Software Bottleneck Identification and Visualization using Layered Queuing Networks

by

Vino Shanmugarajah B. Eng

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

Master of Applied Science

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

May 12, 2006

Your file Votre référence
ISBN: 978-0-494-33666-3
Our file Notre référence
ISBN: 978-0-494-33666-3

Abstract

Software bottleneck identification and saturation breakdown analysis are key aspects of Software Performance Engineering (SPE). A Software bottleneck is caused by resource saturation. Gaining insight of a software bottleneck involves understanding the multiple causes for saturation and breaking down their contributions to the bottleneck.

Pinpointing software bottlenecks in a real distributed system can be a complicated and time consuming multi-stepped task. This thesis attempts to reduce the complexity and effort in identifying and visualizing Software Bottleneck and Saturation breakdown using Layered Queuing Networks. Visualization of Software Bottlenecks and Saturation points are crucial to SPE as it gives insights on how they migrate from one Layer to other.

This thesis extends and integrates number of existing methods and technologies to present (1) Saturation Breakdown algorithm, (2) XML-DOM parser for LQML/CBML, (3) algorithms and GUI for visualizing both bottleneck and saturation breakdown, and (4) a case study demonstrating this research work.

Acknowledgement

Firstly, I must thank my supervisor, Prof. Murray Woodside for his guidance and commitment through out this research.

Secondly, I would like to thank my wife, Sahunthala and my son, Shananthan for their support, encouragement and patience. I greatly appreciate every minutes of family time they sacrificed for this research.

Finally, I would like to thank my professional colleagues at Nortel Networks for their support and encouragement. Especially, I like to thank my team members from Optical Control and Topology (OTC) of Common Photonic Layer (CPL) Systems.

Table of Contents

List of Figures

Chapter 1. Introduction

Software Performance Engineering (SPE) is a vital part of the Software Development Cycle of any performance sensitive distributed system. It has been often stated that SPE must be done in the early stages of development cycle to achieve its full potential [1][2][3][4]. SPE is conducted based on performance models built to reflect the systems under considerations. These performance models can be built using the information from the software models that describe the functional qualities [5][6]. Although there are many performance modeling technologies [1][7][8][9][10][11] and languages available for SPE, the Layered Queuing Network (LQN) [12]is best suited for distributed systems which are inherently layered. LQN is an extension of proven Queuing Network concepts. However, LQN is also hampered by the need for special expertise in building performance models and interpreting its result. This thesis addresses the need to assist in creating LQN models and interpreting the results.

In LQN, processes are modeled as queuing network stations and the messages processed/passed between the processes are modeled as jobs. Once an appropriate model is built for a system, there are solvers (generally known as LQNS) that will provide "accurate enough" performance prediction. By analyzing the output results of LQNS, important performance insights such as Bottleneck identification and Saturation breakdown can be computed. A software bottleneck is a single congestion point that limits system level performance [13]. Once we identify the bottleneck, saturation breakdown of that point will give additional performance insights. Using bottleneck and saturation breakdown, an appropriate mitigation strategy can be proposed to reduce the bottleneck. Experts in this field believe that bottleneck cannot be eliminated in a system. It can be reduced or migrated to a different less critical point [12]. Thus, mitigation strategy must be multi-stepped recursive actions [13][14][15]. It is necessary to visualize the bottleneck changes in each step to come up with an accurate enough mitigation method.

Availability of XML standards and associated technologies have paved new revolutionary paths for SPE using LQN. Component based performance modeling

techniques [16], Library Manager [17] and jEditor or generally known as just Editor are few of the initiatives of RADS recently taken to reduce the overall SPE time. Component based modeling (CBML) and Library Manager Tool reduce the modeling time by reusing previously built models while Editor attempts to provide a Java based development environment for performance engineering.

1.1. Motivation

There are three levels of motivation for this research. The top level or executive level motivation is to provide an easy to use development environment for SPE that considerably reduces the model building and analysis time. The second level motivation is, by using existing LQN tools [16][17][18] and techniques, to automatically identify and visualize crucial SPE information: Bottleneck and Saturation Breakdown. Note, saturation breakdown analysis involves breaking down a Task's CPU utilization, call-block and idle fractions. Visualization of both bottleneck and saturation breakdown is vital for proposing an appropriate mitigation strategy. The bottom level motivation is to provide foundation for future research that will automatically mitigate Software Bottlenecks.

1.2. Contribution

In accordance with above mentioned motivation, following are the thesis contributions:

1) Enhancement of LQN Editor to support LQML (Layered Queuing Model in XML) and CBML (Component Based Modeling Language)
   - **Provides DOM-XML parser for LQML and CBML.**

   This is the foundation for bottleneck identification and saturation breakdown analysis since the output results from LQNS will be in XML format. Background information and contributions are discussed in Chapter 3.
   - **Component Based Model Creation, Manipulation and Visualization**

     Editor is enhanced to support component driven modeling described in [16].
2) **Bottleneck Identification, Saturation Breakdown and Visualization**

Basic algorithms for Software Bottleneck identification established in [15] by Peter Tregunno are encapsulated into an automation algorithm. Visualization of bottleneck and saturation breakdown is produced using Java SWING based frameworks of the Editor.

3) **A case study** demonstrating the capability of the enhanced LQN editor.

1.3. Thesis Organization

This thesis is organized into seven chapters. Necessary background materials such as SPE, Layered Queuing Network and component based SPE are provided in Chapter 2. Chapter 3 explains the DOM-XML parser for LQN, CBML and LQNS results. In Chapter 4, approach to software bottleneck identification and saturation breakdown analysis is given. Chapter 5 discusses the software implementation details of bottleneck and saturation breakdown. In Chapter 6, a case study demonstrating the capability of new Editor is presented. Conclusions and future research opportunities are discussed in Chapter 7.

Chapter 2. Background

2.1 Software Performance Engineering (SPE)

The discipline of Software Performance Engineering has emerged in recent years to provide a standard and well structured approach to performance studies. Smith, a pioneer in this field, defines SPE in [1] as follows:

> "SPE is the systematic process for planning and evaluating a new system's performance throughout the life-cycle of its development. Its goals are to enhance the responsiveness and usability of systems while preserving quality."

The mantra for SPE is "Performance Analysis should be done at early stage of software development." [1]. There are countless projects which have been cancelled or delayed as a consequence of performance failure [19]. Generally, performance is understood as the response time experienced by the end user. Although meeting the response time and throughput requirements of the end user are the ultimate goals, SPE attempts to achieve this by first addressing

1. Bottleneck,
2. Saturation Breakdown,
3. System Delay,
4. System Scalability and
5. System Capacity

SPE heavily relies on building performance models that represent a system under consideration. Then, solving the model will produce analytical results that give insight into performance criteria such as response time and throughput. If the insight gained by analytical result is unsatisfactory, SPE will propose improvement strategies and change the models to reflect them. This process will be repeated until acceptable performance values are gained. If SPE is conducted at the early stage as suggested by pioneers [1][2][3][4], even an architecture change is possible to meet overall performance requirement.

2.1.1 SPE Vs Performance Tuning

Some mistakenly think SPE and Performance Tuning are same. Although both attempt to answer the same ultimate question mentioned above, performance tuning, widely practiced in the industry, is done on an existing system at the last stage of the development cycle. Smith [1] calls this "fix it later" mentality. Performance tuning is conducted by elite members of the Software Engineering team who have in-depth knowledge and historical perspectives of the system. They often randomly tune here and there at the implementation level to improve the overall performance. This will provide only limited performance gain not worthy enough for the effort put in. If the performance constraint is caused by the architecture, it cannot be changed at the last minute.

2.1.2 Practical Issues of SPE

Software industry's acceptance and adoption of SPE has been very slow. Many researches, [14] and [20] for example, were conducted to identify the reasons for not adopting SPE in industries. The common findings were that SPE is inherently difficult, requires special training and above all, it is time consuming. One of the interesting conclusions of [14]is that software engineering discipline lacks "standard of practice" for SPE.

A first attempt to standardize the performance model building is addressed in "UML Profile for Schedulability, Performance and Time" (SPT) [21]. Although UML performance annotations are standard, translating it into different kinds of performance models is not standardized. Performance By Unified Model Analysis (PUMA)[6] attempts to provide a common framework for translating SPT into (1) queuing model, (2) layered queuing model, (3) stochastic Petri nets, (4) stochastic process algebra models, and (5) simulation models. In addition, [22] and [23] address SPT based common software performance ontology. These attempts along with the future research will definitely reduce the practical issues of SPE.

2.2 Use of Models in SPE

There are two popular types of modeling techniques generally used for SPE:

1. Queuing Networks

In a Queuing Network Model, a computer system is modeled as a network of queues. This network of queues consists of "Service Centers" or Servers that represent computer resources and customers that represent transactions or jobs. Service Centers can be any active computer resources such as Processors, Network Interface Cards or disks. Once a Queuing Network Model is built, it can be evaluated using Queuing theory to predict computer performance. In other words, models can be solved either by simulation or analytical solutions that employ queuing theory. Using these techniques, according to [24], throughput and utilization of a computer system can be typically predicted within 10% of measured values. Queuing network models are categorized into three groups based on the population's arrival: closed, open and mixed. In closed queuing network, population is fixed as customers circulate the service centers. A typical example for a closed system is a computer system with fixed number of users who are sharing the time. In an open queuing network, the population is not fixed; that is customers can enter or leave the network. A network device that is processing streams of packet is an example of open network model. In a mixed network, as the name indicates, some customers will circulate the network while others enter or leave the network.

Even though queuing network modeling is capable of predicting a computer system's performance such as throughput and response time, its inability to express software contentions was a challenge for this research community. In other words, a queuing network cannot model the following common software aspects:

a. intermediate software servers which accept request from client process and in turn send requests to other process,
b. parallelism which involves fork and join and
c. jobs or transactions that have phase of execution which typically has early reply.

The general solution approach for software contention is extended queuing networks [1][25]. The Layered Queuing Network (LQN)[13][26] model used in this research also extends the legacy queuing model to address the above mentioned issues. LQN is discussed in details in section 2.3.

