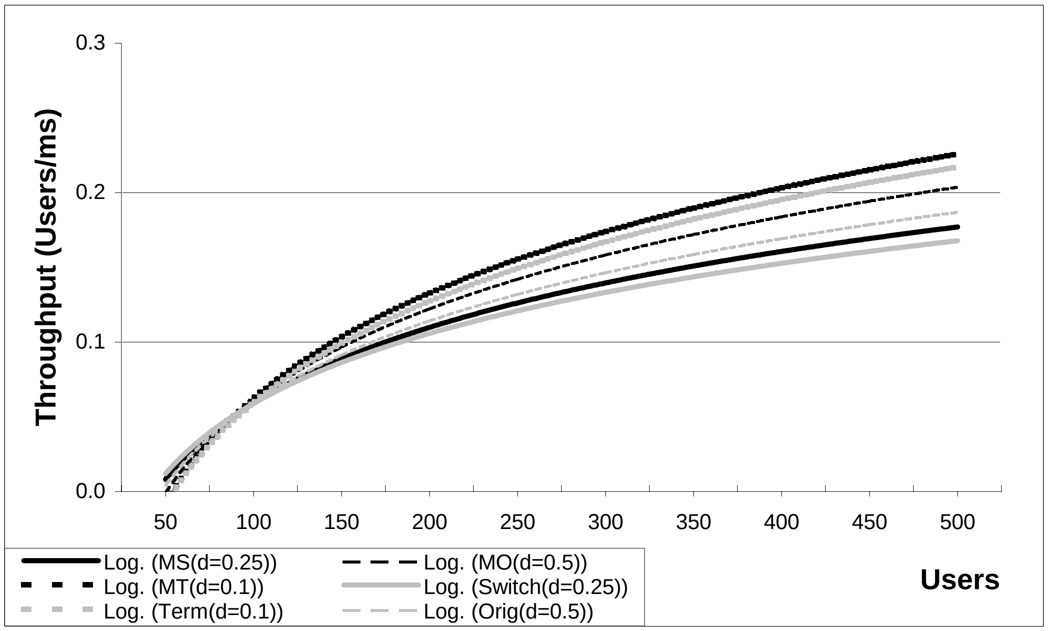


Figure 8.2: Sensitivity of Throughputs to the No of Users

The throughput of the system with processor adjustment does not improve as much as the mean delays values. This is due to the fact that when the OS does not utilize processor whether it shares the processor with the Switch. There was no change in throughput when the maximum mean delay was improved to 63.08% for 50 users when the budgeted CPU demands of values of 0.25 msec, 0.5 msec, 0.1 msec, and 1.0 msec for the Switch, the Orig, the Term and the OS respectively. The maximum percentage in throughput improvement is 7.69% for 100 users when the budgeted CPU demands of the values of 0.25 msec, 1.0 msec, 1.0 msec, and 1.0 msec were for the Switch, the Orig, the Term and the OS respectively.

Users	Separate Processors			Shared Processor		
	Switch(d=0.25) OrigO(d=1.0) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) Orig(d=0.5) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=0.1) OS(d=1.0)	Switch(d=0.25) OrigO(d=1.0) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=0.1) OS(d=1.0)
50	0.026	0.023	0.034	0.025	0.029	0.034
100	0.056	0.053	0.055	0.048	0.048	0.042
150	0.073	0.076	0.076	0.080	0.069	0.079
200	0.091	0.122	0.109	0.104	0.104	0.103
250	0.133	0.117	0.127	0.101	0.127	0.135
300	0.135	0.150	0.146	0.153	0.137	0.145
350	0.146	0.172	0.188	0.139	0.172	0.179
400	0.170	0.202	0.216	0.162	0.178	0.196
450	0.176	0.201	0.239	0.165	0.186	0.221
500	0.187	0.218	0.265	0.166	0.193	0.264

Table 16: POTS Throughputs with processor adjustment

Figure 8.3 shows the comparison of the Switch processor utilization for the shared processor system and separate processors system with logarithmic scale. When separate processors were assigned to the OS and the Switch, the processor utilization of the Switch was reduced from 12.4% to 4.3 % when the maximum mean delay was improved to 63.08% for 50 users when the budgeted CPU demands of values of 0.25 msec, 0.5 msec, 0.1 msec, and 1.0 msec for the Switch, the Orig, the Term and the OS respectively. For the same values of budgeted CPU demands the maximum processor utilization was 84% for 500 users for the shared processor system and it reduced to 45.3% for the separate processors system. Table 17 shows the same result in numerical form.

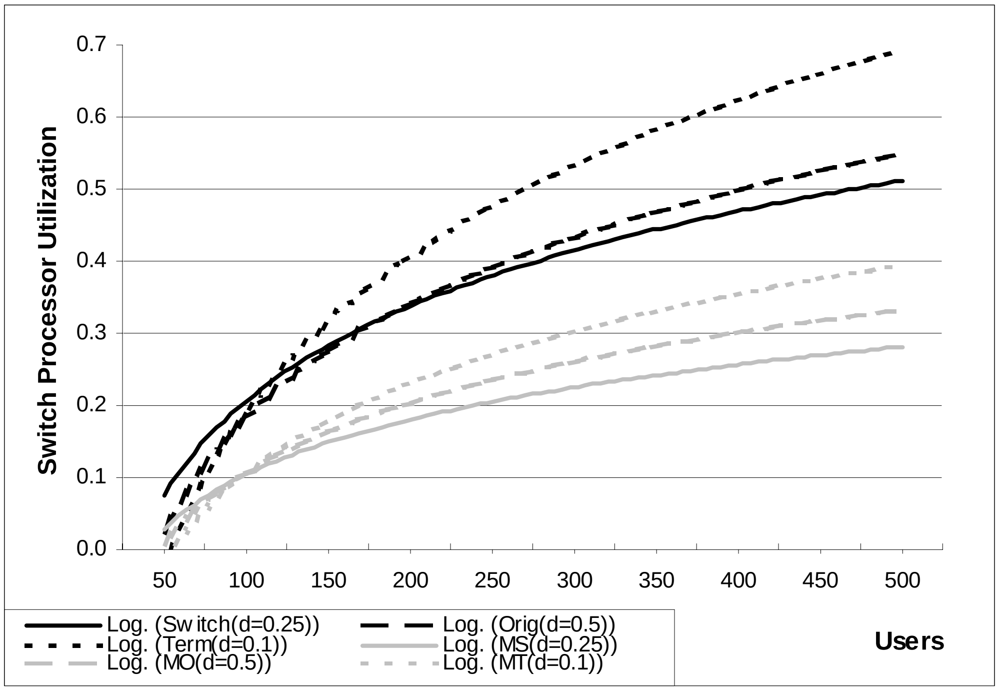


Figure 8.3: Switch Processor Utilization the No of Users

Users	Separate Processors			Shared Processor		
	Switch(d=0.25) OrigO(d=1.0) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) Orig(d=0.5) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=0.1) OS(d=1.0)	Switch(d=0.25) OrigO(d=1.0) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=1.0) OS(d=1.0)	Switch(d=0.25) OrigO(d=0.5) Term(d=0.1) OS(d=1.0)
50	0.027	0.056	0.043	0.110	0.085	0.124
100	0.110	0.084	0.097	0.170	0.105	0.137
150	0.136	0.140	0.168	0.237	0.263	0.218
200	0.159	0.157	0.148	0.342	0.294	0.317
250	0.209	0.191	0.197	0.365	0.362	0.386
300	0.250	0.288	0.276	0.465	0.506	0.461
350	0.261	0.278	0.358	0.479	0.519	0.626
400	0.267	0.329	0.376	0.470	0.508	0.676
450	0.263	0.332	0.400	0.483	0.524	0.658
500	0.256	0.348	0.453	0.492	0.518	0.840

Table 17: Switch Processor Utilization with processor adjustment

8.3 Summary

The chapter describes the architecture adjustment as mean of satisfactory performance results when the budgets are failed to meet the performance targets. His chapter covers the following:

1. POTS example with possible architecture adjustment and
2. The performance results comparison with and without adjustments for POTS example

9.0 Conclusions and Future Work

9.1 Conclusions

9.1.1 Objectives and Requirements

The main focus of the thesis was to describe a performance analysis approach for the software design (especially for the UCM designs), which is based on budgets or estimated figures for the resource demands of all the parts and operations of the system. The budget or estimated figure may be based on prediction, intuition or past experiment results. The key element of this approach is the planning of budgets for the resource demands of all the parts and operations of system and validation check for the required performance.

9.1.2 Thesis Contributions

Different concepts are used from software performance engineering and distributed communication software systems engineering during the development of this approach. The main contributions of this thesis are:

- *The approach of Performance Budgeting process* with based on the Road Map described in Chapter 3. Performance Budgeting provides different steps to carry out performance analysis before the development and implementation of any software system to meet performance goal.
- *Definition of the term "completion"* to overcome the incompleteness of a design specification. We suggest and provide ways to add missing components and parameters to an incomplete specification of a software design. An incomplete specification shows the thinking of a software designer at some early stage of development. The specification is very abstract and may omit some necessary details, which are important for a suitable performance model.
- *Demonstration of the "completion"* described in Chapter 5. The development of four network filters provides automation to the "completion" at LQN level only (model level). Different network components are incorporated in different examples in the form of a single LQN task representing a simple message passing network component (a single network component), a generalized protocol suite

(LQN network subsystem), a shared resource server LQN task and a multicast network LQN task.

- *Measurements and modeling of tuple spaces*. Measurements are done on J-Spaces (J-Space Technologies Inc.) [JSpaces] and the performance model of J-Spaces is constructed and analyzed described in Chapter 6.
- *Two end-to-end Case Studies* on UCM design examples from industry, described in Chapter 7.

These contributions provide a framework to allow for the reuse of performance information, a process for tracking and correcting performance attributes of designer driven the project, components based performance analysis as well as overall system based performance analysis.

Two case studies demonstrated different aspects of the usability of the approach in Chapter 7.0. Distributed Hand-off Protocol UCM design case study involved all the necessary steps mentioned in the budget analysis road map, the usability of the LQN Component filters and tuple spaces. Plain Old Telephone System (POTS) UCM design case study involved only the budgeting of the CPU demands for the system assuming already available “complete” UCM and extracted LQN model for the system.

An approach, to deal with performance shortfall by adjusting the system design in the implementation options in terms of environment, is described in Chapter 8. The problem of adjusting the system design to meet the performance goal is addressed in chapter 8 with an example.

9.2 Future Work

9.2.1 LQN Component Filter

This thesis’ contribution for the “completion” is the development of LQN Component Filter tool with four variants. More communication filters can be designed and developed to support other types of communication mechanisms and components at LQN level.

At this time, LQN Component Filter tool does not have the capability to support activity based LQN models. It can be enhanced to support activity based LQN models.

9.2.2 Graphical User Interface (GUI) for budgeted CPU demands

A Graphic User Interface (GUI) can be designed to provide ease in changing CPU demands of different entries and activities of a LQN model. This will help a user to keep track of CPU demands of entries and activities of a LQN model for a satisfactory performance response. This might be helpful in a case where budget adjustments are required for a desired performance target.

9.2.3 Integrated Environment for UCM based Budget Analysis Road Map

This thesis described a UCM design based budget analysis road map. An integrated environment is required to provide automation for the UCM design based budget analysis road map. Different tools are already described in Chapter 2, which are available for the "completion" at UCM level as well as for additional "completion" at LQN level. Each tool supports a different type of the "completion". These tools do not provide integrated automation for the "completion" as well as budget analysis road map. Further research and development is required in this direction to combine and integrate all the tools to introduce different types of "completions" at UCM level as well as LQN level.

If all the available tools are combined into one integrated tool then it will increase ease of manipulation of designs and development. A short-term goal could be the integration of all these tools into UCM Navigator to provide an integrated environment for UCM based software designs.

9.2.4 Budget Analysis Road Map for Different Specifications

A UCM is not the only possible starting point for budget analysis road map. Another form of specification, which identifies the scenarios, the components, and the activities for which code is to be developed, could be used. For instance an executable state-machine specification in ObjecTime described in [Hrischuk95], SDL described in [El-Syed98], or a UML specification can be used. A profile for UML has recently been defined to support such an analysis [OMG01].

Further research and development is required to apply budget analysis road map on these specifications to exploit the benefits of this software performance engineering approach.

9.3 Summary

This chapter highlighted the conclusions of the research describing the contributions of the research. The future work suggested many issues which are as follows:

1. Further improvement of LQN Component filter,
2. Graphical User Interface (GUI) for budgeted CPU demands,
3. Integrated environment for UCM based Budget Analysis Road Map, and
4. Budget Analysis for different specifications such as UML and SDL etc.

References

[Aho98] A. Aho, S. Gallagher, N. Griffeth, C. Scheel, and D.Swayne, “Sculptor with Chisel: Requirements Engineering for Communication Services,” *Fifth International Workshop on Feature Interaction in Telecommunications and Software Systems (FIW’98)*, IOS Press, Amsterdam, pp. 45-63, October 1998.

[Amyot98] Daniel Amyot, “Use Case Maps for the Design and Validation of Interactions-Free Telephony Features,” CITO Report # 1430, Ottawa, 1998

[Amyot99a] D.Amyot and R. Andrade, “Description of Wireless Intelligent Network Service with Use Case Maps,” *17th Brazillian Symposium on Computer Networks (SBRC’99),* Salvador, Brazil, May 1999

[Amyot99b] D. Amyot, L. Logirippo, R.J.A. Buhr, and T. Gray, “Use Case Maps for the Capture and Validation of Distributed Systems Requirements,” *Fourth International Symposium on Requirements Engineering (RE’99),* Limerick, Ireland, June 1999

[Amyot00] D. Amyot, L. Charfi, N. Gorse, T. Gray, L. Logirippo, J. Sincennes, B. Stepien, and T. Ware, “Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS,” *Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW’00),* Glasgow, Scotland, May 2000

[Auerbach94] J. Auerbach, S. Demion, G. Goldszmidt, J. R. Rao, and J. Russell, “Concert/C Rel 3.1: Availability of a language for distributed C programming,” Technical Report, IBM Research, July 1994

[Ananda91] A. L. Ananda, B. H. Tay, and E. K. Koh, “Astra, An asynchronous remote procedure call facility.” *In Proceedings of 11th International Conference on Distributed Computing Systems,* Arlington, Texas, pages 172-179, May 1991

[Arnold99a] K. Arnold. B, O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath, *The Jini[tm] Specification,* Addison-Wesley Press, June 1999

[Arnold99b] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces[tm] Principles, Patterns, and Practice*, Addison-Wesley Press, June 1999

[Arnold00] K. Arnold, *The Jini[tm] Specifications,* Addison-Wesley Press, 2nd Edition, December 2000

[Arons] B. Arons, “Tools for Building Asynchronous Servers to Support Speech and Audio Applications,” *In Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'92),* Monterey, California, USA, pages 71-78, November 15-18, 1992

[Buhr95] Buhr, R.J.A. and Casselman, R.S.: *Use Case Maps for Object-Oriented System*s, Prentice-Hall, USA, 1995.