2. Stochastic Petri Nets

Petri Nets are a well developed modeling technique introduced by Carl Adam Petri in 1962. Petri Nets are the ultimate models for depicting and studying concurrency in computer systems, especially for discrete event systems. It has "places" that represent states or resources of the system and "transitions" that represent transactions. There is a concept of "token" that moves between places upon meeting certain conditions. They represent the state of the system. Appropriate transitions will be triggered to indicate state changes if a place has one or more tokens. Once the transition is triggered, token will move to its output place enabling the output place's transition.

Since Standard Petri Nets could not express the triggering probabilities and delay of transitions, Stochastic Petri Nets (SPN) were derived. It has different type of transitions such as timed and immediate. Once a transition is enabled, it has a probability of firing and an exponentially distributed delay. Once an SPN model is built to reflect a computer system, it can be solved to predict the performance.

Clearly there are advantages of SPN as a modeling technique for SPE. Its inherent closeness to state machine is an attractive point to transform UML state charts into SPN [9][27]. However, the disadvantage of SPN comes from "State Space Explosion" since SPN is converted into Markov Chain states prior to producing SPN solution.

2.3 Layered Queuing Networks

As mentioned earlier, the foundation for this thesis work is Layered Queuing Networks (LQN) models. LQN is an extension of traditional queuing networks with the support for intermediate servers, phase of execution and both software and hardware contention [12][28]. In addition, the layered structure of LQN bears a resemblance to N-tier client-server systems that are inherently layered. This resemblance helps the software engineers to visualize the systems clearly.

LQN introduces the following notation to model computer systems:

1) **Processors:** embody the physical resources such as disc, CPU or NIC.
2) **Tasks:** tasks are the interacting elements of LQN model. It has a thread of control and initiates or accepts request from or to servers. Apart from carrying out operations, it has attributes such as scheduling discipline, queues and

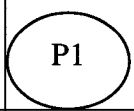
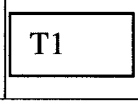
multiplicity. Tasks in LQN can represent a software process or a hardware resource such as CPUs, disks and network interface cards (NIC). They have one or many Entries (discussed in detail below) that offer particular services and a single host processor that models the physical entity that executes the operations. There are three types of tasks:

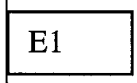
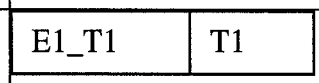
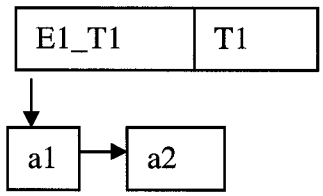
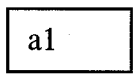
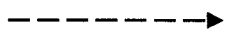
i. Active Tasks – initiate and accept requests
ii. Reference Tasks- do not accept any requests. They only initiate requests. They are used to model load generators or users.
iii. Pure Servers – only accept requests, never send requests.

3) **Entries:** entries are the service offering entities of LQN model. For example, they can be realized as C++ functions and their parent task can be realized as an Object. They will make requests to the other entries residing in lower layer task to complete a service. Entries call neither their sibling entries nor entries in the same layer. It also has the execution demand parameter.

4) **Activities:** Activities can be realized as a smaller version of entries. They represent the smallest units of executions. They allow an LQN model to express parallel, alternative and repetitive executions. They do so by employing the concept and notation of "AND forks and Joins" and "OR forks and Joins".
5) **Calls:** calls represent the request of services from an Entry or Activity to another Entry or Activity. LQN supports three types of calls: 1) Synchronous, 2) Asynchronous and 3) Forwarding calls.
6) **Demands:** demands are the average execution time to complete the operation of Entry at the local host processor.

2.3.1 Graphical Notation of LQN

Entity Name	Graphical Notation	Description
Processor		Representing a host where service is executed.
Task		Each task box must have a unique name in the model.

Entry		Each entry box must have a unique name in the model. Generally, <entry name>_<task name> convention is followed to form unique entry name.
Task with Entry		Task boxes are displayed in the most left. Entries are displayed from right to left. There is no rule in the order in which entries are displayed.
Task with Activities		Activities are considered to be an attribute of Tasks. However, an entry should initiate an activity. Therefore, there will be a starting entry.
Activity		Activity is also graphically represented by box with its name in it.
Activity Connection/Sequence		OR Fork/Join
		AND Fork/Join
Synchronous Calls		Send and wait until reply is received.
Asynchronous Calls		Send and don't wait for the reply
Forwarding Calls		Request is forwarded to a different entry.

2.3.2 Realizing LQN models from UML

UML (Unified Modeling Language) is a widely used standard notation for modeling objects as a first step in software development. As stated in the introduction, SPE should be conducted at the early stages of software development cycle. Thus, finding a common pattern between UML and LQN is crucial to reduce the LQN modeling time for the system expressed in UML. We can even go one step further and

include performance parameters specified by [21] into UML notations to automatically generate LQN models. Automatically generating LQN models has two advantages: (1) reduced time to build models with less complexity for the users and (2) less error prone. In the following, through examples taken from [12] and [13], modeling LQN from UML notation is described:

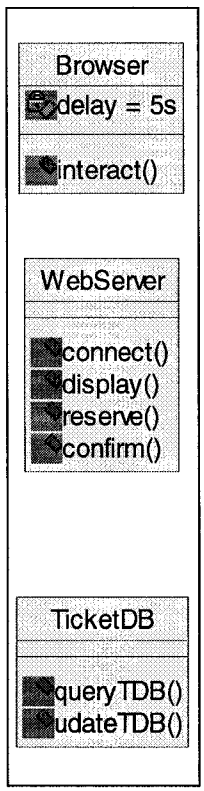


Figure 1 Class Diagram for Ticket reservation system

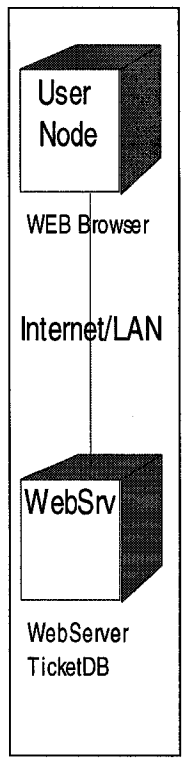


Figure 2 Deployment Diagram for Ticket Reservation System

The two nodes in the deployment diagram in Figure 2 would suggest there are two layers for the Ticket Reservation system. Although the WebServer and TicketDB are running on the same node, WebSrv, they are two unique processes with completely different goals. Furthermore, TicketDB is there to serve WebServer. Thus, it is important to model them into two separate layers. Figure 1 depicts the class diagram for the ticket system whereas Figure 3 depicts the LQN equivalent model.

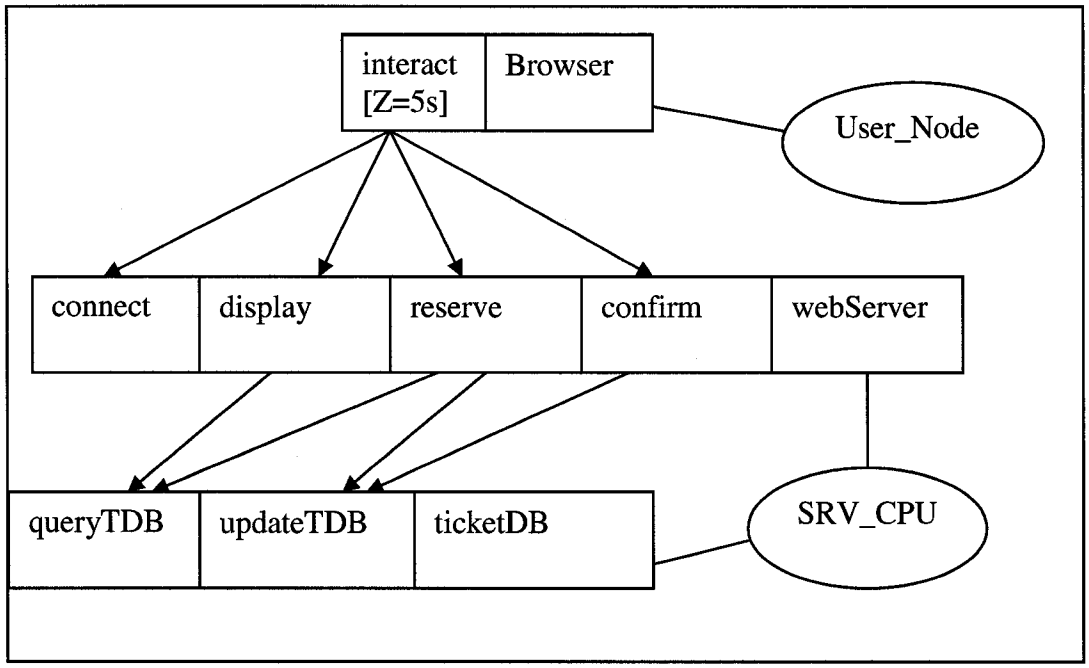


Figure 3 LQN model derived by using UML information from Figure 1 and Figure 2

2.4 LQN Editor

The LQN Editor, commonly known as Editor, is a JAVA Swing/AWT based Graphical User Interface (GUI) that allows a model builder to easily create, edit, view and solve a LQN model.

2.4.1 Purpose of the Editor

The motivation for developing this tool was to hide the LQN input language from the modeler. By doing so, the learning curve and complexity of manually creating input files are eliminated. This reduction in complexity is an attractive solution to difficulties in applying SPE in industry documented in [1][2][20]. A modeler needs to know only few fundamental procedures to create an LQN model rather than learning the syntax of input file formats. In other words, creating, editing and visualizing an LQN model is few mouse clicks away in LQN Editor.

2.4.2 Foundations of LQN Editor

Two foundations of the Editor are:

1) Java classes representing LQN language

   A collection of Java classes were developed by researchers at RADS lab to represent the core LQN language. It is commonly known as core classes and will be referred as CJC (Core Java Class) hereafter. The motivation was for CJC is to provide a standard software framework which is flexible enough to develop LQN related tools. Research work [15] and [16] are few examples where this framework is heavily used. CJC is architectured to reflect the LQN hierarchy depicted in Figure 3-3 in [16]. It has four major groups of Java classes: (1) a group that models LQN entities, (2) a group that reflects LQN models, (3) a group that representing the logical aspects of LQN such as evaluating the Layers and (4) the parser classes to parse Java instance to LQN input format text.