[Buhr98] Buhr, R.J.A, “Use Case Maps as Architectural Entities for Complex Systems,” *In: Transactions on Software Engineering, IEEE,* Vol. 24, No. 12, pp. 1131-1155, December 1998.

[Cabri98] G. Cabri, L. Leonardi, and F. Zambonelli, “Reactive Tuple Spaces for Mobile Agent Coordination,*” 2nd International Workshop on Mobile Agents, Stuttgart (D), Lecture Notes in Computer Science*, No. 1477, pp. 237-248, Sept. 1998.

[Czarnecki01] K. Czarnecki and U. W. Eisenecker, *Generative Programming*. Addison-Wesley, 2001

[Dasgupta98] P. Dasgupta, “Network Operating Systems,*” A comprehensive article in Encyclopedia of Electrical Engineering,* John Wiley, Department of Computer Science and Engineering, Arizona State University, Tempe AZ 85287-5406, USA, 1998.

[Edwards00] W. K. Edwards, *Core Jini[tm]*, Prentice-Hall Press, 2nd Edition, December 2000

[El-Sayed98] H. El-Sayed, D. Cameron, C. M. Woodside, "Automated Performance Modeling from Scenarios and SDL Designs of Telecom Systems," *In Proc. of the Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE98),* Kyoto, April 1998.

[Foster95] I. Foster, *Designing and Building Parallel Programs*, Addison-Wesley Press, Department of Computer Science, University of Chicago, Illinois, USA, 1995

[Franks99] G. Franks, PhD., "Performance Analysis of Distributed Server Systems", Ph. D. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, December 1999.

[Gelerntner97] A. Smith, D. Gelerntner and N. Carriero, “Towards Wide-Area Network Piranha: Implementing Java-Linda,” Department of Computer Science, Yale University, New Haven, USA, Technical Report, November 1997.

[Halang91] W. Halang and A. Stoyenko, *Constructing Predictable Real-Time Systems*, Kluwer Academic Publishers, Boston, 1991

[Hrischuk95] C. Hrischuk, J. Rolia and C.M. Woodside, "Automatic Generation of a Software Performance Model Using an Object-Oriented Prototype," *Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 399-409, Durham, NC, January 1995.

[IEEE-610.12] Institute of Electrical and Electronic Engineers. "IEEE Standard Glossary of Software Engineering Terminology." IEEE Standards Collection. New York, NY: Institute of Electrical and Electronics Engineers, 1993.

[Jain91] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling,* Wiley Publisher, New York, NY, April 1991.

[JSpaces] J-Spaces™ Platform 1.0 documentation
http://www.j-spaces.com/

[Kawai88] S. Kawai and S. Matsuoka., "Using Tuple Space Communication in Distributed Object Oriented Languages," *Proc. of the OOPSLA '88,* pages 276-283, 1988.

[Lee01] K. C. Lee, "Software Modeling Tools and Studies," Technical Report, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, April 2001

[Matloff99] N. Matloff, "TupleDSM: An Educational Tool for Software Distributed Shared Memory," *Workshop on Computer Architecture Education (WCAE-99),* Orlando, Florida, USA, January 9-13, 1999

[Mayers00] N. Mayers, *Java Programming on Linux,* Waite Group Press, 2000

[McMullan00] D. McMullan, "Components in Layered Queuing Networks," Technical Report, Department of Systems and Computer Engineering, Carleton University, Canada, April 2000

[McNamara99] C. McNamara, Business Report on "Basic Guide to Non-Profit Financial Management", 1999

[Miga98] A. Miga, “Application of Use Case Maps to System Design with Tool Support,” M. Eng. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, July 1998

[Mowbray97] T. J. Mowbray, W. A. Ruth, *Inside CORBA: Distributed Object Standards and Applications*, Published by Addison Wesley Longman Inc, 1997

[Mussbacher01] G. Mussbacher and D. Amyot, *A Collection of Patterns for Use Case Maps – 3rd Draft*, Mitel Networks, 350 Legget Dr., Kanata (ON), Canada, K2K 2W7, PRIVATE COMMUNICATION

[Oaks00] S. Oaks and H. Wong, *Jini in a Nutshell*, O'Reilly & Associates Publisher, March 2000

[OMG95] Object Management Group, *Common Object Request Broker Architecture and Specification (CORBA)*, Revision 2, New York,: John Wiley, August 1995

[OMG01] OMG document "UML Profile for Schedulability, Performance, and Time," Revised submission, June 2001, available from OMG at www.omg.org.

[Peine97] H. Peine, T. Stolpmann, “The Architecture of the Ara platform for Mobile Agents,” *Proceedings of the Ist International Workshop on Mobile Agents, Berlin (D), Lecture Notes in Computer Science*, No. 1219, Springer-Vaerlag(D), pp, 50-61, April 1997

[Petriu01a] D. B. Petriu, “Layered Software Performance Models Constructed from Use Case Map Specifications,” M. Eng. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 2001.

[Petriu01b] D. B. Petriu and C.M. Woodside, “Generating A Performance Model From A Design Specification,” *The 6th Mitel Workshop (MICON2001)*, Mitel Networks, Ottawa, August 2001

[Roberts92] John Roberts, *Introduction to Parallel Processing and the Transputer*, published by Van Nostrand Reinhold, New York, U. S. A, 1992, ISBN 0-442-00872-4

[Rolia95] J.A. Rolia and K.C. Seveik, “The Method of Layers,” *IEEE Trans. on Software Engineering*, vol. 21, no.8, August 1995, pp.689-700.

[Siddiqui00] K H. Siddiqui and C.M. Woodside, “A Description of Time/Perfomrance Budgeting for UCM Desgin,” *The 5th Mitel Workshop (MICON2000)*, Mitel Networks, Ottawa, August 2000

[Siddiqui01] K H. Siddiqui and C.M. Woodside, "Performance Aware Software Development (PASD) Using Execution Time Budgets," *The 6th Mitel Workshop (MICON2001)*, Mitel Networks, Ottawa, August 2001

[Spruit97] C. M.C. Spruit, L. P. J. Groenewegen and I. G. Sprinkhuizen-Kuyper, "Blackboard Systems modeled in SOCCA", Technical Report, Department of Computer Science, Leiden University, The Netherlands, 1997

[Smith93] C.U. Smith and L.G. Williams, "Software Performance Engineering" A case study including performance comparison with design alternatives." *IEEE Transactions on Software Engineering*, 19(7): 720-741, July 1993.

[Smith95] Smith, C. U. & Williams, L. G., "A Performance Model Interchange Format," Performance Engineering Services, Feb. 1995.

[Smith01] Smith, C. U. & Williams, L. G., Performance *Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Addison-Wesley Publishing Company, 2001.

[Szyperski98] C. Szyperski, *Component Software; Beyond Object-Oriented Programming*. New York: ACM Press, 1998

[UCMorg] Use Case Maps Web Page and UCM User Group (since March 1999) http://www.UseCaseMaps.org

[Waldo98] Waldo, J., JavaSpaces^{TM} Specification 1.0, SUN Microsystems Inc., Technical Report, March 1998

[Walkner95] David W. Walker, "An Introduction To Message Passing Paradigms," *In Proceedings of the 1995 CERN School of Computing*, CERN 95-05, pages 165-184, ed. C. E. Vandoni, held August 20 - September 2, 1995, in Arles, France, 1995.

[Woodside89] C.M.Woodside, "Throughput calculation for basic stochastic rendezvous networks," *Performance Evaluation*, Vol. 9, No. 2, pp.143-160, 1989

[Woodside95] C.M. Woodside J.E. Neilson, S. Majumdar, and D. C. Petriu, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software," *IEEE Trans. on Computers*, vol. 44, no. 1, January 1995, pp 20-34.

[Woodside95c] C.M. Woodside, R.G. Franks, A. Hubbard, S. Majumdar, J.E. Neilson, D. C. Petriu, and J.A. Rolia, "A Toolset for Performance Engineering and Software Design of Client-Server Systems," *Performance Evaluation*, vol-24, no 1-2, 1995, pp.117-136.

[Woodside95b] C.M.Woodside and G. Ragunath, “General Bypass Architecture for High- Performance Distributed Algorithms,” *Proc. 6th IFIP Conference on Performance of Computer Networks*, Istanbul, Oct.23-26, 1995, *In “Data Communication and their Performance”*, eds. S. Fdida and R.U. Onvural, Chapman and Hall, pp51-65,

[Woodside98] C.M. Woodside, C. Hrischuk, B. Selic, S. Bayarov, “A Wideband Approach to Integrating Performance Prediction into a Software Design Environment,” *Proc. First Int. Workshop on Software and Performance (WOSP98),* pp. 31-41, October 1998.

[Woodside99] C.M. Woodside and C. U. Smith, "Performance Validation at Early Stages of Development", Position paper, Performance 99, Istanbul, Turkey, October 99

[Woodside01] C. M. Woodside, A. Miga, D. Amyot, F. Bordeleau, and D. Cameron, “Deriving Message Sequence Charts from Use Case Maps Scenario Specifications”, *In Tenth SDL Forum (SDL'01)*, Copenhagen, Denmark, June 2001.

APPENDIX A – Simple Call Agent LQN File

```
# Khalid H. Siddiqui
# Student #256847
# August 05. 2000
# File name: sample.xlqn
# *** For 1 20 20 Name Service User ****
# *** For 1 to 200 Remote Users ****
# <khs@markab> spex -F mat sample.xlqn

G "Call Agent Model: Name Service, Two Users Call Connection, first
phases only" .000001 100 1 0.5 -1

$nRemoteUser= 1:200,10
$nNSUser=1:20,4
$hosts = sunset,sunspot,sunrise,markab,homefree


P 7
p NSUserProc f  i #infinite processors (one per user)
p RemoteUserProc f i #infinite processors (one per user)
p WorkLoadUserProc f i #infinite processors (one per user)
p NodeA f
p NodeB f
p CompetingWL f
p Network f i
p Disk f
-1

T 15
t NSUser r NSUser -1 NSUserProc m $nNSUser
t RemoteUser r RemoteUser -1 RemoteUserProc m  $nRemoteUser %f
$RemoteUserThruput
t WLUserA r WLUserA -1 WorkLoadUserProc m 10
t WLUserB r WLUserB -1 WorkLoadUserProc m 10
t CompetingWLA n CompetingWLA -1 CompetingWL
t CompetingWLB n CompetingWLB -1 CompetingWL
t CallAgentA n CallAgentAL CallAgentAR -1 NodeA m 2
t ProtocolAgentA n ProtocolAgentA -1 NodeA
t CResourceA n  CResourceA -1 NodeA
t NameServeice n  NameService -1 NodeA
t CallAgentB n CallAgentB -1 NodeB m 2
t ProtocolAgentB n ProtocolAgentB -1 NodeB
t CResourceB n  CResourceB -1 NodeB
t NetworkDelay n NetworkDelay -1 Network
t Disk n Disk -1 Disk

-1

E 37
s NSUser 1 0 0 -1 # NS User
Z NSUser 10 0 0 -1
s RemoteUser 1 0 0 -1 %s $RemoteUserEntryTime # Remote User
Z RemoteUser 20 0 0 -1
s WLUserA 1 0 0 -1 # WorkLoad User A
```

```
Z WLUserA 10 0 0 -1
s WLUserB 1 0 0 -1 # WorkLoad User B
Z WLUserB 10 0 0 -1
y NSUser CallAgentAL 1 0 0 -1
s CallAgentAL 1 0 0 -1
y CallAgentAL NameService 1 0 0 -1
s NameService 1 0 0 -1
y NameService Disk 1 0 0 -1
s Disk 1 0 0 -1
y RemoteUser CallAgentAR  1 0 0 -1
s CallAgentAR 1 0 0 -1
y CallAgentAR NetworkDelay 1 0 0 -1
s NetworkDelay 1 0 0 -1
y CallAgentAR ProtocolAgentA 1 0 0 -1
s ProtocolAgentA 1 0 0 -1
y ProtocolAgentA ProtocolAgentB 1 0 0 -1
s ProtocolAgentB 1 0 0 -1
y ProtocolAgentB CallAgentB 1 0 0 -1
s CallAgentB 1 0 0 -1
y WLUserA CompetingWLA  1 0 0 -1
y CallAgentAR CResourceA 1 0 0 -1
y CompetingWLA  CResourceA 1 0 0 -1
y ProtocolAgentA  CResourceA 1 0 0 -1
s CompetingWLA 1 0 0 -1
s CResourceA 1 0 0 -1
y WLUserB CompetingWLB  1 0 0 -1
y CallAgentB CResourceB 1 0 0 -1
y CompetingWLB  CResourceB 1 0 0 -1
y ProtocolAgentB  CResourceB 1 0 0 -1
s CompetingWLB 1 0 0 -1
s CResourceB 1 0 0 -1
-1

R 0
$0 = $nRemoteUser
$RemoteUserThruput
$RemoteUserResponseTime  = $RemoteUserEntryTime - 20
```