2) Parsers to transform LQN files into Objects and vise versa.

   Editor uses a well known third party parser and Lexer generator, JB [29] to parse LQN input files and produce corresponding CJC objects. JB takes parsers produced by Gnu Bison and translates them into Java. It takes the C file output from Bison, scans it to extract parsed tables and constants.

   Once CJC objects are created using this parser they can be manipulated using any JAVA application and saved back to LQN input file format. This enables researches to automate LQN related activities such as merging or aggregating Tasks.

2.4.3 Architecture of LQN Editor

Initially, the Editor was developed in RADS labs as a separate application without using CJC. As preparation for this research work, Editor was modified by the author to use the CJC framework. Now, Editor has a one-to-one Graphical Mapping with CJC. For example, a Task class is mapped into TaskDialog class which extends the JDialog from JAVA Swing component. This TaskDialog class will provide graphical interface to

modify the CJC Task instance within a Model. There is yet one additional mapping involved with CJC entities when composing the Layered View, discussed later in this chapter. Figure 4 illustrates how jLQN Editor is composed into logical components and interacts with CJC objects and LQNS solver.

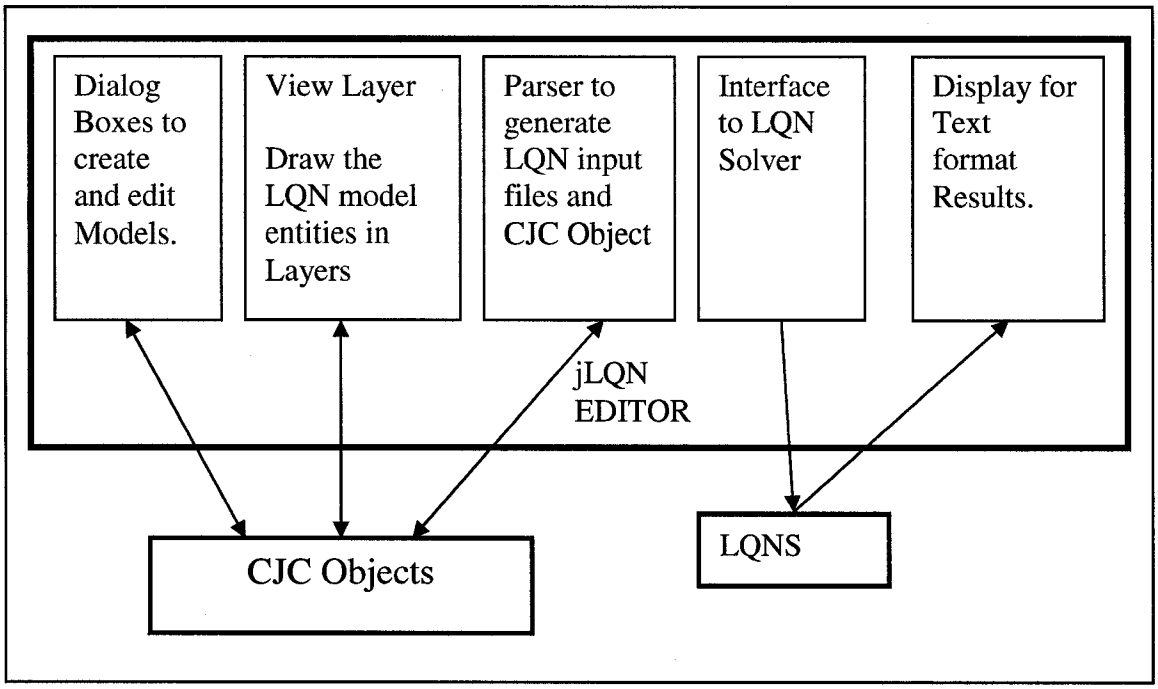


Figure 4 Description of jLQN components

2.4.4 Building a Model using LQN Editor

There is a main window that acts as a dashboard in LQN Editor. Its Dialog Boxes and menu bar items provide all the necessary functionalities and mechanisms to create, edit and solve a model. It has a list for Processors, a list for Tasks and a list for Slots (used for CBML [16]) [5]. Plus, it has a text area where you can enter the comments for the model. In the menu bar, it has File, Solve, View and Help items that are self-explanatory.

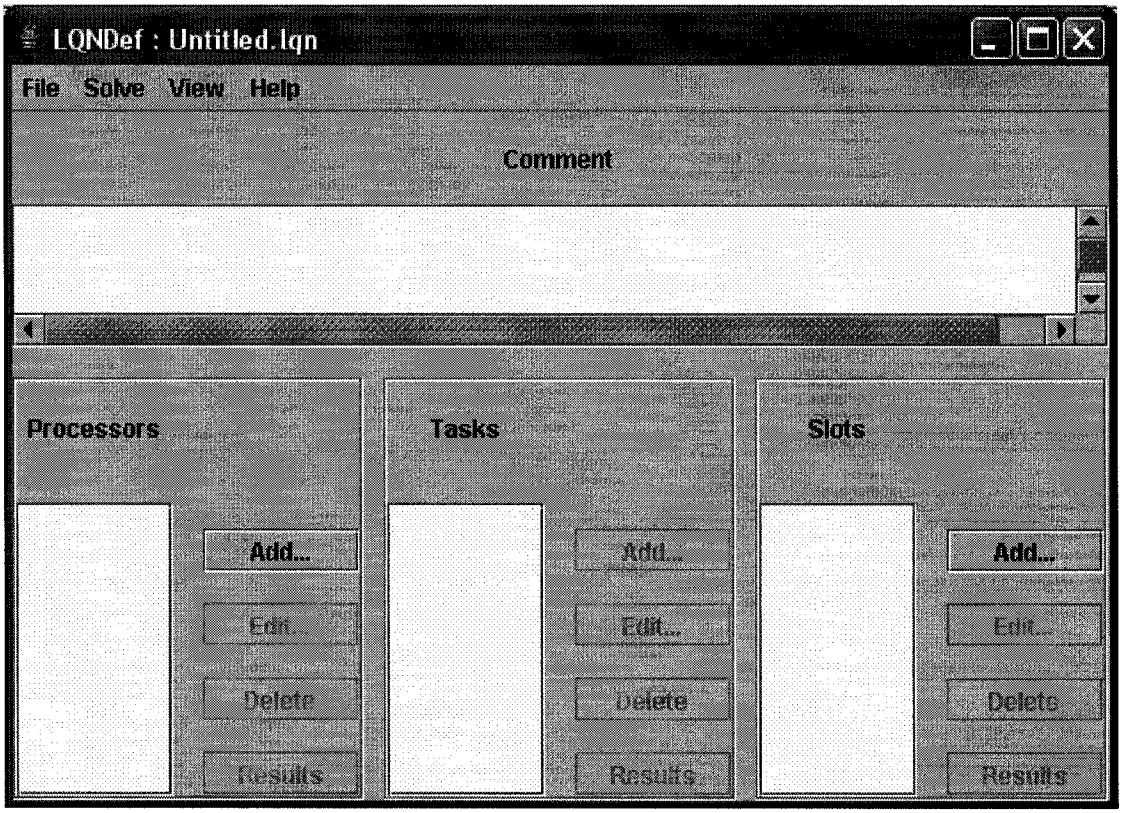


Figure 5 Main Window of LQN Editor

To build a model, one can start by adding the Processors to the model by clicking on the Add button of the Processors list that is located on the bottom left corner of the main window. This will launch a Processor Information dialog box (see Figure 6)where the user should provide 1) Scheduling discipline, 2) Quantum, 3) Multiplicity 4) Replicas and 5) Speed factors. LQN Editor will provide default values for each of these fields. It is user's responsibility to change them according to the model specification.

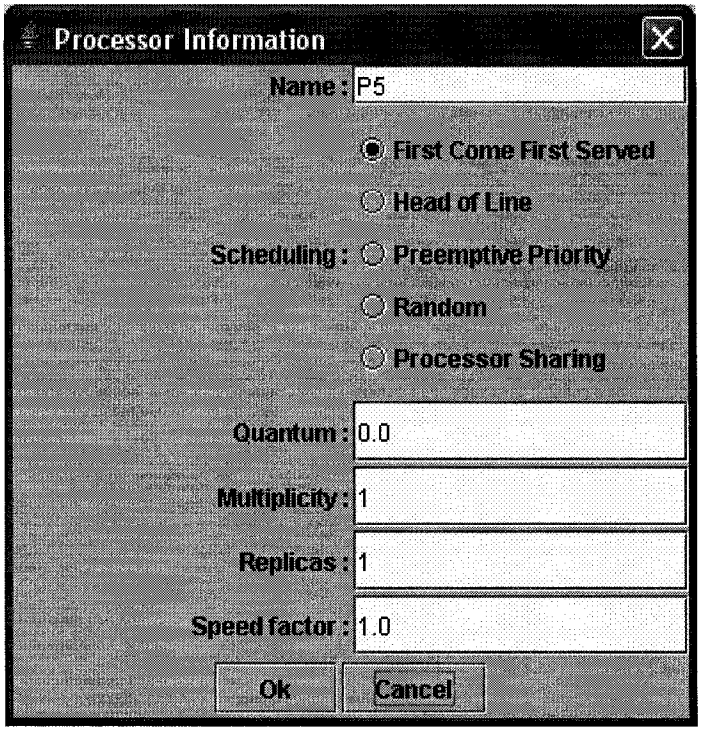


Figure 6 Processor Information Dialog box

Once all the Processors are created, the next step is to create the Tasks supported by each Processor. The "Add" button in Tasks list will be enabled once there is at least one Processor available in the model. Clicking on it will launch the "Task Information" dialog box which asks for 1) type of the Task, 2) list of Entries to support, 3) Activities it supports, 4) Priority, 5) Multiplicity, 6) Think Time 7) Replicas and 8) Processor where this Task is executing.