-1

APPENDIX B – J-Spaces LQN File

```
# Khalid H. Siddiqui
# Student #256847
# Sept 14, 2001
# File name: JSPacesModelv1012.xlqn
# Number of Workers =6
G "JSpaces Model: Master, Worker, Prime Demo Exmaple" .000001 1000 1
0.5 -1

$solver=parasrvn
$nUser= 1:15,1

P 0
p jP f %u $Putil
-1

T 0
t RefTask1 r RefTask1 -1  jP
t Master n Mwrite Mtake Mrmid Mlus Mhttpd  -1 jP m $nUser  %f $Thruput
t Workers1 n Wwrite1 Wread1 Wrmid1 Wlus1 Whttpd1 -1 jP
t Workers2 n Wwrite2 Wread2 Wrmid2 Wlus2 Whttpd2 -1 jP
t Workers3 n Wwrite3 Wread3 Wrmid3 Wlus3 Whttpd3 -1 jP
t Workers4 n Wwrite4 Wread4 Wrmid4 Wlus4 Whttpd4 -1 jP
t Workers5 n Wwrite5 Wread5 Wrmid5 Wlus5 Whttpd5 -1 jP
t Workers6 n Wwrite6 Wread6 Wrmid6 Wlus6 Whttpd6 -1 jP
t Workers7 n Wwrite7 Wread7 Wrmid7 Wlus7 Whttpd7 -1 jP
t Workers8 n Wwrite8 Wread8 Wrmid8 Wlus8 Whttpd8 -1 jP
t Workers9 n Wwrite9 Wread9 Wrmid9 Wlus9 Whttpd9 -1 jP
t Workers10 n Wwrite10 Wread10 Wrmid10 Wlus10 Whttpd10 -1 jP
t Workers11 n Wwrite11 Wread11 Wrmid11 Wlus11 Whttpd11 -1 jP
t Workers12 n Wwrite12 Wread12 Wrmid12 Wlus12 Whttpd12 -1 jP
t RMID n eRMID -1 jP
t TSM n  eTSM -1 jP
t LUS n eLUS -1 jP
t HTTPD n eHTTPD -1 jP
t JSpaces n JSread JSwrite -1 jP
-1

E 0
s RefTask1 1.000000 -1  %s $UserEntryTime
Z RefTask1 100 -1
s Mwrite 3.92 0 0 -1
s Mtake 3.92 0 0 -1
s Wwrite1 3.92 0 0 -1
s Wread1 3.92 0 0 -1
s Wwrite2 3.92 0 0 -1
s Wread2 3.92 0 0 -1
s Wwrite3 3.92 0 0 -1
s Wread3 3.92 0 0 -1
s Wwrite4 3.92 0 0 -1
s Wread4 3.92 0 0 -1
s Wwrite5 3.92 0 0 -1
s Wread5 3.92 0 0 -1
```

```
s Wwrite6 3.92 0 0 -1
s Wread6 3.92 0 0 -1
s Wwrite7 3.92 0 0 -1
s Wread7 3.92 0 0 -1
s Wwrite8 3.92 0 0 -1
s Wread8 3.92 0 0 -1
s Wwrite9 3.92 0 0 -1
s Wread9 3.92 0 0 -1
s Wwrite10 3.92 0 0 -1
s Wread10 3.92 0 0 -1
s Wwrite11 3.92 0 0 -1
s Wread11 3.92 0 0 -1
s Wwrite12 3.92 0 0 -1
s Wread12 3.92 0 0 -1
s eRMID 131.5 0 0 -1
s eLUS 31.83 0 0 -1
s eTSM 8.33 0 0 -1
s eHTTPD 56.42 0 0 -1
s JSread 3.92 0 0 -1
s JSwrite 3.92 0 0 -1
s Mrmid 1 0 0 -1
s Wrmid1 1 0 0 -1
s Wrmid2 1 0 0 -1
s Wrmid3 1 0 0 -1
s Wrmid4 1 0 0 -1
s Wrmid5 1 0 0 -1
s Wrmid6 1 0 0 -1
s Wrmid7 1 0 0 -1
s Wrmid8 1 0 0 -1
s Wrmid9 1 0 0 -1
s Wrmid10 1 0 0 -1
s Wrmid11 1 0 0 -1
s Wrmid12 1 0 0 -1
s Mlus 1 0 0 -1
s Wlus1 1 0 0 -1
s Wlus2 1 0 0 -1
s Wlus3 1 0 0 -1
s Wlus4 1 0 0 -1
s Wlus5 1 0 0 -1
s Wlus6 1 0 0 -1
s Wlus7 1 0 0 -1
s Wlus8 1 0 0 -1
s Wlus9 1 0 0 -1
s Wlus10 1 0 0 -1
s Wlus11 1 0 0 -1
s Wlus12 1 0 0 -1
s Mhttpd 1 0 0 -1
s Whttpd1 1 0 0 -1
s Whttpd2 1 0 0 -1
s Whttpd3 1 0 0 -1
s Whttpd4 1 0 0 -1
s Whttpd5 1 0 0 -1
s Whttpd6 1 0 0 -1
s Whttpd7 1 0 0 -1
s Whttpd8 1 0 0 -1
s Whttpd9 1 0 0 -1
```

```
s Whttpd10 1 0 0 -1
s Whttpd11 1 0 0 -1
s Whttpd12 1 0 0 -1
y RefTask1 Mwrite 1.000000 -1
y RefTask1 Mtake 1.000000 -1
y RefTask1 Wwrite1 1.000000 -1
y RefTask1 Wread1 1.000000 -1
y RefTask1 Wwrite2 1.000000 -1
y RefTask1 Wread2 1.000000 -1
y RefTask1 Wwrite3 1.000000 -1
y RefTask1 Wread3 1.000000 -1
y RefTask1 Wwrite4 1.000000 -1
y RefTask1 Wread4 1.000000 -1
y RefTask1 Wwrite5 1.000000 -1
y RefTask1 Wread5 1.000000 -1
y RefTask1 Wwrite6 1.000000 -1
y RefTask1 Wread6 1.000000 -1
y RefTask1 Wwrite7 1.000000 -1
y RefTask1 Wread7 1.000000 -1
y RefTask1 Wwrite8 1.000000 -1
y RefTask1 Wread8 1.000000 -1
y RefTask1 Wwrite9 1.000000 -1
y RefTask1 Wread9 1.000000 -1
y RefTask1 Wwrite10 1.000000 -1
y RefTask1 Wread10 1.000000 -1
y RefTask1 Wwrite11 1.000000 -1
y RefTask1 Wread11 1.000000 -1
y RefTask1 Wwrite12 1.000000 -1
y RefTask1 Wread12 1.000000 -1
y RefTask1 Mrmid 1.000000 -1
y RefTask1 Wrmid1 1.000000 -1
y RefTask1 Wrmid2 1.000000 -1
y RefTask1 Wrmid3 1.000000 -1
y RefTask1 Wrmid4 1.000000 -1
y RefTask1 Wrmid5 1.000000 -1
y RefTask1 Wrmid6 1.000000 -1
y RefTask1 Wrmid7 1.000000 -1
y RefTask1 Wrmid8 1.000000 -1
y RefTask1 Wrmid9 1.000000 -1
y RefTask1 Wrmid10 1.000000 -1
y RefTask1 Wrmid11 1.000000 -1
y RefTask1 Wrmid12 1.000000 -1
y RefTask1 Mlus 1.000000 -1
y RefTask1 Wlus1 1.000000 -1
y RefTask1 Wlus2 1.000000 -1
y RefTask1 Wlus3 1.000000 -1
y RefTask1 Wlus4 1.000000 -1
y RefTask1 Wlus5 1.000000 -1
y RefTask1 Wlus6 1.000000 -1
y RefTask1 Wlus7 1.000000 -1
y RefTask1 Wlus8 1.000000 -1
y RefTask1 Wlus9 1.000000 -1
y RefTask1 Wlus10 1.000000 -1
y RefTask1 Wlus11 1.000000 -1
y RefTask1 Wlus12 1.000000 -1
y RefTask1 Mhttpd 1.000000 -1
```

```
y RefTask1 Whttpd1 1.000000 -1
y RefTask1 Whttpd2 1.000000 -1
y RefTask1 Whttpd3 1.000000 -1
y RefTask1 Whttpd4 1.000000 -1
y RefTask1 Whttpd5 1.000000 -1
y RefTask1 Whttpd6 1.000000 -1
y RefTask1 Whttpd7 1.000000 -1
y RefTask1 Whttpd8 1.000000 -1
y RefTask1 Whttpd9 1.000000 -1
y RefTask1 Whttpd10 1.000000 -1
y RefTask1 Whttpd11 1.000000 -1
y RefTask1 Whttpd12 1.000000 -1
y Mwrite eTSM 1.000000 -1
y Mtake eTSM 1.000000 -1
y Wwrite1 eTSM 1.000000 -1
y Wread1 eTSM 1.000000 -1
y Wwrite2 eTSM 1.000000 -1
y Wread2 eTSM 1.000000 -1
y Wwrite3 eTSM 1.000000 -1
y Wread3 eTSM 1.000000 -1
y Wwrite4 eTSM 1.000000 -1
y Wread4 eTSM 1.000000 -1
y Wwrite5 eTSM 1.000000 -1
y Wread5 eTSM 1.000000 -1
y Wwrite6 eTSM 1.000000 -1
y Wread6 eTSM 1.000000 -1
y Wwrite7 eTSM 1.000000 -1
y Wread7 eTSM 1.000000 -1
y Wwrite8 eTSM 1.000000 -1
y Wread8 eTSM 1.000000 -1
y Wwrite9 eTSM 1.000000 -1
y Wread9 eTSM 1.000000 -1
y Wwrite10 eTSM 1.000000 -1
y Wread10 eTSM 1.000000 -1
y Wwrite11 eTSM 1.000000 -1
y Wread11 eTSM 1.000000 -1
y Wwrite12 eTSM 1.000000 -1
y Wread12 eTSM 1.000000 -1
y Mrmid eRMID 1.000000 -1
y Wrmid1 eRMID 1.000000 -1
y Wrmid2 eRMID 1.000000 -1
y Wrmid3 eRMID 1.000000 -1
y Wrmid4 eRMID 1.000000 -1
y Wrmid5 eRMID 1.000000 -1
y Wrmid6 eRMID 1.000000 -1
y Wrmid7 eRMID 1.000000 -1
y Wrmid8 eRMID 1.000000 -1
y Wrmid9 eRMID 1.000000 -1
y Wrmid10 eRMID 1.000000 -1
y Wrmid11 eRMID 1.000000 -1
y Wrmid12 eRMID 1.000000 -1
y Mlus eLUS 1.000000 -1
y Wlus1 eLUS 1.000000 -1
y Wlus2 eLUS 1.000000 -1
y Wlus3 eLUS 1.000000 -1
y Wlus4 eLUS 1.000000 -1
```

```
y Wlus5 eLUS 1.000000 -1
y Wlus6 eLUS 1.000000 -1
y Wlus7 eLUS 1.000000 -1
y Wlus8 eLUS 1.000000 -1
y Wlus9 eLUS 1.000000 -1
y Wlus10 eLUS 1.000000 -1
y Wlus11 eLUS 1.000000 -1
y Wlus12 eLUS 1.000000 -1
y Mhttpd eHTTPD 1.000000 -1
y Whttpd1 eHTTPD 1.000000 -1
y Whttpd2 eHTTPD 1.000000 -1
y Whttpd3 eHTTPD 1.000000 -1
y Whttpd4 eHTTPD 1.000000 -1
y Whttpd5 eHTTPD 1.000000 -1
y Whttpd6 eHTTPD 1.000000 -1
y Whttpd7 eHTTPD 1.000000 -1
y Whttpd8 eHTTPD 1.000000 -1
y Whttpd9 eHTTPD 1.000000 -1
y Whttpd10 eHTTPD 1.000000 -1
y Whttpd11 eHTTPD 1.000000 -1
y Whttpd12 eHTTPD 1.000000 -1
y eTSM JSread 1.000000 -1
y eTSM JSwrite 1.000000 -1
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime
$Putil
-1
```

APPENDIX C1 – FDMC(m=1) LQN File

```
G
"        Comment: ''
        TSW101.xlqn
        Author: Khalid H. Siddiqui
        Model with
        -100% Probabilities for all the users
        - Think Time: Z= 1000 ms
        - Main Controller Threads: m=1
        July 29, 2001
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100000 -m logjul28
$nUser= 1:300,30

P 8
p PartyA_P f
p PartyB_P f
p TupleSpace_P f
p Device_A f
p Device_B f
p OS_P f
p Process_AB f
p Backup_P f
-1

T 12
t Party_AUser r Party_AUser -1 PartyA_P m $nUser %f $Thruput
t Party_A f ptyA_tae  -1 PartyA_P m $nUser
t Party_B f ptyB_e11  -1 PartyB_P m $nUser
t Device_A f devA_e11 devA_e12 -1 Device_A m $nUser
t Main_Controller f mc_e11 mc_tae -1 OS_P
t Process_A f psA_e11 psA_e12 -1 Process_AB
t Process_B f psB_e11 psB_e12 -1 Process_AB
t Process_Backup f psBk_e11 -1 Backup_P
t Device_B f devB_e11 -1 Device_B
t TS_CriticalSection f ts_cs  -1 TupleSpace_P
t TS_psA_psB f ts_psA_psB -1 TupleSpace_P
t TS_psB_psBk f ts_psB_psBk -1 TupleSpace_P
-1

E 16
s Party_AUser 1 0 0 -1 %s $UserEntryTime
Z Party_AUser 1000 0 0 -1
s ptyB_e11 1.0 0.0 0.0 -1
s devA_e11 1.0 0.0 0.0 -1
s mc_e11 1.0 0.0 0.0 -1
s devB_e11 1.0 0.0 0.0 -1
s devA_e12 1.0 0.0 0.0 -1
```

```
s ts_cs  1.0 0.0 0.0 -1
s ts_psA_psB 1.0 0.0 0.0 -1
s ts_psB_psBk 1.0 0.0 0.0 -1
s psA_e11 1.0 0.0 0.0 -1
s psA_e12 1.0 0.0 0.0 -1
s psB_e11 1.0 0.0 0.0 -1
s psB_e12 1.0 0.0 0.0 -1
s psBk_e11 1.0 0.0 0.0 -1
y Party_AUser ptyA_tae 1 0 0 -1
A ptyA_tae ptyA_e11
A mc_tae mc_e12
F devA_e11 mc_e11 1.0 -1
F mc_e11 psA_e11 1.0 -1
z devA_e12 mc_tae 1.0 0.0 0.0 -1
F psA_e12 ts_psA_psB 1.0 -1
F ts_psA_psB psB_e11 1.0 -1
F psB_e11 ptyB_e11 1.0 -1
F ptyB_e11 devB_e11 1.0 -1
y psB_e12 ts_psB_psBk 1.0 0.0 0.0 -1
y ts_psB_psBk psBk_e11 1.0 0.0 0.0 -1
y psB_e11 ptyB_e11 1.0 0.0 0.0 -1
y ts_psA_psB ts_cs 1.0 0.0 0.0 -1
y ts_psB_psBk ts_cs 1.0 0.0 0.0 -1
-1

A Party_A
s ptyA_e11 1.0
y ptyA_e11 devA_e11 1.0

s ptyA_e12 1.0
z ptyA_e12 devA_e12 1.0

:
ptyA_e11 ->  ptyA_e12;
ptyA_e12[ptyA_tae]
-1

A Main_Controller
s mc_e12 1.0
y mc_e12 psA_e12 1.0

s mc_e13 1.0
z mc_e13 psB_e12 1.0

:
mc_e12 ->  mc_e13;
mc_e13[mc_tae]
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime

-1
```

APPENDIX C2 – MC(0.05, 0.15, 0.15) LQN File

```
G
"        Comment: ''
        TSW107.xlqn
        Author: Khalid H. Siddiqui
        Model with
        -100% Probabilities for all the users
        - Think Time: Z= 1000 ms
        - Main Controller Threads: m=10
         -Changes in demands of MainController 0.05, 0.15 and 0.15
        July 29, 2001
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100000 -m logjul28
$nUser= 1:300,30


P 8
p PartyA_P f
p PartyB_P f
p TupleSpace_P f
p Device_A f
p Device_B f
p OS_P f
p Process_AB f
p Backup_P f
-1

T 12
t Party_AUser r Party_AUser -1 PartyA_P m $nUser %f $Thruput
t Party_A f ptyA_tae  -1 PartyA_P m $nUser
t Party_B f ptyB_e11  -1 PartyB_P m $nUser
t Device_A f devA_e11 devA_e12 -1 Device_A m $nUser
t Main_Controller f mc_e11 mc_tae -1 OS_P m 10
t Process_A f psA_e11 psA_e12 -1 Process_AB
t Process_B f psB_e11 psB_e12 -1 Process_AB
t Process_Backup f psBk_e11 -1 Backup_P
t Device_B f devB_e11 -1 Device_B
t TS_CriticalSection f ts_cs  -1 TupleSpace_P
t TS_psA_psB f ts_psA_psB -1 TupleSpace_P
t TS_psB_psBk f ts_psB_psBk -1 TupleSpace_P
-1

E 16
s Party_AUser 1 0 0 -1 %s $UserEntryTime
Z Party_AUser 1000 0 0 -1
s ptyB_e11 1.0 0.0 0.0 -1
s devA_e11 1.0 0.0 0.0 -1
s mc_e11 0.05 0.0 0.0 -1
```

```
s devB_e11 1.0 0.0 0.0 -1
s devA_e12 1.0 0.0 0.0 -1
s ts_cs  1.0 0.0 0.0 -1
s ts_psA_psB 1.0 0.0 0.0 -1
s ts_psB_psBk 1.0 0.0 0.0 -1
s psA_e11 1.0 0.0 0.0 -1
s psA_e12 1.0 0.0 0.0 -1
s psB_e11 1.0 0.0 0.0 -1
s psB_e12 1.0 0.0 0.0 -1
s psBk_e11 1.0 0.0 0.0 -1
y Party_AUser ptyA_tae 1 0 0 -1
A ptyA_tae ptyA_e11
A mc_tae mc_e12
F devA_e11 mc_e11 1.0 -1
F mc_e11 psA_e11 1.0 -1
z devA_e12 mc_tae 1.0 0.0 0.0 -1
F psA_e12 ts_psA_psB 1.0 -1
F ts_psA_psB psB_e11 1.0 -1
F psB_e11 ptyB_e11 1.0 -1
F ptyB_e11 devB_e11 1.0 -1
y psB_e12 ts_psB_psBk 1.0 0.0 0.0 -1
y ts_psB_psBk psBk_e11 1.0 0.0 0.0 -1
y psB_e11 ptyB_e11 1.0 0.0 0.0 -1
y ts_psA_psB ts_cs 1.0 0.0 0.0 -1
y ts_psB_psBk ts_cs 1.0 0.0 0.0 -1
-1

A Party_A
s ptyA_e11 1.0
y ptyA_e11 devA_e11 1.0

s ptyA_e12 1.0
z ptyA_e12 devA_e12 1.0

:
ptyA_e11 ->  ptyA_e12;
ptyA_e12[ptyA_tae]
-1

A Main_Controller
s mc_e12 0.15
y mc_e12 psA_e12 1.0

s mc_e13 0.15
z mc_e13 psB_e12 1.0
:
mc_e12 ->  mc_e13;
mc_e13[mc_tae]
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime

-1
```

APPENDIX C3 – DVD(0.1, 0.1) LQN File

```
G
"        Comment: ''
        TSW108.xlqn
        Author: Khalid H. Siddiqui
        Model with
        -100% Probabilities for all the users
        - Think Time: Z= 1000 ms
        - Main Controller Threads: m=10
         -Changes in demands of MainController 0.05, 0.15 and 0.15
         -Changes in demands of Device A and Device B 0.1, 0.1
        July 29, 2001
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100000 -m logjul28
$nUser= 1:300,30

P 8
p PartyA_P f
p PartyB_P f
p TupleSpace_P f
p Device_A f
p Device_B f
p OS_P f
p Process_AB f
p Backup_P f
-1

T 12
t Party_AUser r Party_AUser -1 PartyA_P m $nUser %f $Thruput
t Party_A f ptyA_tae  -1 PartyA_P m $nUser
t Party_B f ptyB_e11  -1 PartyB_P m $nUser
t Device_A f devA_e11 devA_e12 -1 Device_A m $nUser
t Main_Controller f mc_e11 mc_tae -1 OS_P m 10
t Process_A f psA_e11 psA_e12 -1 Process_AB
t Process_B f psB_e11 psB_e12 -1 Process_AB
t Process_Backup f psBk_e11 -1 Backup_P
t Device_B f devB_e11 -1 Device_B
t TS_CriticalSection f ts_cs  -1 TupleSpace_P
t TS_psA_psB f ts_psA_psB -1 TupleSpace_P
t TS_psB_psBk f ts_psB_psBk -1 TupleSpace_P
-1

E 16
s Party_AUser 1 0 0 -1 %s $UserEntryTime
Z Party_AUser 1000 0 0 -1
s ptyB_e11 1.0 0.0 0.0 -1
s devA_e11 0.1 0.0 0.0 -1
s mc_e11 0.05 0.0 0.0 -1
```

```
s devB_e11 0.1 0.0 0.0 -1
s devA_e12 0.1 0.0 0.0 -1
s ts_cs  1.0 0.0 0.0 -1
s ts_psA_psB 1.0 0.0 0.0 -1
s ts_psB_psBk 1.0 0.0 0.0 -1
s psA_e11 1.0 0.0 0.0 -1
s psA_e12 1.0 0.0 0.0 -1
s psB_e11 1.0 0.0 0.0 -1
s psB_e12 1.0 0.0 0.0 -1
s psBk_e11 1.0 0.0 0.0 -1
y Party_AUser ptyA_tae 1 0 0 -1
A ptyA_tae ptyA_e11
A mc_tae mc_e12
F devA_e11 mc_e11 1.0 -1
F mc_e11 psA_e11 1.0 -1
z devA_e12 mc_tae 1.0 0.0 0.0 -1
F psA_e12 ts_psA_psB 1.0 -1
F ts_psA_psB psB_e11 1.0 -1
F psB_e11 ptyB_e11 1.0 -1
F ptyB_e11 devB_e11 1.0 -1
y psB_e12 ts_psB_psBk 1.0 0.0 0.0 -1
y ts_psB_psBk psBk_e11 1.0 0.0 0.0 -1
y psB_e11 ptyB_e11 1.0 0.0 0.0 -1
y ts_psA_psB ts_cs 1.0 0.0 0.0 -1
y ts_psB_psBk ts_cs 1.0 0.0 0.0 -1
-1

A Party_A
s ptyA_e11 1.0
y ptyA_e11 devA_e11 1.0

s ptyA_e12 1.0
z ptyA_e12 devA_e12 1.0

:
ptyA_e11 ->  ptyA_e12;
ptyA_e12[ptyA_tae]
-1

A Main_Controller
s mc_e12 0.15
y mc_e12 psA_e12 1.0

s mc_e13 0.15
z mc_e13 psB_e12 1.0

:
mc_e12 ->  mc_e13;
mc_e13[mc_tae]
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime
-1
```

APPENDIX C4 – PABD (0.1, 0.1) LQN File

```
G
"        Comment: ''
        TSW109.xlqn
        Author: Khalid H. Siddiqui
        Model with
        -100% Probabilities for all the users
        - Think Time: Z= 1000 ms
        - Main Controller Threads: m=10
         -Changes in demands of MainController 0.05, 0.15 and 0.15
         -Changes in demands of Device A and Device B 0.1, 0.1
         -Changes in demands of Process A and Process B 0.1, 0.1
        July 29, 2001
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100000 -m logjul28
$nUser= 1:300,30


P 8
p PartyA_P f
p PartyB_P f
p TupleSpace_P f
p Device_A f
p Device_B f
p OS_P f
p Process_AB f
p Backup_P f
-1

T 12
t Party_AUser r Party_AUser -1 PartyA_P m $nUser %f $Thruput
t Party_A f ptyA_tae  -1 PartyA_P m $nUser
t Party_B f ptyB_e11  -1 PartyB_P m $nUser
t Device_A f devA_e11 devA_e12 -1 Device_A m $nUser
t Main_Controller f mc_e11 mc_tae -1 OS_P m 10
t Process_A f psA_e11 psA_e12 -1 Process_AB
t Process_B f psB_e11 psB_e12 -1 Process_AB
t Process_Backup f psBk_e11 -1 Backup_P
t Device_B f devB_e11 -1 Device_B
t TS_CriticalSection f ts_cs  -1 TupleSpace_P
t TS_psA_psB f ts_psA_psB -1 TupleSpace_P
t TS_psB_psBk f ts_psB_psBk -1 TupleSpace_P
-1

E 16
s Party_AUser 1 0 0 -1 %s $UserEntryTime
Z Party_AUser 1000 0 0 -1
s ptyB_e11 1.0 0.0 0.0 -1
```

```
s devA_e11 0.1 0.0 0.0 -1
s mc_e11 0.05 0.0 0.0 -1
s devB_e11 0.1 0.0 0.0 -1
s devA_e12 0.1 0.0 0.0 -1
s ts_cs  1.0 0.0 0.0 -1
s ts_psA_psB 1.0 0.0 0.0 -1
s ts_psB_psBk 1.0 0.0 0.0 -1
s psA_e11 0.1 0.0 0.0 -1
s psA_e12 0.1 0.0 0.0 -1
s psB_e11 0.1 0.0 0.0 -1
s psB_e12 0.1 0.0 0.0 -1
s psBk_e11 1.0 0.0 0.0 -1
y Party_AUser ptyA_tae 1 0 0 -1
A ptyA_tae ptyA_e11
A mc_tae mc_e12
F devA_e11 mc_e11 1.0 -1
F mc_e11 psA_e11 1.0 -1
z devA_e12 mc_tae 1.0 0.0 0.0 -1
F psA_e12 ts_psA_psB 1.0 -1
F ts_psA_psB psB_e11 1.0 -1
F psB_e11 ptyB_e11 1.0 -1
F ptyB_e11 devB_e11 1.0 -1
y psB_e12 ts_psB_psBk 1.0 0.0 0.0 -1
y ts_psB_psBk psBk_e11 1.0 0.0 0.0 -1
y psB_e11 ptyB_e11 1.0 0.0 0.0 -1
y ts_psA_psB ts_cs 1.0 0.0 0.0 -1
y ts_psB_psBk ts_cs 1.0 0.0 0.0 -1
-1

A Party_A
s ptyA_e11 1.0
y ptyA_e11 devA_e11 1.0

s ptyA_e12 1.0
z ptyA_e12 devA_e12 1.0
:
ptyA_e11 ->  ptyA_e12;
ptyA_e12[ptyA_tae]
-1

A Main_Controller
s mc_e12 0.15
y mc_e12 psA_e12 1.0
s mc_e13 0.15
z mc_e13 psB_e12 1.0
:
mc_e12 ->  mc_e13;
mc_e13[mc_tae]
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime

-1
```