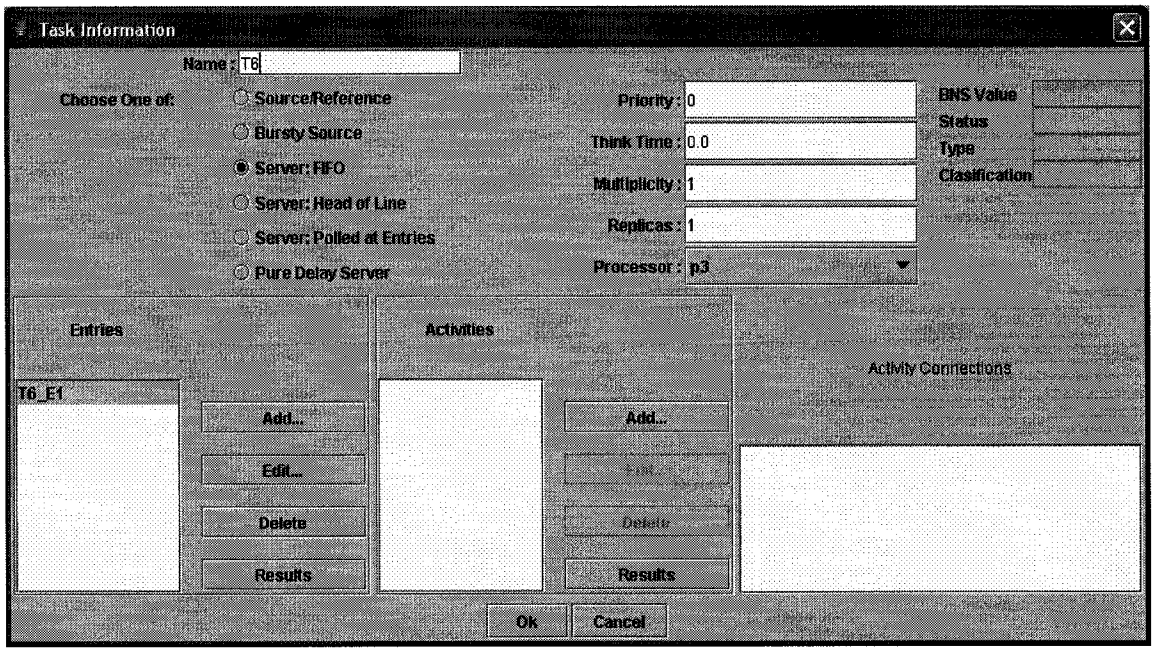


Figure 7 Task Information Box

Entry and Activity information is provided using this "Task Information" box.

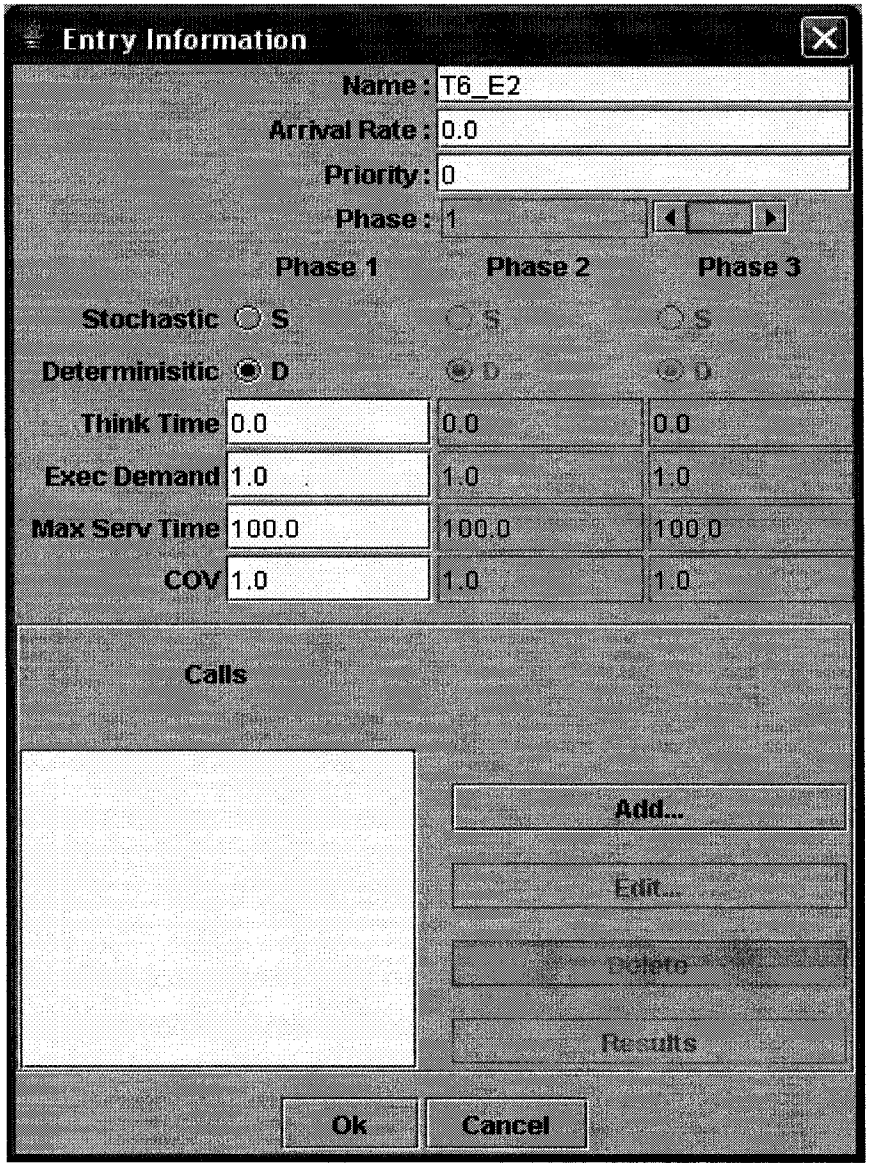


Figure 8 Entry Information

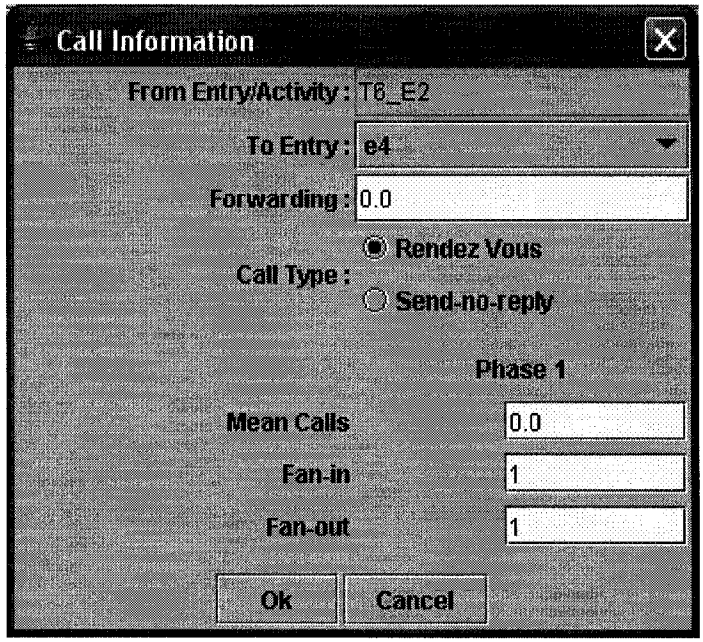


Figure 9 Call Information

Adding Entries are straight forward compared to Activities. Activities need additional information known as "Activity Connections". This is done using a pure string format and parsed internally to map into LQN input format.

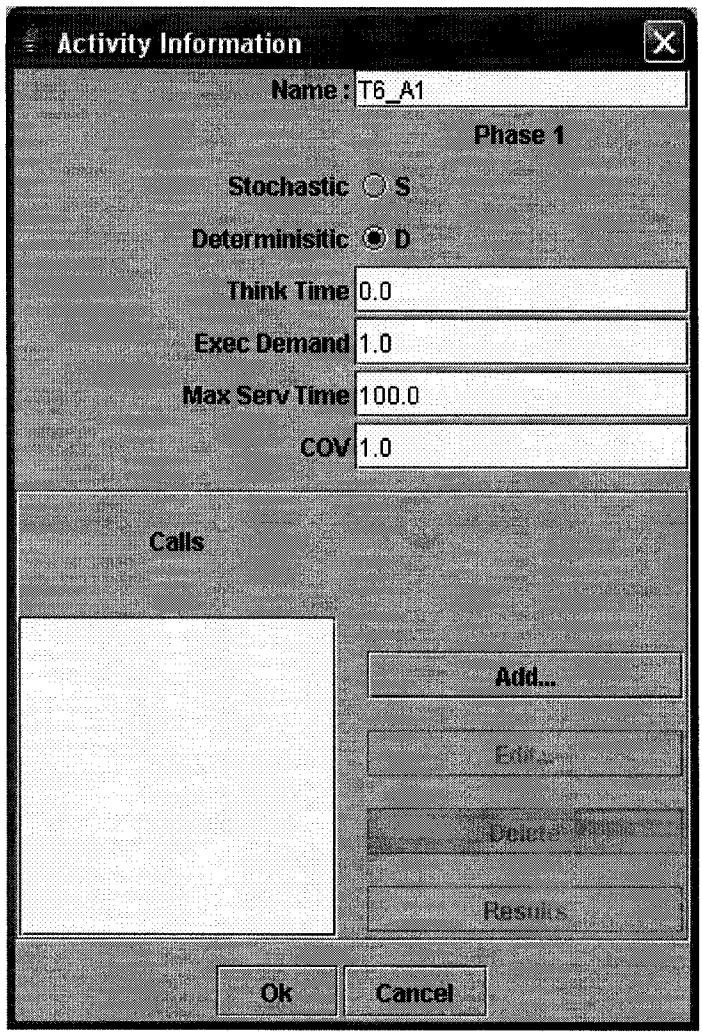


Figure 10 Activity Information Box

2.4.5 Solving a Model using LQN Editor

When models are built or loaded, Editor will evoke the analytical solver (LQNS) to solve the model and display results in pure text format. With this text Result format, filtering and presenting the performance metric that a user wishes is a highly complicated task. Part of this research involved providing foundational support in CJC for XML based results. XML based results are much more flexible for filtering. In fact, the focus of this research, identifying and breaking down the Software Bottleneck, is made possible by having XML result capability in Editor.

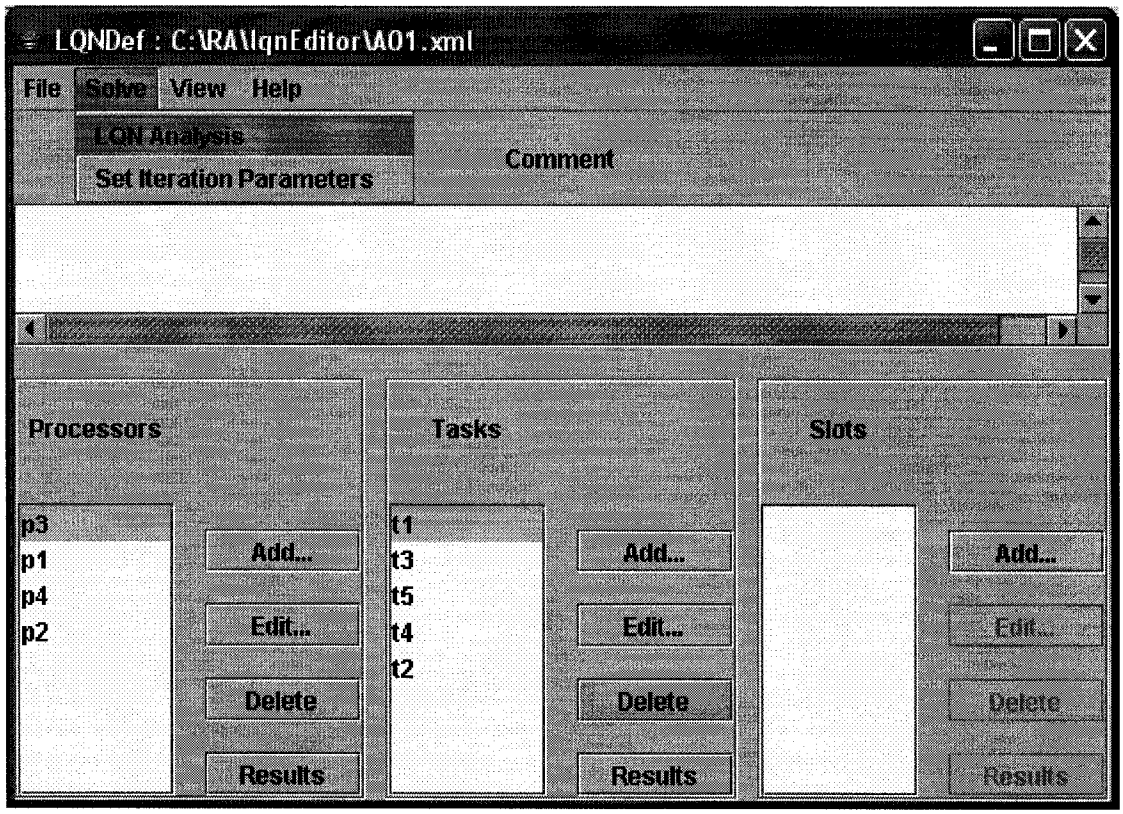


Figure 11 Solving a model using LQN Editor

2.4.6 Layered View

The purpose of Layered View functionality is to visualize a model's layers and calls. Apart from gaining intuitive understanding of the system, call congestion and modeling mistakes can be easily identified by visualizing a model. In addition, it can be printed for research publication.

Starting from the reference tasks, it displays all tasks along with their Entry (see Figure 12 for example) and Activities. Processors are not shown as part of the layers. For graphical displaying purpose, there is a one to one mapping of major CJC model object to its respective Graphical object. For example, a Task object will have a corresponding GraphicalTask object once Layered View is launched. Figure 13 depicts class diagram of the graphical objects used in layered view. Layers are formed and labeled based on the calls each Entry makes or receives. Each layer must have one or more GraphicalTask. Upon launching this view, Graphical Task, Entry and Activity will be drawn as rectangle boxes using Java Swing 2D graphic utilities. Calls will be drawn

as arrows. These graphical notations are compliance to the notations discussed in section 2.3.1.

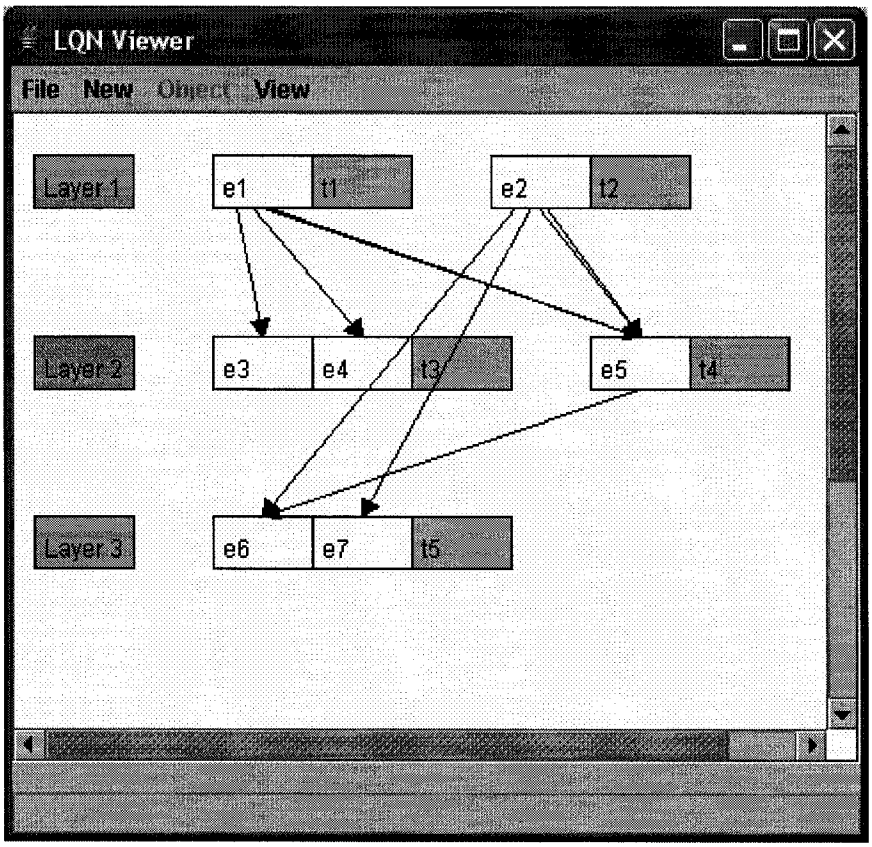


Figure 12 Layered View

Information such as Task or Entry can be changed by double clicking on the desired graphical object causing the respective dialog box to pop-up. Using these graphical objects and layered view as foundation, this research provides a completely unique dimension to visualizing LQN model by introducing (1) bottleneck view and (2) saturation breakdown.

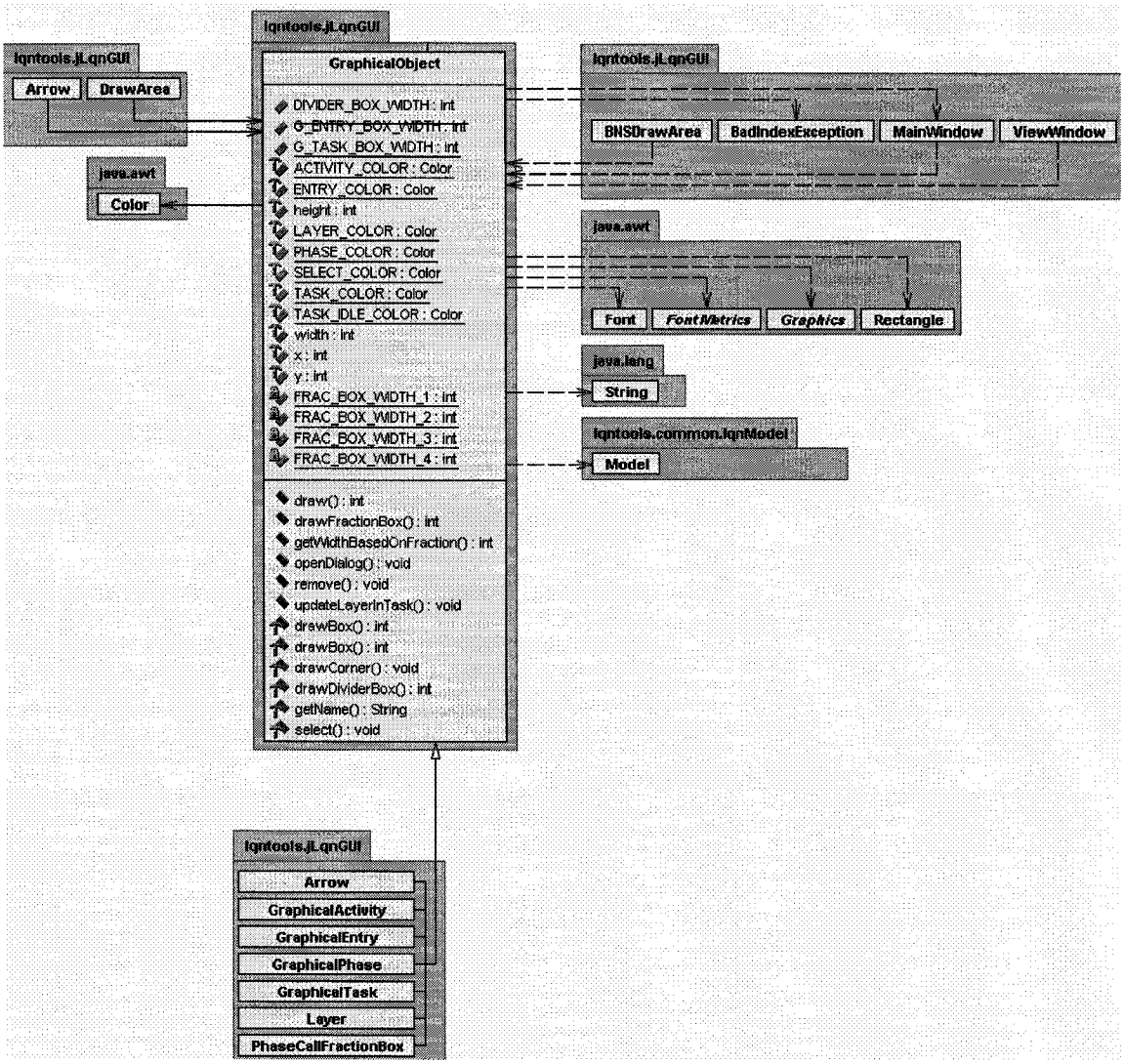


Figure 13 Class Diagram for Graphical Object[1]

2.5 Introduction to LQML

An initial version of XML schema for LQN input file known as LQML was developed by Wu in [16]. Although it has been changed in subsequent researches, the main architecture remains the same. Since LQN language is organized hierarchically, XML's inherent tree structure is appropriate to express LQN model. Figure 3-3 of [16] portrays the hierarchy of LQN model. Figure 3-5 of [16] depicts the XML equivalent (LQML) of LQN model hierarchy.