APPENDIX C5 – PAB (m = user) LQN File

```
G
"        Comment: ''
        TSW110.xlqn
        Author: Khalid H. Siddiqui
        Model with
        -100% Probabilities for all the users
        - Think Time: Z= 1000 ms
        - Main Controller Threads: m=10
         -Changes in demands of MainController 0.05, 0.15 and 0.15
         -Changes in demands of Device A and Device B 0.1, 0.1
         -Changes in demands of Process A and Process B 0.1, 0.1 With
Multiple copies
        July 29, 2001
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100000 -m logjul28
$nUser= 1:300,30

P 8
p PartyA_P f
p PartyB_P f
p TupleSpace_P f
p Device_A f
p Device_B f
p OS_P f
p Process_AB f
p Backup_P f
-1

T 12
t Party_AUser r Party_AUser -1 PartyA_P m $nUser %f $Thruput
t Party_A f ptyA_tae  -1 PartyA_P m $nUser
t Party_B f ptyB_e11  -1 PartyB_P m $nUser
t Device_A f devA_e11 devA_e12 -1 Device_A m $nUser
t Main_Controller f mc_e11 mc_tae -1 OS_P m 10
t Process_A f psA_e11 psA_e12 -1 Process_AB m $nUser
t Process_B f psB_e11 psB_e12 -1 Process_AB  m $nUser
t Process_Backup f psBk_e11 -1 Backup_P
t Device_B f devB_e11 -1 Device_B
t TS_CriticalSection f ts_cs  -1 TupleSpace_P
t TS_psA_psB f ts_psA_psB -1 TupleSpace_P
t TS_psB_psBk f ts_psB_psBk -1 TupleSpace_P
-1

E 16
s Party_AUser 1 0 0 -1 %s $UserEntryTime
Z Party_AUser 1000 0 0 -1
s ptyB_e11 1.0 0.0 0.0 -1
```

```
s devA_e11 0.1 0.0 0.0 -1
s mc_e11 0.05 0.0 0.0 -1
s devB_e11 0.1 0.0 0.0 -1
s devA_e12 0.1 0.0 0.0 -1
s ts_cs  1.0 0.0 0.0 -1
s ts_psA_psB 1.0 0.0 0.0 -1
s ts_psB_psBk 1.0 0.0 0.0 -1
s psA_e11 0.1 0.0 0.0 -1
s psA_e12 0.1 0.0 0.0 -1
s psB_e11 0.1 0.0 0.0 -1
s psB_e12 0.1 0.0 0.0 -1
s psBk_e11 1.0 0.0 0.0 -1
y Party_AUser ptyA_tae 1 0 0 -1
A ptyA_tae ptyA_e11
A mc_tae mc_e12
F devA_e11 mc_e11 1.0 -1
F mc_e11 psA_e11 1.0 -1
z devA_e12 mc_tae 1.0 0.0 0.0 -1
F psA_e12 ts_psA_psB 1.0 -1
F ts_psA_psB psB_e11 1.0 -1
F psB_e11 ptyB_e11 1.0 -1
F ptyB_e11 devB_e11 1.0 -1
y psB_e12 ts_psB_psBk 1.0 0.0 0.0 -1
y ts_psB_psBk psBk_e11 1.0 0.0 0.0 -1
y psB_e11 ptyB_e11 1.0 0.0 0.0 -1
y ts_psA_psB ts_cs 1.0 0.0 0.0 -1
y ts_psB_psBk ts_cs 1.0 0.0 0.0 -1
-1

A Party_A
s ptyA_e11 1.0
y ptyA_e11 devA_e11 1.0

s ptyA_e12 1.0
z ptyA_e12 devA_e12 1.0
:
ptyA_e11 ->  ptyA_e12;
ptyA_e12[ptyA_tae]
-1

A Main_Controller
s mc_e12 0.15
y mc_e12 psA_e12 1.0
s mc_e13 0.15
z mc_e13 psB_e12 1.0
:
mc_e12 ->  mc_e13;
mc_e13[mc_tae]
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime

-1
```

APPENDIX D1 – Switch (CPUd =0.25) LQN File

```
# UCM2LQN output
G
"        Author: Khalid H. Siddiqui
        - 100% Probabilities for all the users
        - CPU demands for Switch 0.25
        - Common Processor for Switch and CPU
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100
$nUser= 50:500,50

P 0
p p1 f %u $OrigUtil
p p2 f %u $SwitchUtil
p p3 f %u $TermUtil
-1

T 0
t RefTask1 r RefTask1 -1  p1 m $nUser  %f $Thruput
t Orig f Orig_E1 Orig_E2 -1  p1 m $nUser
t Switch f Switch_E1 Switch_E2a Switch_E2b Switch_E3 -1 p2
t OS f OS_E1 -1  p2
t Term f Term_E1a Term_E1b Term_E2 -1  p3 m $nUser
-1

E 0
A Switch_E1 Switch_A1
A Switch_E2a Switch_A3
A Switch_E2b Switch_A4
A Switch_E3 Switch_A17
A Orig_E1 Orig_A2
A Orig_E2 Orig_A11
A Term_E1a Term_A1a
A Term_E1b Term_A1b
A Term_E2 Term_A10
A OS_E1 OS_A1
s RefTask1 1.000000  -1 %s $UserEntryTime
Z RefTask1 1000 -1
y RefTask1 Orig_E1 1.000000 -1
-1

A Switch
f Switch_A1 1
s Switch_A1 0.25000000
f Switch_A3 1
s Switch_A3 0.25000000
f Switch_A4 1
s Switch_A4 0.25000000
```

```
f Switch_A6 1
s Switch_A6 0.25000000
f Switch_A8 1
s Switch_A8 0.25000000
f Switch_A8a 1
s Switch_A8a 0.25000000
y Switch_A8a Term_E1a 1.000000
f Switch_A8b 1
s Switch_A8b 0.25000000
y Switch_A8b Term_E1b 1.000000
f Switch_A9 1
s Switch_A9 0.25000000
z Switch_A9 Orig_E2 1.000000
f Switch_A13 1
s Switch_A13 0.25000000
z Switch_A13 OS_E1 1.000000
f Switch_A15 1
s Switch_A15 0.25000000
f Switch_A17 1
s Switch_A17 0.25000000
f Switch_A19 1
s Switch_A19 0.25000000
z Switch_A19 Term_E2 1.000000
:
Switch_A1[Switch_E1];
Switch_A3 -> Switch_A6;
Switch_A6[Switch_E2a];
Switch_A4 -> Switch_A8;
Switch_A8 -> Switch_A8a & Switch_A8b;
Switch_A8a & Switch_A8b -> Switch_A9;
Switch_A9 -> Switch_A13 & Switch_A15 & Switch_A19;
Switch_A15[Switch_E2b];
Switch_A17[Switch_E3]
-1

A Orig
f Orig_A2 1
s Orig_A2 1.000000
y Orig_A2 Switch_E1 1.000000
f Orig_A4 1
s Orig_A4 1.000000
f Orig_A4a 1
s Orig_A4a 1.000000
y Orig_A4a Switch_E2a 1.000000
f Orig_A4b 1
s Orig_A4b 1.000000
y Orig_A4b Switch_E2b 1.000000
f Orig_A5 1
s Orig_A5 1.000000
f Orig_A7 1
s Orig_A7 1.000000
f Orig_A9 1
s Orig_A9 1.000000
f Orig_A11 1
s Orig_A11 1.000000
y Orig_A11 Switch_E3 1.000000
```

```
:
Orig_A2 -> Orig_A4;
Orig_A4 -> (0.500000) Orig_A4a + (0.500000) Orig_A4b;
Orig_A4a -> Orig_A5;
Orig_A4b -> Orig_A7;
Orig_A7 -> Orig_A9;
Orig_A9 -> Orig_A11;
Orig_A11[Orig_E1]
-1

A Term
f Term_A1a 1
s Term_A1a 1.000000
f Term_A1b 1
s Term_A1b 1.000000
f Term_A3 1
s Term_A3 1.000000
f Term_A5 1
s Term_A5 1.000000
f Term_A8 1
s Term_A8 1.000000
f Term_A9 1
s Term_A9 1.000000
f Term_A10 1
s Term_A10 1.000000
f Term_A11 1
s Term_A11 1.000000
:
Term_A1a -> Term_A5 & Term_A8;
Term_A1b -> Term_A3;
Term_A3[Term_E1b];
Term_A5 & Term_A8 -> Term_A9;
Term_A9[Term_E1a];
Term_A10 -> Term_A11
-1

A OS
f OS_A1 1
s OS_A1 1.000000
f LogBegin_ABA_t_hid33 1
s LogBegin_ABA_t_hid33 1.000000
f OS_A2 1
s OS_A2 1.000000
:
OS_A1 -> LogBegin_ABA_t_hid33;
LogBegin_ABA_t_hid33 -> OS_A2
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime
$OrigUtil
$SwitchUtil
$TermUtil
-1
```

APPENDIX D2 – Orig (CPUd= 0.5) LQN File

```
G
"       - 100% Probabilities for all the users
        - CPU demands for Switch 0.25
        - CPU demands for Orig 0.5
        - RefTask1=Orig=Term=nUser
        - Common Processor for Switch and CPU
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100
$nUser= 50:500,50

P 0
p p1 f %u $OrigUtil
p p2 f %u $SwitchUtil
p p3 f %u $TermUtil
-1

T 0
t RefTask1 r RefTask1 -1  p1 m $nUser  %f $Thruput
t Orig f Orig_E1 Orig_E2 -1  p1 m $nUser
t Switch f Switch_E1 Switch_E2a Switch_E2b Switch_E3 -1 p2
t OS f OS_E1 -1  p2
t Term f Term_E1a Term_E1b Term_E2 -1  p3 m $nUser
-1

E 0
A Switch_E1 Switch_A1
A Switch_E2a Switch_A3
A Switch_E2b Switch_A4
A Switch_E3 Switch_A17
A Orig_E1 Orig_A2
A Orig_E2 Orig_A11
A Term_E1a Term_A1a
A Term_E1b Term_A1b
A Term_E2 Term_A10
A OS_E1 OS_A1
s RefTask1 1.000000  -1 %s $UserEntryTime
Z RefTask1 1000 -1
y RefTask1 Orig_E1 1.000000 -1
-1

A Switch
f Switch_A1 1
s Switch_A1 0.25000000
f Switch_A3 1
s Switch_A3 0.25000000
f Switch_A4 1
s Switch_A4 0.25000000
f Switch_A6 1
```

```
s Switch_A6 0.25000000
f Switch_A8 1
s Switch_A8 0.25000000
f Switch_A8a 1
s Switch_A8a 0.25000000
y Switch_A8a Term_E1a 1.000000
f Switch_A8b 1
s Switch_A8b 0.25000000
y Switch_A8b Term_E1b 1.000000
f Switch_A9 1
s Switch_A9 0.25000000
z Switch_A9 Orig_E2 1.000000
f Switch_A13 1
s Switch_A13 0.25000000
z Switch_A13 OS_E1 1.000000
f Switch_A15 1
s Switch_A15 0.25000000
f Switch_A17 1
s Switch_A17 0.25000000
f Switch_A19 1
s Switch_A19 0.25000000
z Switch_A19 Term_E2 1.000000
:
Switch_A1[Switch_E1];
Switch_A3 -> Switch_A6;
Switch_A6[Switch_E2a];
Switch_A4 -> Switch_A8;
Switch_A8 -> Switch_A8a & Switch_A8b;
Switch_A8a & Switch_A8b -> Switch_A9;
Switch_A9 -> Switch_A13 & Switch_A15 & Switch_A19;
Switch_A15[Switch_E2b];
Switch_A17[Switch_E3]
-1

A Orig
f Orig_A2 1
s Orig_A2 0.5000000
y Orig_A2 Switch_E1 1.000000
f Orig_A4 1
s Orig_A4 0.5000000
f Orig_A4a 1
s Orig_A4a 0.5000000
y Orig_A4a Switch_E2a 1.000000
f Orig_A4b 1
s Orig_A4b 0.5000000
y Orig_A4b Switch_E2b 1.000000
f Orig_A5 1
s Orig_A5 0.5000000
f Orig_A7 1
s Orig_A7 0.5000000
f Orig_A9 1
s Orig_A9 0.5000000
f Orig_A11 1
s Orig_A11 0.5000000
y Orig_A11 Switch_E3 1.000000
:
```

```
Orig_A2 -> Orig_A4;
Orig_A4 -> (0.500000) Orig_A4a + (0.500000) Orig_A4b;
Orig_A4a -> Orig_A5;
Orig_A4b -> Orig_A7;
Orig_A7 -> Orig_A9;
Orig_A9 -> Orig_A11;
Orig_A11[Orig_E1]
-1

A Term
f Term_A1a 1
s Term_A1a 1.000000
f Term_A1b 1
s Term_A1b 1.000000
f Term_A3 1
s Term_A3 1.000000
f Term_A5 1
s Term_A5 1.000000
f Term_A8 1
s Term_A8 1.000000
f Term_A9 1
s Term_A9 1.000000
f Term_A10 1
s Term_A10 1.000000
f Term_A11 1
s Term_A11 1.000000
:
Term_A1a -> Term_A5 & Term_A8;
Term_A1b -> Term_A3;
Term_A3[Term_E1b];
Term_A5 & Term_A8 -> Term_A9;
Term_A9[Term_E1a];
Term_A10 -> Term_A11
-1

A OS
f OS_A1 1
s OS_A1 1.000000
f LogBegin_ABA_t_hid33 1
s LogBegin_ABA_t_hid33 1.000000
f OS_A2 1
s OS_A2 1.000000
:
OS_A1 -> LogBegin_ABA_t_hid33;
LogBegin_ABA_t_hid33 -> OS_A2
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime
$OrigUtil
$SwitchUtil
$TermUtil

-1
```

APPENDIX D3 – Term (CPUd=0.1) LQN File

```
G
"
        - CPU demands for Switch 0.25
        - CPU demands for Orig 0.5
        - CPU demands for Term 0.1
       "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100
$nUser= 50:500,50

P 0
p p1 f %u $OrigUtil
p p2 f %u $SwitchUtil
p p3 f %u $TermUtil
-1

T 0
t RefTask1 r RefTask1 -1  p1 m $nUser  %f $Thruput
t Orig f Orig_E1 Orig_E2 -1  p1 m $nUser
t Switch f Switch_E1 Switch_E2a Switch_E2b Switch_E3 -1 p2
t OS f OS_E1 -1  p2
t Term f Term_E1a Term_E1b Term_E2 -1  p3 m $nUser
-1

E 0
A Switch_E1 Switch_A1
A Switch_E2a Switch_A3
A Switch_E2b Switch_A4
A Switch_E3 Switch_A17
A Orig_E1 Orig_A2
A Orig_E2 Orig_A11
A Term_E1a Term_A1a
A Term_E1b Term_A1b
A Term_E2 Term_A10
A OS_E1 OS_A1
s RefTask1 1.000000  -1 %s $UserEntryTime
Z RefTask1 1000 -1
y RefTask1 Orig_E1 1.000000 -1
-1
A Switch
f Switch_A1 1
s Switch_A1 0.25000000
f Switch_A3 1
s Switch_A3 0.25000000
f Switch_A4 1
s Switch_A4 0.25000000
f Switch_A6 1
s Switch_A6 0.25000000
f Switch_A8 1
```

```
s Switch_A8 0.25000000
f Switch_A8a 1
s Switch_A8a 0.25000000
y Switch_A8a Term_E1a 1.000000
f Switch_A8b 1
s Switch_A8b 0.25000000
y Switch_A8b Term_E1b 1.000000
f Switch_A9 1
s Switch_A9 0.25000000
z Switch_A9 Orig_E2 1.000000
f Switch_A13 1
s Switch_A13 0.25000000
z Switch_A13 OS_E1 1.000000
f Switch_A15 1
s Switch_A15 0.25000000
f Switch_A17 1
s Switch_A17 0.25000000
f Switch_A19 1
s Switch_A19 0.25000000
z Switch_A19 Term_E2 1.000000
:
Switch_A1[Switch_E1];
Switch_A3 -> Switch_A6;
Switch_A6[Switch_E2a];
Switch_A4 -> Switch_A8;
Switch_A8 -> Switch_A8a & Switch_A8b;
Switch_A8a & Switch_A8b -> Switch_A9;
Switch_A9 -> Switch_A13 & Switch_A15 & Switch_A19;
Switch_A15[Switch_E2b];
Switch_A17[Switch_E3]
-1

A Orig
f Orig_A2 1
s Orig_A2 0.5000000
y Orig_A2 Switch_E1 1.000000
f Orig_A4 1
s Orig_A4 0.5000000
f Orig_A4a 1
s Orig_A4a 0.5000000
y Orig_A4a Switch_E2a 1.000000
f Orig_A4b 1
s Orig_A4b 0.5000000
y Orig_A4b Switch_E2b 1.000000
f Orig_A5 1
s Orig_A5 0.5000000
f Orig_A7 1
s Orig_A7 0.5000000
f Orig_A9 1
s Orig_A9 0.5000000
f Orig_A11 1
s Orig_A11 0.5000000
y Orig_A11 Switch_E3 1.000000
:
Orig_A2 -> Orig_A4;
Orig_A4 -> (0.500000) Orig_A4a + (0.500000) Orig_A4b;
```

```
Orig_A4a -> Orig_A5;
Orig_A4b -> Orig_A7;
Orig_A7 -> Orig_A9;
Orig_A9 -> Orig_A11;
Orig_A11[Orig_E1]
-1

A Term
f Term_A1a 1
s Term_A1a 0.1000000
f Term_A1b 1
s Term_A1b 0.1000000
f Term_A3 1
s Term_A3 0.1000000
f Term_A5 1
s Term_A5 0.1000000
f Term_A8 1
s Term_A8 0.1000000
f Term_A9 1
s Term_A9 0.1000000
f Term_A10 1
s Term_A10 0.1000000
f Term_A11 1
s Term_A11 0.1000000
:
Term_A1a -> Term_A5 & Term_A8;
Term_A1b -> Term_A3;
Term_A3[Term_E1b];
Term_A5 & Term_A8 -> Term_A9;
Term_A9[Term_E1a];
Term_A10 -> Term_A11
-1

A OS
f OS_A1 1
s OS_A1 1.000000
f LogBegin_ABA_t_hid33 1
s LogBegin_ABA_t_hid33 1.000000
f OS_A2 1
s OS_A2 1.000000
:
OS_A1 -> LogBegin_ABA_t_hid33;
LogBegin_ABA_t_hid33 -> OS_A2
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime
$OrigUtil
$SwitchUtil
$TermUtil

-1
```

APPENDIX E1 – Switch (CPUd =0.25) LQN File

```
G
"        - Common Processor for Switch and OS
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100
$nUser= 50:500,50

P 0
p p1 f %u $OrigUtil
p p2 f %u $SwitchUtil
p p3 f %u $TermUtil
p p4 f %u $OSUtil
-1

T 0
t RefTask1 r RefTask1 -1  p1 m $nUser  %f $Thruput
t Orig f Orig_E1 Orig_E2 -1  p1 m $nUser
t Switch f Switch_E1 Switch_E2a Switch_E2b Switch_E3 -1 p2
t OS f OS_E1 -1  p4
t Term f Term_E1a Term_E1b Term_E2 -1  p3 m $nUser
-1

E 0
A Switch_E1 Switch_A1
A Switch_E2a Switch_A3
A Switch_E2b Switch_A4
A Switch_E3 Switch_A17
A Orig_E1 Orig_A2
A Orig_E2 Orig_A11
A Term_E1a Term_A1a
A Term_E1b Term_A1b
A Term_E2 Term_A10
A OS_E1 OS_A1
s RefTask1 1.000000  -1 %s $UserEntryTime
Z RefTask1 1000 -1
y RefTask1 Orig_E1 1.000000 -1
-1

A Switch
f Switch_A1 1
s Switch_A1 0.25000000
f Switch_A3 1
s Switch_A3 0.25000000
f Switch_A4 1
s Switch_A4 0.25000000
f Switch_A6 1
s Switch_A6 0.25000000
f Switch_A8 1
s Switch_A8 0.25000000
```

```
f Switch_A8a 1
s Switch_A8a 0.25000000
y Switch_A8a Term_E1a 1.000000
f Switch_A8b 1
s Switch_A8b 0.25000000
y Switch_A8b Term_E1b 1.000000
f Switch_A9 1
s Switch_A9 0.25000000
z Switch_A9 Orig_E2 1.000000
f Switch_A13 1
s Switch_A13 0.25000000
z Switch_A13 OS_E1 1.000000
f Switch_A15 1
s Switch_A15 0.25000000
f Switch_A17 1
s Switch_A17 0.25000000
f Switch_A19 1
s Switch_A19 0.25000000
z Switch_A19 Term_E2 1.000000

:
Switch_A1[Switch_E1];
Switch_A3 -> Switch_A6;
Switch_A6[Switch_E2a];
Switch_A4 -> Switch_A8;
Switch_A8 -> Switch_A8a & Switch_A8b;
Switch_A8a & Switch_A8b -> Switch_A9;
Switch_A9 -> Switch_A13 & Switch_A15 & Switch_A19;
Switch_A15[Switch_E2b];
Switch_A17[Switch_E3]
-1

A Orig
f Orig_A2 1
s Orig_A2 1.000000
y Orig_A2 Switch_E1 1.000000
f Orig_A4 1
s Orig_A4 1.000000
f Orig_A4a 1
s Orig_A4a 1.000000
y Orig_A4a Switch_E2a 1.000000
f Orig_A4b 1
s Orig_A4b 1.000000
y Orig_A4b Switch_E2b 1.000000
f Orig_A5 1
s Orig_A5 1.000000
f Orig_A7 1
s Orig_A7 1.000000
f Orig_A9 1
s Orig_A9 1.000000
f Orig_A11 1
s Orig_A11 1.000000
y Orig_A11 Switch_E3 1.000000
:
Orig_A2 -> Orig_A4;
Orig_A4 -> (0.500000) Orig_A4a + (0.500000) Orig_A4b;
```

```
Orig_A4a -> Orig_A5;
Orig_A4b -> Orig_A7;
Orig_A7 -> Orig_A9;
Orig_A9 -> Orig_A11;
Orig_A11[Orig_E1]
-1

A Term
f Term_A1a 1
s Term_A1a 1.000000
f Term_A1b 1
s Term_A1b 1.000000
f Term_A3 1
s Term_A3 1.000000
f Term_A5 1
s Term_A5 1.000000
f Term_A8 1
s Term_A8 1.000000
f Term_A9 1
s Term_A9 1.000000
f Term_A10 1
s Term_A10 1.000000
f Term_A11 1
s Term_A11 1.000000
:
Term_A1a -> Term_A5 & Term_A8;
Term_A1b -> Term_A3;
Term_A3[Term_E1b];
Term_A5 & Term_A8 -> Term_A9;
Term_A9[Term_E1a];
Term_A10 -> Term_A11
-1

A OS
f OS_A1 1
s OS_A1 1.000000
f LogBegin_ABA_t_hid33 1
s LogBegin_ABA_t_hid33 1.000000
f OS_A2 1
s OS_A2 1.000000
:
OS_A1 -> LogBegin_ABA_t_hid33;
LogBegin_ABA_t_hid33 -> OS_A2
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime
$OrigUtil
$SwitchUtil
$TermUtil
$OSUtil
-1
```

APPENDIX E2 – Orig (CPUd= 0.5) LQN File

```
G
"       - CPU demands for Switch 0.25
        - CPU demands for Orig 0.5
        - Common Processor for Switch and OS
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100
$nUser= 50:500,50

P 0
p p1 f %u $OrigUtil
p p2 f %u $SwitchUtil
p p3 f %u $TermUtil
p p4 f %u $OSUtil
-1

T 0
t RefTask1 r RefTask1 -1  p1 m $nUser  %f $Thruput
t Orig f Orig_E1 Orig_E2 -1  p1 m $nUser
t Switch f Switch_E1 Switch_E2a Switch_E2b Switch_E3 -1 p2
t OS f OS_E1 -1  p4
t Term f Term_E1a Term_E1b Term_E2 -1  p3 m $nUser
-1

E 0
A Switch_E1 Switch_A1
A Switch_E2a Switch_A3
A Switch_E2b Switch_A4
A Switch_E3 Switch_A17
A Orig_E1 Orig_A2
A Orig_E2 Orig_A11
A Term_E1a Term_A1a
A Term_E1b Term_A1b
A Term_E2 Term_A10
A OS_E1 OS_A1
s RefTask1 1.000000  -1 %s $UserEntryTime
Z RefTask1 1000 -1
y RefTask1 Orig_E1 1.000000 -1
-1

A Switch
f Switch_A1 1
s Switch_A1 0.25000000
f Switch_A3 1
s Switch_A3 0.25000000
f Switch_A4 1
s Switch_A4 0.25000000
f Switch_A6 1
```

```
s Switch_A6 0.25000000
f Switch_A8 1
s Switch_A8 0.25000000
f Switch_A8a 1
s Switch_A8a 0.25000000
y Switch_A8a Term_E1a 1.000000
f Switch_A8b 1
s Switch_A8b 0.25000000
y Switch_A8b Term_E1b 1.000000
f Switch_A9 1
s Switch_A9 0.25000000
z Switch_A9 Orig_E2 1.000000
f Switch_A13 1
s Switch_A13 0.25000000
z Switch_A13 OS_E1 1.000000
f Switch_A15 1
s Switch_A15 0.25000000
f Switch_A17 1
s Switch_A17 0.25000000
f Switch_A19 1
s Switch_A19 0.25000000
z Switch_A19 Term_E2 1.000000
:
Switch_A1[Switch_E1];
Switch_A3 -> Switch_A6;
Switch_A6[Switch_E2a];
Switch_A4 -> Switch_A8;
Switch_A8 -> Switch_A8a & Switch_A8b;
Switch_A8a & Switch_A8b -> Switch_A9;
Switch_A9 -> Switch_A13 & Switch_A15 & Switch_A19;
Switch_A15[Switch_E2b];
Switch_A17[Switch_E3]
-1

A Orig
f Orig_A2 1
s Orig_A2 0.5000000
y Orig_A2 Switch_E1 1.000000
f Orig_A4 1
s Orig_A4 0.5000000
f Orig_A4a 1
s Orig_A4a 0.5000000
y Orig_A4a Switch_E2a 1.000000
f Orig_A4b 1
s Orig_A4b 0.5000000
y Orig_A4b Switch_E2b 1.000000
f Orig_A5 1
s Orig_A5 0.5000000
f Orig_A7 1
s Orig_A7 0.5000000
f Orig_A9 1
s Orig_A9 0.5000000
f Orig_A11 1
s Orig_A11 0.5000000
y Orig_A11 Switch_E3 1.000000
:
```

```
Orig_A2 -> Orig_A4;
Orig_A4 -> (0.500000) Orig_A4a + (0.500000) Orig_A4b;
Orig_A4a -> Orig_A5;
Orig_A4b -> Orig_A7;
Orig_A7 -> Orig_A9;
Orig_A9 -> Orig_A11;
Orig_A11[Orig_E1]
-1

A Term
f Term_A1a 1
s Term_A1a 1.000000
f Term_A1b 1
s Term_A1b 1.000000
f Term_A3 1
s Term_A3 1.000000
f Term_A5 1
s Term_A5 1.000000
f Term_A8 1
s Term_A8 1.000000
f Term_A9 1
s Term_A9 1.000000
f Term_A10 1
s Term_A10 1.000000
f Term_A11 1
s Term_A11 1.000000
:
Term_A1a -> Term_A5 & Term_A8;
Term_A1b -> Term_A3;
Term_A3[Term_E1b];
Term_A5 & Term_A8 -> Term_A9;
Term_A9[Term_E1a];
Term_A10 -> Term_A11
-1

A OS
f OS_A1 1
s OS_A1 1.000000
f LogBegin_ABA_t_hid33 1
s LogBegin_ABA_t_hid33 1.000000
f OS_A2 1
s OS_A2 1.000000
:
OS_A1 -> LogBegin_ABA_t_hid33;
LogBegin_ABA_t_hid33 -> OS_A2
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime
$OrigUtil
$SwitchUtil
$TermUtil
$OSUtil
-1
```