One of the advantages of having LQML is the ability to build LQN models without the knowledge of LQN input format. Once a modeler has the schema for LQN and an

[1] UML class diagram was automatically generated by JBuilder Enterprise Edition from source code.

XML editing tool, he or she can easily build the models without worrying about the LQN input file format. However, composing LQML model manually using ordinary text editor requires extensive work. In doing so, a modeler has to type all the XML element tags and attributes which is time consuming and recursive process. LQN Editor, which hides even the XML input format with few mouse clicks, can play a vital role in such situation.

2.6 Introduction to CBML

Wu also introduced CBML (Component Based Modeling Language) in [16] to meet the growing demand of Performance of Component Based Software Engineering (CBSE). CBSE is a methodology to build new systems reusing existing components [27][30][31][32][33][34][35][36]. A component can be a multipurpose subsystem or a single module that can be reused as a building block for creating new systems. Since reusability is attractive due to low cost and high quality, CBSE gained momentum in industry.

As seen earlier, LQN is a system oriented performance modeling language. In the context of CBSE, LQN can be applied when a new system is built using components. In other words, every time a new system is produced using existing components, an LQN model must be built from scratch to represent the newly built system. Rather than building from scratch, Wu's approach is to maintain individual performance models representing each component and assemble them to form a final system level model. However, the legacy LQN language is not sophisticated enough to handle the concept of "sub-model", a term representing a performance model of a component. Therefore, CBML is introduced to model a sub-model. From Figure 14, which depicts a graphical representation of an example component, the reader can observe Wu's newly added concepts:

1. **Incoming Interfaces**- represented by circles, provide hooks to plug into a main model.
2. **Component body**- contains an LQN model.
3. **Outgoing interfaces** - represented by squares, provides hooks to plug into a main model.

4. **Replaceable processors** – represented by a circle with embedded 'P' represents processors that can be replaced by the main model processors.

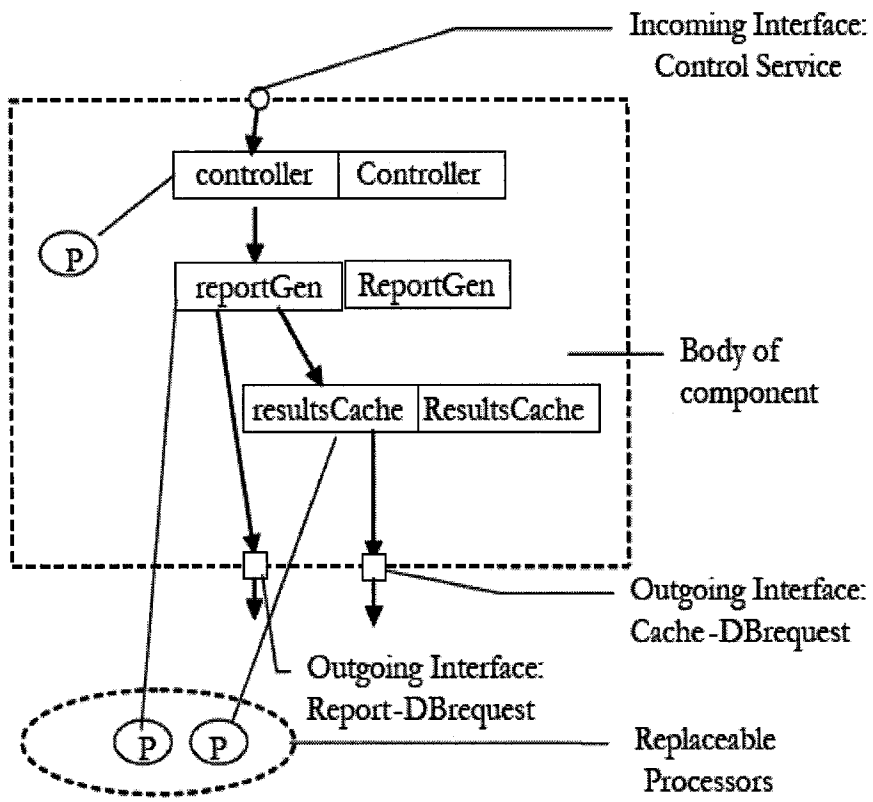


Figure 14 Graphical representation of an example CBML model [16p.27]

One additional concept, not shown in Figure 14, is a Slot. It's a conceptual holder that connects the main model and a sub-model. It also allows nested components depicted in figure 3-6 of [16]. Mechanical aspects of the slot are discussed in section 2.7.1.

2.7 Notations introduced for CBML

To support CBML modeling, Wu in [16] introduced few additional notations. These notations are elaborated in the following subsections and depicted in Figure 15 taken from [16].

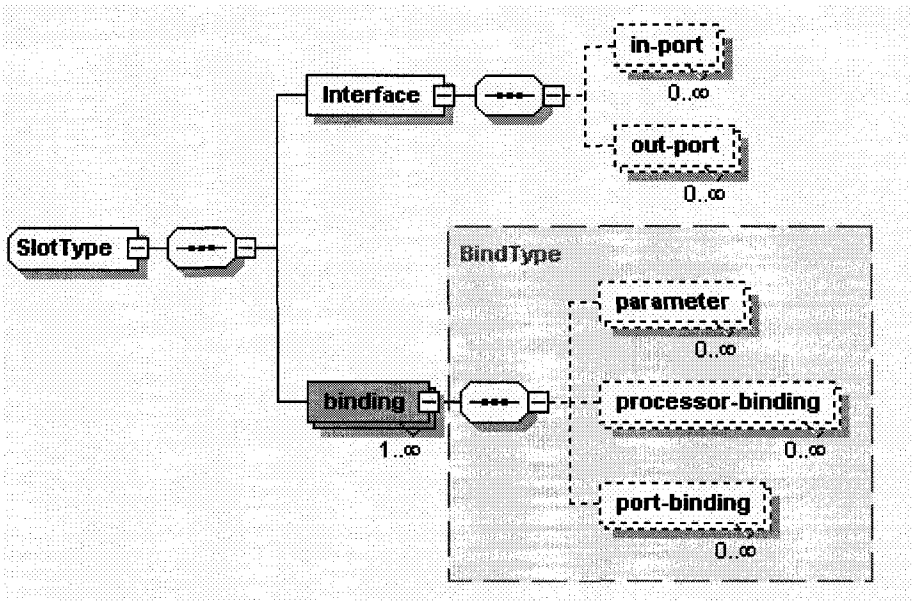


Figure 15 Sub-model related notations

2.7.1 Slot

Slot is a conceptual placeholder which can contain a sub model. As seen in Figure 15, it consists of Interface that defines the connections between sub-model and slot, and binding that describes how a sub-model can be bound to form system level model. A system model can have any number of slots corresponding to each component or sub-model it includes. In addition, a Slot can have nested slots within its sub-model enabling to build super size components from a collection of smaller ones.

2.7.2 Interface

An Interface of a Slot contains two elements: (1) collection of in-port and (2) collection of out-ports to define the connection between the sub-model and slot. An In-port always connect to an Entry within the sub-model where as out-ports receive a connection from a sub-model's Entries. Graphically, in-ports are represented by circles whereas out-ports are represented by square boxes.

2.7.3 Binding

Information in Binding is necessary to plug-in the sub-model into the main system level model. During this process, known as "assembling" a model [16], there exists a need to substitute system specific information in the sub-model. A perfect example

would be the processor information of the new system trying to build. It is possible that the sub-model was built to the specifications of a different processor. In order to assemble a new system level model using this sub-model, a modeler must replace all the old information with the current ones. Rather than manually changing it in the sub-model every time, the Binding tag under Slot will hold the replaceable information that will be automatically substituted by the assembler [16].

Thus, binding contains Processor-binding (replaceable processors), Port-binding (replaceable ports) and Parameters (any parameters within the sub-model to be replaced) as its information holders.

2.7.4 Summary of Notations for CBML

The following table summarizes the notations and concepts introduced to handle CBML. Information in this table is vital for understanding the parsing algorithm (Algorithm 2) and visualization of sub-model in Editor that are discussed in later sections.

Table 1 CBML notations

Name of the Element	Attributes	Description of Attributes
Slot	Id	A unique "String" id of the Slot
	bind-target	The name of the sub-model that is to connected this slot.
In-Port	name	Port name
	connect-to	Name of the Associated Entry in sub-model
	description	Description of the port
Out-Port	name	Port Name
	connect-from	Name of the Associated Entry in sub-model
	description	Description of the port
Parameter	name	Parameter name in the sub-model

	value	Value to be substituted
Processor-binding	source	Name of processor in the sub-model needs to be replaced
	target	Name of the substitute processor
Port-binding	source	Port name of the inner sub-model
	target	Port name that will replace the source.

Chapter 3. Editor Enhancement for XML

The author restructured the Editor and extended it to accept LQML/CBML models. Hereafter, use of the acronym LQML means CBML as well. Changes were made to provide:

1. a parser to read in LQML files and transform them into CJC objects. Once a model is parsed into CJC objects, manipulation and presentation of the model is easy. Thus, this parser is the foundation for this research.
2. a converter that transforms CJC objects into LQML. This is necessary when models are built using the Editor's various dialog boxes and need to be saved as LQML. In such cases, Editor stores a model as a collection of CJC objects. These objects can be preserved as legacy "lqn" model or LQML/CBML before interacting with various solver/simulator tools.

Since DOM [37] technology is used to implement both the converter and the parser, it is appropriate to discusses them together.

This chapter articulates (1) introduction to DOM and related technologies, (2) high level requirements for the parser and the converter and (3) a detailed design of transformation algorithms.

3.1. High Level Requirement

At a highest level, this parser should read in the LQML model and convert it to CJC objects. Once the conversion is done, editor should be able to manipulate these objects as usual using existing dialog boxes.

3.2. Introduction to DOM

The Document Object Model (DOM), is a language independent programming interface specification developed by World Wide Web Consortium. It is an open effort to offer programming control over documents. In other words, DOM allows HTML and XML documents to be transformed into full-fledged program objects. There by, programmers have the flexibility to create and modify HTML or XML document by

means of dynamic object interactions and modifications. Any XML or HTML element can be individually addressable by programs.

It is worthwhile to note that DOM is not the only XML parser available. There is a light-duty event-driven SAX (Simple API for XML) parser. Unlike DOM, SAX reacts to the event of finding any particular XML element. This is useful when many or large XML files are processed. The disadvantage of SAX is its limited capabilities in manipulating data contents at any time as required. In DOM, the document is loaded into memory as an element tree and can be accessed or manipulated at any time.

DOM has two levels of specification. One is DOM Core that supports XML and other is DOM HTML that extends the core to support HTML documents. Since this research focus is to parse only XML documents, DOM Core is used. As the name means, DOM core provides basic set of objects and interfaces needed to create and manipulate XML documents. Further functionalities such as Cascading Style Sheets and events are not supported by DOM core [37].

DOM contains three core interfaces: (1) Documents, (2) Elements and (3) Entity Nodes. These interfaces correspond to objects at various level of XML hierarchy.

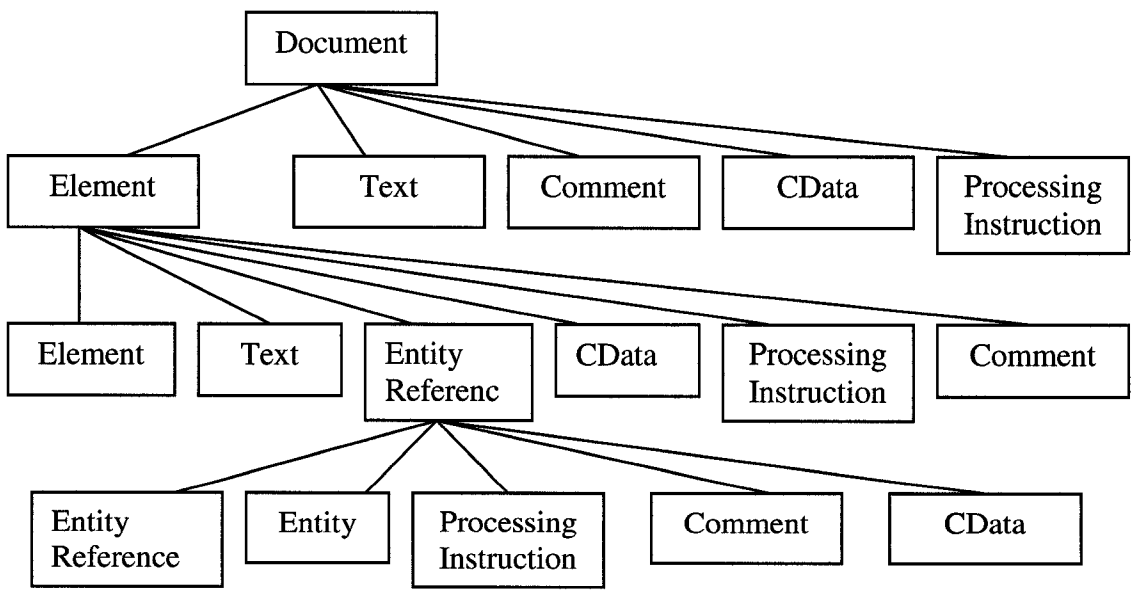


Figure 16 Parent/Child Relationship in DOM [37p24]

DOM nodes represent the branches, sub-branches and leaf elements/tags of an XML tree. A Node provides a set of hierarchical APIs. Every other interface is descended from Node thereby inheriting basic functionalities to do the following:

1. Find Node information such as name, type and value,
2. Read, update and delete Node information and
3. Find parent, sibling and children Node information.

One may ask, if Node represents everything, how to differentiate various nodes and information contained in them? This mystery is solved by maintaining a node "type". The getNodeType() method will return type of the Node defined in Table 2. A DOM tree walk through can be easily done by casting the Node to appropriate objects based on the type and using the widely used methods listed in Table 3. The LQML parser transforms LQML information into CJC objects by walking through the DOM tree in a systematic manner. For example, as a Processor Node in Figure 17 is walked through, using APIs listed in Table 3, a corresponding CJC Processor object will be created. This systematic walk through is governed by XML Schema validation and complex algorithms that are discussed in subsequent sections.

NodeType Value	**Named Constant**
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE

12	NOTATION_NODE

Table 2 DOM Node Type and Name Constant [17]