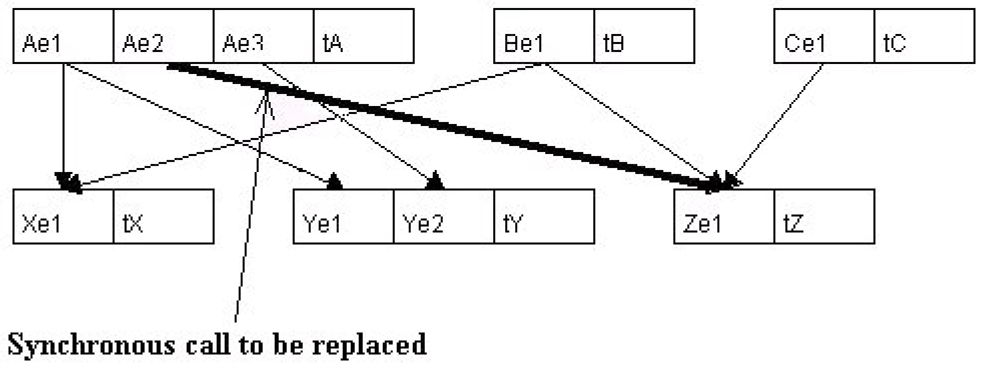
APPENDIX E3 – Term (CPUd=0.1) LQN File

```
# UCM2LQN output
G
"       - CPU demands for Switch 0.25
        - CPU demands for Orig 0.5
        - CPU demands for Term 0.1
        "
1.0E-5
50
1
0.5
-1

$solver = parasrvn -B 20,100
$nUser= 50:500,50

P 0
p p1 f %u $OrigUtil
p p2 f %u $SwitchUtil
p p3 f %u $TermUtil
p p4 f %u $OSUtil
-1

T 0
t RefTask1 r RefTask1 -1  p1 m $nUser  %f $Thruput
t Orig f Orig_E1 Orig_E2 -1  p1 m $nUser
t Switch f Switch_E1 Switch_E2a Switch_E2b Switch_E3 -1 p2
t OS f OS_E1 -1  p4
t Term f Term_E1a Term_E1b Term_E2 -1  p3 m $nUser
-1

E 0
A Switch_E1 Switch_A1
A Switch_E2a Switch_A3
A Switch_E2b Switch_A4
A Switch_E3 Switch_A17
A Orig_E1 Orig_A2
A Orig_E2 Orig_A11
A Term_E1a Term_A1a
A Term_E1b Term_A1b
A Term_E2 Term_A10
A OS_E1 OS_A1
s RefTask1 1.000000  -1 %s $UserEntryTime
Z RefTask1 1000 -1
y RefTask1 Orig_E1 1.000000 -1
-1
A Switch
f Switch_A1 1
s Switch_A1 0.25000000
f Switch_A3 1
s Switch_A3 0.25000000
f Switch_A4 1
s Switch_A4 0.25000000
f Switch_A6 1
s Switch_A6 0.25000000
```

```
f Switch_A8 1
s Switch_A8 0.25000000
f Switch_A8a 1
s Switch_A8a 0.25000000
y Switch_A8a Term_E1a 1.000000
f Switch_A8b 1
s Switch_A8b 0.25000000
y Switch_A8b Term_E1b 1.000000
f Switch_A9 1
s Switch_A9 0.25000000
z Switch_A9 Orig_E2 1.000000
f Switch_A13 1
s Switch_A13 0.25000000
z Switch_A13 OS_E1 1.000000
f Switch_A15 1
s Switch_A15 0.25000000
f Switch_A17 1
s Switch_A17 0.25000000
f Switch_A19 1
s Switch_A19 0.25000000
z Switch_A19 Term_E2 1.000000
:
Switch_A1[Switch_E1];
Switch_A3 -> Switch_A6;
Switch_A6[Switch_E2a];
Switch_A4 -> Switch_A8;
Switch_A8 -> Switch_A8a & Switch_A8b;
Switch_A8a & Switch_A8b -> Switch_A9;
Switch_A9 -> Switch_A13 & Switch_A15 & Switch_A19;
Switch_A15[Switch_E2b];
Switch_A17[Switch_E3]
-1

A Orig
f Orig_A2 1
s Orig_A2 0.5000000
y Orig_A2 Switch_E1 1.000000
f Orig_A4 1
s Orig_A4 0.5000000
f Orig_A4a 1
s Orig_A4a 0.5000000
y Orig_A4a Switch_E2a 1.000000
f Orig_A4b 1
s Orig_A4b 0.5000000
y Orig_A4b Switch_E2b 1.000000
f Orig_A5 1
s Orig_A5 0.5000000
f Orig_A7 1
s Orig_A7 0.5000000
f Orig_A9 1
s Orig_A9 0.5000000
f Orig_A11 1
s Orig_A11 0.5000000
y Orig_A11 Switch_E3 1.000000
:
Orig_A2 -> Orig_A4;
```

```
Orig_A4 -> (0.500000) Orig_A4a + (0.500000) Orig_A4b;
Orig_A4a -> Orig_A5;
Orig_A4b -> Orig_A7;
Orig_A7 -> Orig_A9;
Orig_A9 -> Orig_A11;
Orig_A11[Orig_E1]
-1

A Term
f Term_A1a 1
s Term_A1a 0.1000000
f Term_A1b 1
s Term_A1b 0.1000000
f Term_A3 1
s Term_A3 0.1000000
f Term_A5 1
s Term_A5 0.1000000
f Term_A8 1
s Term_A8 0.1000000
f Term_A9 1
s Term_A9 0.1000000
f Term_A10 1
s Term_A10 0.1000000
f Term_A11 1
s Term_A11 0.1000000
:
Term_A1a -> Term_A5 & Term_A8;
Term_A1b -> Term_A3;
Term_A3[Term_E1b];
Term_A5 & Term_A8 -> Term_A9;
Term_A9[Term_E1a];
Term_A10 -> Term_A11
-1

A OS
f OS_A1 1
s OS_A1 1.000000
f LogBegin_ABA_t_hid33 1
s LogBegin_ABA_t_hid33 1.000000
f OS_A2 1
s OS_A2 1.000000
:
OS_A1 -> LogBegin_ABA_t_hid33;
LogBegin_ABA_t_hid33 -> OS_A2
-1

R 0
$0 = $nUser
$Thruput
$ResponseTime = $UserEntryTime
$OrigUtil
$SwitchUtil
$TermUtil
$OSUtil

-1
```

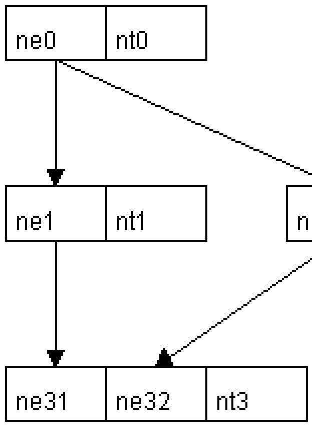
APPENDIX F1 – Synchronous Call Replacement



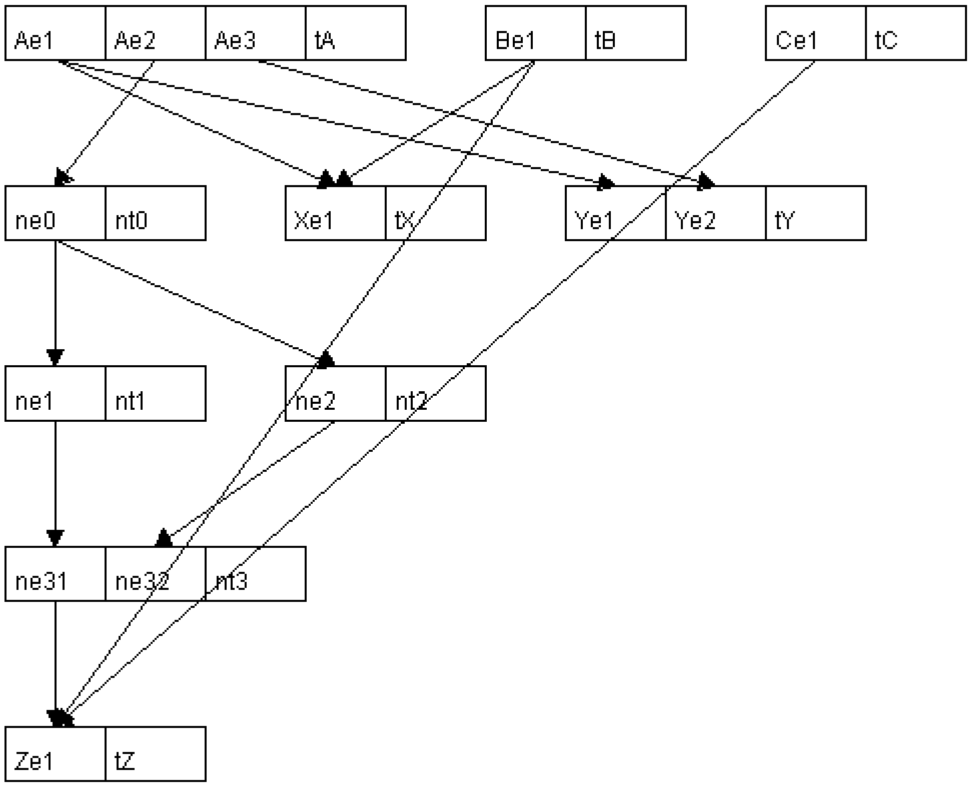
Input LQN Model

APPENDIX F2 – Synchronous Call Replacement Cases

Case S1:

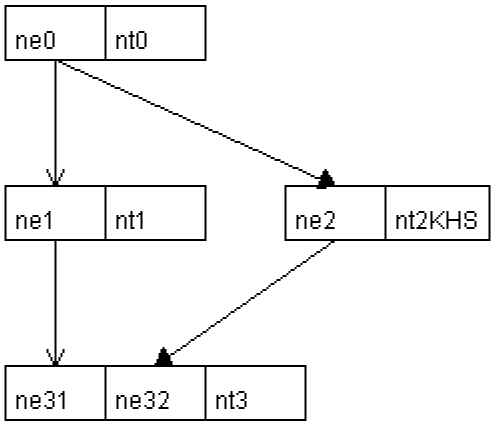


Network Subsystem

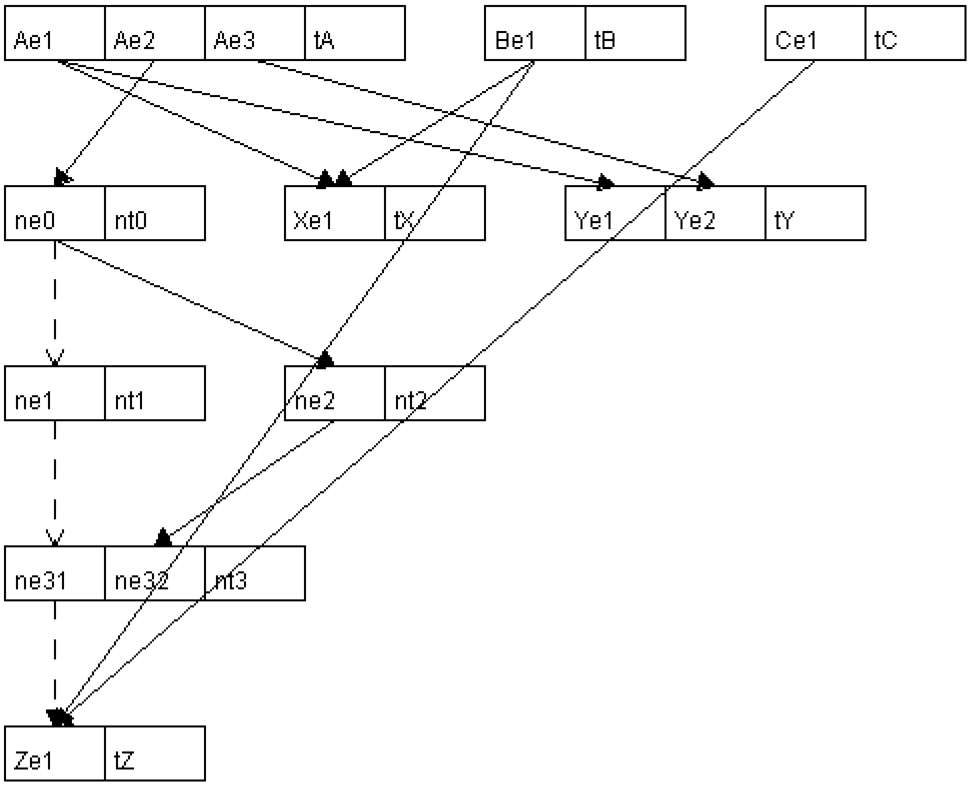


Output LQN Model

Case S2:

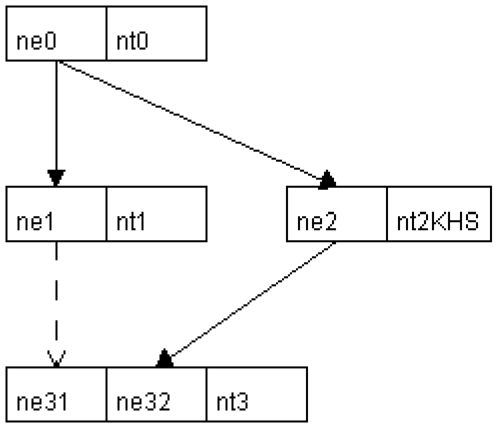


Network Subsystem

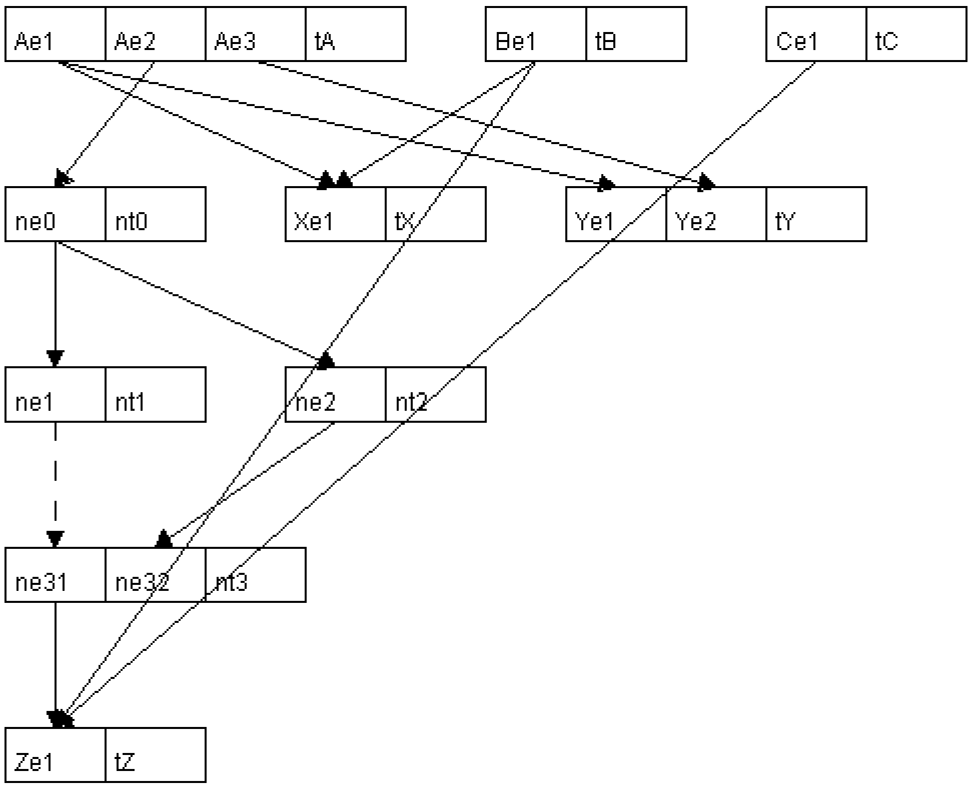


Output LQN Model

Case S3:

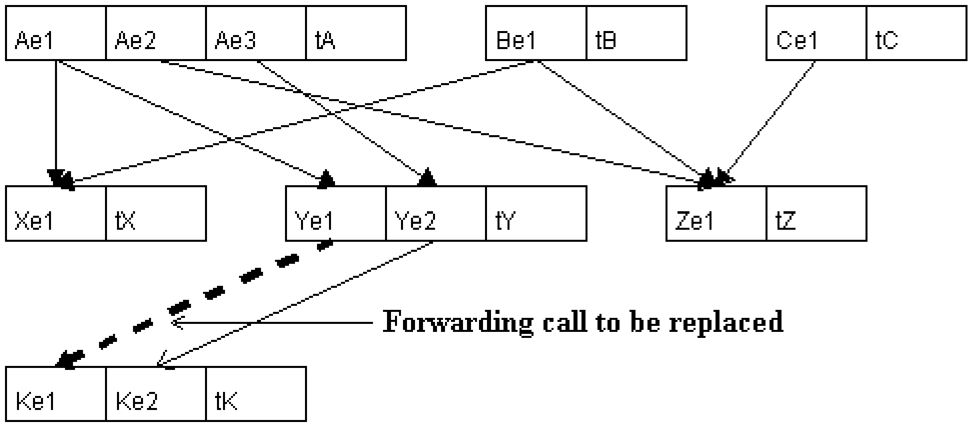


Network Subsystem



Output LQN Model

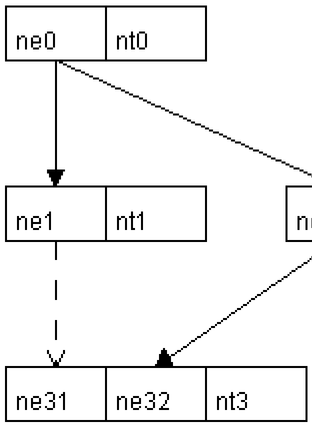
APPENDIX F3 – Forwarding Call Replacement Cases



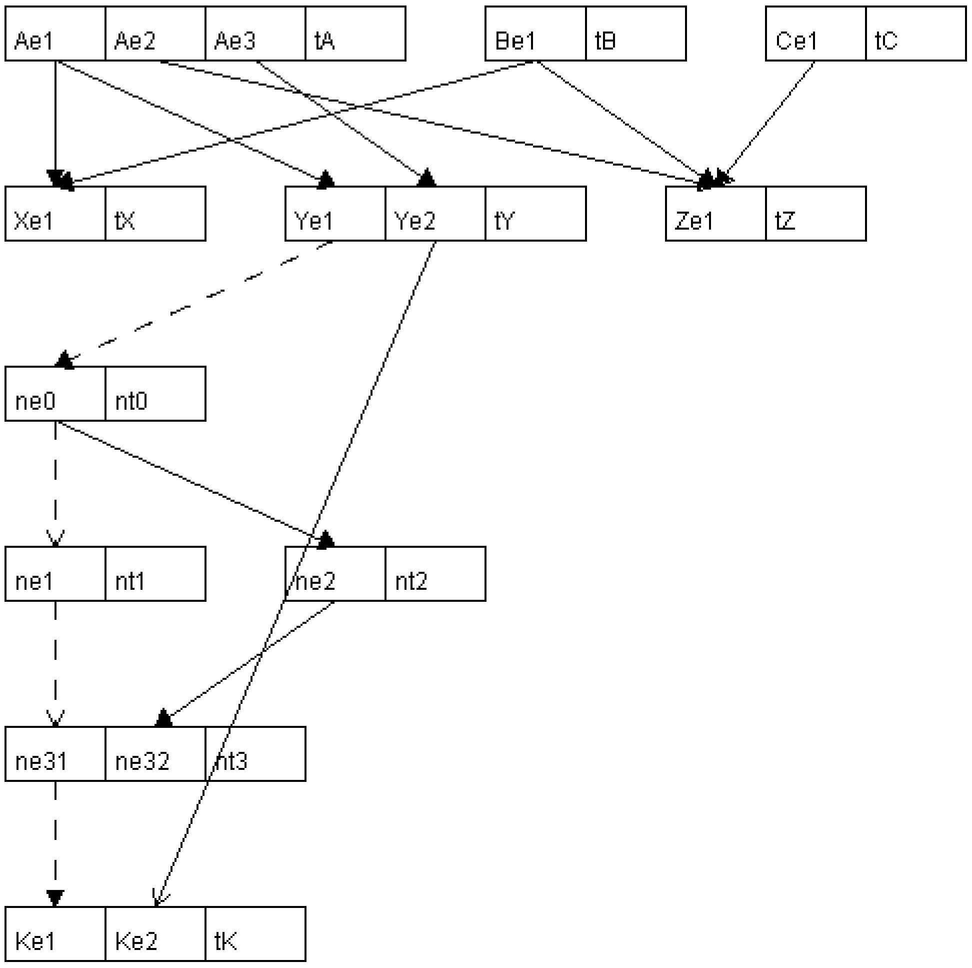
Input LQN Model

APPENDIX F4 – Forwarding Call Replacement Cases

Case F1:

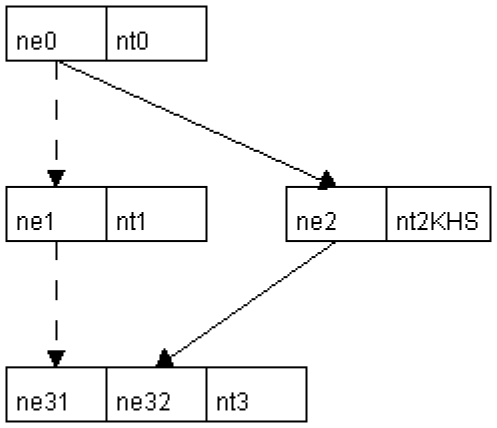


Network Subsystem

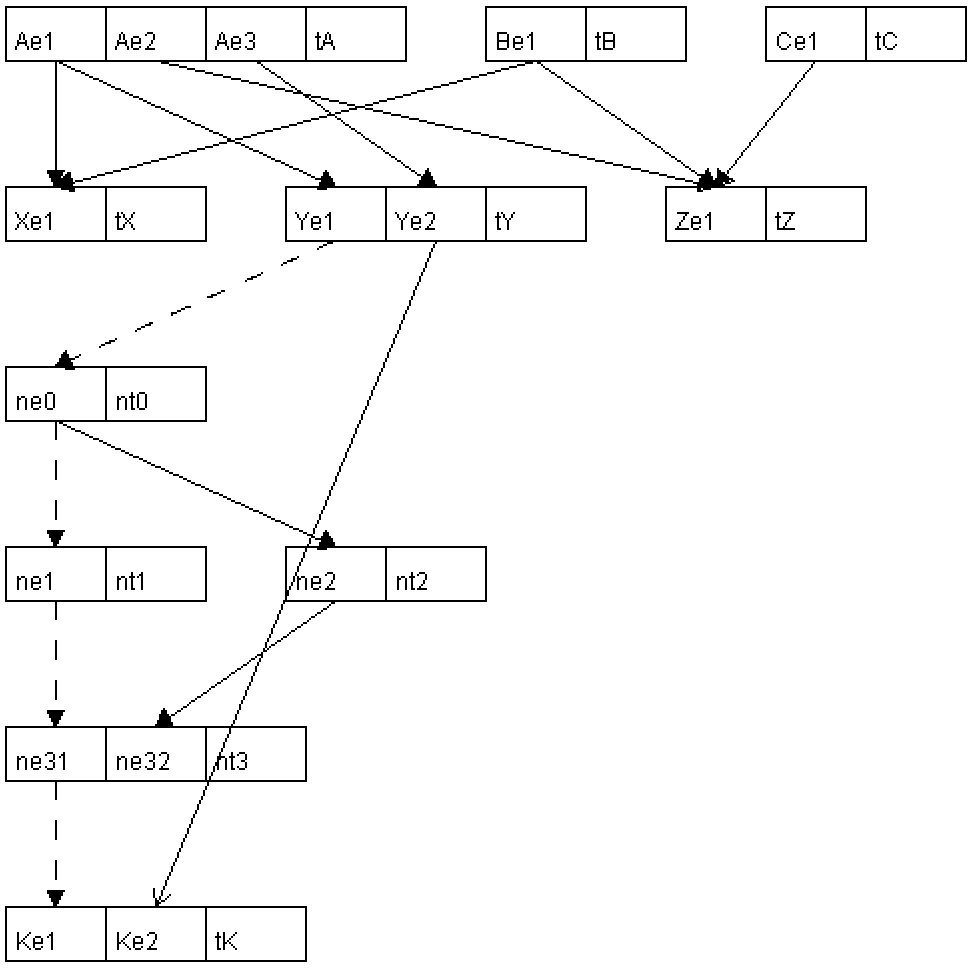


Output LQN Model

Case F2:



Network Subsystem



Output LQN Model

APPENDIX G1 – Input LQN File for Network Subsystem Filter

```
G
"Network Component Template File, Khalid H. Siddiqui"
1.0E-5
100
1
0.9
-1

P 6
p p1 f
p p2 f
p p3 f
p p4 f
p p5 f
p p6 f
-1

T 6
t tA r Ae1 Ae2 Ae3  -1 p1
t tB r Be1  -1 p2
t tX f Xe1  -1 p3
t tY f Ye1 Ye2  -1 p4
t tZ f Ze1  -1 p5
t tC f Ce1  -1 p6
-1

E 9
s Ae1 1.0 0.0 0.0 -1
y Ae1 Xe1 1.0 0.0 0.0 -1
y Ae1 Ye1 1.0 0.0 0.0 -1
s Ae2 1.0 0.0 0.0 -1
y Ae2 Ze1 1.0 0.0 0.0 -1
s Ae3 1.0 0.0 0.0 -1
y Ae3 Ye2 1.0 0.0 0.0 -1
s Be1 1.0 0.0 0.0 -1
y Be1 Xe1 1.0 0.0 0.0 -1
y Be1 Ze1 1.0 0.0 0.0 -1
s Xe1 1.0 0.0 0.0 -1
s Ye1 1.0 0.0 0.0 -1
s Ye2 1.0 0.0 0.0 -1
s Ze1 1.0 0.0 0.0 -1
s Ce1 1.0 0.0 0.0 -1
y Ce1 Ze1 1.0 0.0 0.0 -1
-1
```

APPENDIX G2 – TaskEntryFile.txt for Network Subsystem Filter

```
# Khalid H. Siddiqui
# Student #256847
# Sept. 10, 2001
# File name: EntryTaskData.txt
# Network Component Filter
# Usage: SourceEntry SourceTask DestinationEntry DestinationTask

Ae2 tA Ye2 tY

#end
-1
```

APPENDIX G3 – Network Subsystem LQN File

```
G
"Network Component Template File, Khalid H. Siddiqui"
1.0E-5
100
1
0.9
-1

P 4
p np0 f
p np1 f
p np2 f
p np3 f
-1

T 4
t nt0 r ne0  -1 np0
t nt1 f ne1  -1 np1
t nt2KHS f ne2  -1 np2
t nt3 f ne31 ne32  -1 np3
-1

E 5
s ne0 1.0 0.0 0.0 -1
y ne0 ne1 1.0 0.0 0.0 -1
y ne0 ne2 1.0 0.0 0.0 -1
s ne1 1.0 0.0 0.0 -1
y ne1 ne31 1.0 0.0 0.0 -1
s ne2 1.0 0.0 0.0 -1
y ne2 ne32 1.0 0.0 0.0 -1
s ne31 1.0 0.0 0.0 -1
s ne32 1.0 -1
-1
```

APPENDIX G4 – Output LQN File with Shared Processor P3

```
G
"Network Component Template File, Khalid H. Siddiqui"
1.0E-5
100
1
0.9
-1

P 0
p np0 f
p np1 f
p np3 f
p p1 f
p p2 f
p p3 f
p p4 f
p p5 f
p p6 f
-1

T 0
t nt0 n ne0  -1 np0
t nt1 n ne1  -1 np1
t nt2 n ne2  -1 p3
t nt3 n ne31 ne32  -1 np3
t tA r Ae1 Ae2 Ae3  -1 p1
t tB r Be1  -1 p2
t tC f Ce1  -1 p6
t tX f Xe1  -1 p3
t tY f Ye1 Ye2  -1 p4
t tZ f Ze1  -1 p5
-1

E 0
s Ae1 1.0 0.0 0.0 -1
y Ae1 Xe1 1.0 0.0 0.0 -1
y Ae1 Ye1 1.0 0.0 0.0 -1
s Ae2 1.0 0.0 0.0 -1
y Ae2 ne0 1.0 1.0 1.0 -1
s Ae3 1.0 0.0 0.0 -1
y Ae3 Ye2 1.0 0.0 0.0 -1
s Be1 1.0 0.0 0.0 -1
y Be1 Xe1 1.0 0.0 0.0 -1
y Be1 Ze1 1.0 0.0 0.0 -1
s Ce1 1.0 0.0 0.0 -1
y Ce1 Ze1 1.0 0.0 0.0 -1
s Xe1 1.0 0.0 0.0 -1
s Ye1 1.0 0.0 0.0 -1
s Ye2 1.0 0.0 0.0 -1
s Ze1 1.0 0.0 0.0 -1
s ne0 1.0 0.0 0.0 -1
y ne0 ne1 1.0 0.0 0.0 -1
```

```
y ne0 ne2 1.0 0.0 0.0 -1
s ne1 1.0 0.0 0.0 -1
y ne1 ne31 1.0 0.0 0.0 -1
s ne2 1.0 0.0 0.0 -1
y ne2 ne32 1.0 0.0 0.0 -1
s ne31 1.0 0.0 0.0 -1
y ne31 Ze1 1.0 1.0 1.0 -1
s ne32 1.0 -1
-1
```