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <lqn-model description="Generated by: srvn2eepic, version 3.0" name="A01" xmlns:xsi="http://www.w3.org/2001/XMLS
  instance" xsi:noNamespaceSchemaLocation="/home/maly/usr/src/xml/lqn.xsd">
   <!-- Invoked as: srvn2eepic -Oxml A01.in  -->
   <!-- Wed Apr 28 17:23:47 2004  -->
   <solver-params comment="A1" conv_val="1e-06" it_limit="50" print_int="5" underrelax_coeff="0.9" />
  + <processor name="p1" scheduling="fcfs">
  + <processor name="p3" scheduling="fcfs">
  + <processor name="p4" scheduling="fcfs">
  + <processor name="p2" scheduling="fcfs">
  </lqn-model>
```

Figure 17 XML Tag for Processor

Table 3 Important API to interact with a DOM Node

Scenario	Methods

Query Node Information	• Short getNodeType()
	  o Returns the type of the node.
	• String getNodeName()
	  o Returns the node name
	• String getNodeValue
	  o Returns the node value
	• void setNodeValue(String)
	  o Sets the Node value.
	• NameNodeList getAttributes()
	  o Returns the attributes.
	• boolean hasChildren()
	  o Returns TRUE if this Node has any children.

Query Children Node Information	• Node getFirstChild()
	  o Returns the first child of the node. If there are not any children, this will return NULL.
	• Node getLastChild()
	  o Returns the last child node or null.
	• NodeList getChildNodes()
	  o Returns a List of child Nodes.

Query Parent or Siblings related Information	• Node getParentNode()
	  o Return the parent of the node or return NULL if the parent Node is Document, Document Fragment, or Attribute object.
	• Node getPreviousSibling()
	  o Returns the immediate sibling to the present node.
	• Node getNextSibling()
	  o Returns next sibling to the current node.

Manipulate the Children Node	• Node insertBefore (Node new, Node reference)
	  o Inserts a node before the reference Node
	• Node replaceNode (Node replacement, Node tobereplace)
	  o Replace the current node with the given one.
	• Node removeNode(Node rmvNode)
	  o Remove the node.
	• Node appendNode(Node tobeapended)
	  o Appned a Node to the current node.
	• Node cloneNode(boolean deep)

- **XML Schema**

XML schema is a language that constrains and describes the structure of an XML document. It governs and guarantees all XML elements of a document follow the predefined structure. As recommended by W3, it is written in XSD (XML Schema Definition) with capabilities of data types and namespaces. The primary motivation of namespaces is to avoid naming conflicts when reusing multiple vocabularies. Namespace are absent in LQN related schemas since the LQN language doesn't contain conflicting vocabularies. This research utilizes two schemas, (1) lqn-core.xsd and (2) lqn-submodel.xsd, developed and maintained by RADS lab researchers.

3.2.1. Selection of DOM implementation

As stated earlier, DOM is a standardized interface. Thus, there are plenty of implementation libraries available for developers to choose from. The DOM parser implementation world is very dynamic as it evolves with changes in the standard. However, they differ in performance, reliability, ease of use and conformance to standard. Therefore, it is vital to select an appropriate DOM implementation for this

research. Sun, Oracle and the Apache Software Foundation are the dominant players in this space. Apache (widely known as Xerces) and Sun (JXAP) are attractive to many researchers since they are free. An informal performance comparison of these parsers is documented in [38]. In [38], the author used a 92KB XML file using Windows NT4 running on a Pentium 300MGHz with 64MB of memory. The author used JAVA's System.currentTimeMillis() call to measure the time. Table 4 summarizes the result.

Parser Implementation	Avg. Parse Time (ms)/92KB XML file
Oracle Parser	1094
Sun –JAXP	1344
Apache- Xerces	1719

Table 4 Average Parser Load Times [38]

This result ranks Oracle as the fastest parser. JAXP is the second fastest parser in Table 4. Since, it is free and comes with JDK 1.4, this research chose to use it.

3.3. Detail Design of LQML parser

The very first action for this parser is to read in the XML file and create the DOM tree. Once the DOM tree is created successfully, it can contain either an LQN model (some times referred to as the lqn-core in the context of XML schema) or LQN sub-model. As seen earlier, a sub-model always contains an LQN model known as component body (see Figure 14). Thus, the LQN model elements such as Processors, Tasks etc are processed first. If parsing LQN model is successful, the parser must then check if that model belongs to a sub-model or not. This check can be done by searching for an element in DOM tree named "lqn-submodel". If it does belong to a sub-model, continue to parse the sub-model related tags.

3.3.1. Algorithm-1: Parsing lqn-core elements

The first step in parsing the lqn-core model is to retrieve a list of Processor Nodes from the DOM tree as Processors are the second level root of any LQN model [16]. Method getElementsByTagName from the Document with input parameter "processor" will provide a list of processor nodes. Check if there exists a Processor Node or not. If it exists, loop through the list and create a corresponding CJC Processor object, get the

attributes from Processor Node and set them to CJC Processor object. Then search for Tasks Nodes under it and create the corresponding CJC Task object. Plus, set the Task Node attributes to respective CJC Task objects. This process is followed for creating Entries, Activities and Calls. The following Algorithm depicts how this is done.

Algorithm-1: LQN Core model parser

Assume DOM tree is created successfully

- Create a CJC-Model object
- List *processorList* = get list of all the **processor** Nodes in DOM by looking for "processor" Tag name.
- ***for*** (all the processor nodes in *processorList*)

{

- Create CJC-Processor object and set it to Model Object
- Get "name", "multiplicity", "speed-factor", "scheduling" and "replication" attributes from Processor Node into local variables.
- Set the above attributes to corresponding CJC-Processor attributes.
  - ***if*** (this Processor Node contains "result-processor" Child)
    - Set the result Node to CJC-Processor object
- ***if*** (this Processor Node has any children Nodes ***&&*** the children are Task Nodes)

{

- List taskList = get list of tasks nodes from this Processor Node.
- ***for*** (all the Tasks in taskList)

{

- Create CJC-Task object set it to Model Object
- Get attributes "name", "multiplicity", "replication", "scheduling", and "think-time".
- Set the above attributes to corresponding CJC-Task attributes.
- ***if*** (this Task Node contains "result-task" Child)
  - Set the result Node to CJC-Task object
- ***if*** (this Task Node contains "entry" Child)

{

- List entityList = get list of Entries in this Task Node.
- For (all the Entity Nodes in the enetityList)

{

- Create CJC-Entity object set it to Model Object

- Get "name", "open-arrival-rate" and "priority" from Entity Node
- Set the above attributes to the corresponding attributes in CJC-Entity object
  - *if* (this Entity Node contains "result-entry" Child)
    - ❖ Set the result Node to CJC-Task object
  - *if* (this Entity Node contains "result-task" Child)
    - ❖ Set the result Node to CJC-Task object

Note: Entries have *Phases* that contain *calls*. At this point we can't create the call objects since we don't know if the destination Entry is created or not. Thus, phase will be handled after all the entries in the models are created.

}

}

➢ *if* (this Task Node contains "task-activities"Child)

{

- Create corresponding CJC-Activity objects
- Form and set the activity-connection string and set it to the CJC-Task object

}

}

}

}// end of for loop Processors

Note: Now ready to set the phases and their calls in Entries.

List *entryList* = Get all the "entry" Node from the DOM tree

for (all the entries in the *entryList*)

{

- Get "synch-call" and "asynch-call" Nodes in each phase of Entries
- Cereate corresponding CJC-Call and CJC-Phase objects
- Get the appropriate attributes form the Nodes and set them to CJC objects.

}

Note: Now ready to set the Activity calls

List *taskActivityList* = Get all the "task-activities" from the DOM tree.

for (all the entries in the *taskActivityList*)

{

- Get "synch-call" and "asynch-call" Nodes from the activities

- Cereate corresponding CJC-Call objects
- Get the appropriate attributes form the Nodes and set them to CJC objects.

}

3.3.2. Algorithm-2: Parsing the lqn sub-model

As described earlier, sub-model related tags are parsed after successfully parsing lqn-core tags of the model. This parser checks if there is a sub-model in the DOM by searching for "lqn-submodel" tag name. In addition, it must verify there is one and only one "lqn-submodel" with this name. Once these checks pass, child nodes of lqn-submodel such as InPort, OutPort, Parameter, and Bindings are parsed identically to that of lqn-core model. In short, corresponding InPort, OutPort and other CJC objects are created along with attribute transformation. Since the Bindings related XML schema was not standardized by fellow researchers, Editor doesn't support it. However, it provides internal holders for them for easy integration of this feature in the future.

Algorithm-2: lqn sub-model Parser

if (get **"lqn-submodel"** Node in DOM tree == SUCCESS)
{
- Create a CJC Slot object.

if(get **"Interface"** Node in DOM tree == SUCCESS)
{
List inportList = get list of all the **InPort** Nodes in DOM
for(all inportList){
- Create CJC InPort object.
- Get "name", "connect-to" and "description" attributes from this InPort Node.
- Set these attributes into appropriate member of CJC InPort object.

}
List outportList = get list of all the **OutPort** Nodes in DOM
for (all outportList){
- Create CJC OutPort object.
- Get "name", "connect-from" and "description" attributes from this OutPort Node.
- Set these attributes into appropriate member of CJC OutPort object.

}

List repProcessorList = get list of all the **"Replaceable-Processor"** Nodes in DOM

for (all repProcessorList){

- Get "name" attribute from this "Replaceable-Processor" Node.
- Add it to HashMap dedicated for replaceable-processor in Slot object.

}

}

3.4. XML Output

In order to understand the design of this CJC to XML converter, it is important to clearly define the activities involved in it. A model can exist as a collection of CJC objects either by manually building it using dialog boxes or by opening legacy (lqn) models in the Editor. When building a model, a modeler should have the flexibility to save it as XML whenever necessary. Thus, a model is not required to be complete for XML conversion.

Once a CJC model is available, converting it to XML involves the following steps:

1. Create an empty DOM tree
2. Access CJC model and create respective Model, Processors, Tasks, Entries and other related Nodes.
3. Using the parent-child and sibling relationship, connect the nodes between each other created in step 2 accordingly.
4. Connect the Model node and Comment Node (if available) to the DOM tree.
5. Transform the DOM tree into XML file using the user provided file name.

The above steps are common for CJC to LQML model or CJC to component sub-models.

3.4.1. Algorithm 3: Converting CJC to LQML

i. Create DOM Tree

ii. Create *Comment* node and add it to the DOM tree.

iii. Create a Model Node ("lqn-model"), M

iv. Retrieve the list of Processors(*processorList)* from CJC *Model* class

v. *for* (all Processors in *processorList*)

{

- Create a processor Node
- Get all the attributes from CJC Processor object

- Set all the attributes in Processor Node using the above information.
- Retrieve a list of Tasks available in this Processor (*taskList*)
- *for* (all Tasks in *taskList*)

  {
  - Create a Task Node, T*t*
  - Get all the attributes from CJC Task object
  - Set all the attributes in Task Node using the above information.
  - Retrieve a list of Entries available in this Task (*entryList*)
  - *for* (all Entries in *entryList*)

    {
    - Create an Entry Node, E*e*
    - Get all the attributes from CJC Entry object
    - Set all the attributes in Entry Node using the above information.
    - Set Ee node as a Child of the Task Node, T*t*
    - Retrieve list of Calls, *callList*[2], originating from this Entry
    - *for* (all Calls in *entryList*)

      {
      - Create appropriate call Node, C*c*.
      - Get the attributes from CJC-Call object and set them into C*c*.
      - Set Cc node as a Child of the Entry Node, E*e*

      }

[2] Strictly speaking, calls are in Phase object with the exception of *Forward* calls. In order to enable the algorithm flow at a high level, *callList* is assumed to have both phase and forward calls.

}//end of for entryList

- Retrieve a list of Activities available in this Task (*ActivityList*)
- *for* (all Activities in *activityList*)

{

- Create an Activity Node, A*a*
- Get all the attributes from CJC Activity object
- Set all the attributes in Activity Node using the above information.
- Set Aa node as a Child of the Entry Node, E*e*
- Retrieve list of Calls, *callList*, originating from this Activity
- *For* (all Calls in *callList*)

{

- Create appropriate call Node, C*c*.
- Get the attributes from CJC-Call object and set them into C*c*.
- Set Cc node as a Child of the Activity Node, A*a*

}

}//end of for activityList

}// end of for *taskList*

- Set Pp node as a Child of the Model Node, M

}//end of for processorList

vi. Convert the DOM tree into XML file using JAXP transformation factory libraries.

3.4.2. Algorithm 4: Converting CJC to CBML

i. Create a DOM tree

ii. Create a root Node named "lqn-submodel"

iii. Create a Comment Node, *Interface Node* and append them to the Root Node

iv. From CJC SubModelInterface class, get a list of In-ports (*inPortList*) and Out-Ports(*outPortList*) .

v. *for* (all Ports in *inPortList)*

{

- Create an in-port Node, INP*i*
- Get all the attributes from CJC-inPort object and set them to INP*i* Node
- Append INP*i* Node to *Interface* Node

}//end-for inPortList

vi. *for* (all Ports in *outPortList)*

{

- Create an out-port Node, OutP*i*
- Get all the attributes from CJC-outPort object and set them to OutP*i* Node
- Append OutP*i* Node to *Interface* Node

}//end-for outPortList

vii. Retrieve a list of Parameters (*paramList*) available in CJC SubModelInterface class

viii. Create a *Parameter* Node.

ix. *for (all the parameters in paramList)*

{

- Create a Para Node, Para*p*
- Get all the attributes from CJC-Parameter object and set them to Para*p* Node
- Append Para*p* Node to *Parameter* Node
-

}

x. Support for Bindings. Note: At the time of this research binding related tags were not established by the RADS research group. Thus, it's not supported in Editor.

3.5. CBML Creation, Modification and Visualization

Essentially, supporting CBML in Editor involves the ability to handle the information within the Interface elements of a DOM tree. In other words, architecture of the Editor must recognize the notations listed in Table 1. Since CBML is purely XML oriented, the following foundations are necessary in Editor to support CBML notations:

1. Extension to CJC to support classes representing CBML notation,
2. Parsers to read and write CBML notation,
3. Dialog boxes to view and modify CBML related information
4. GUI to visualize a sub-model

These foundations are elaborated in the coming sub-sections. Figure 18 depicts the architectural block diagram of CBML capable Editor and CBML aware CJC Framework. The dotted rectangle on the top of Figure 18 represents the Editor's GUI framework whereas the dashed lined rectangle in the bottom represents the CJC Framework. The boxes inside these rectangles represent different kinds of subsystems that make the frameworks. Self-explanatory arrows originating from Editor's boxes indicate the particular CJC subsystems they use.

Newly added support for CBML is represented by four shaded blocks: (1) CBML Dialog Boxes, (2) CBML View, (3) CBML Classes and (4) CBML-Parser. Internal details of these blocks are described in their respective sub-sections below.

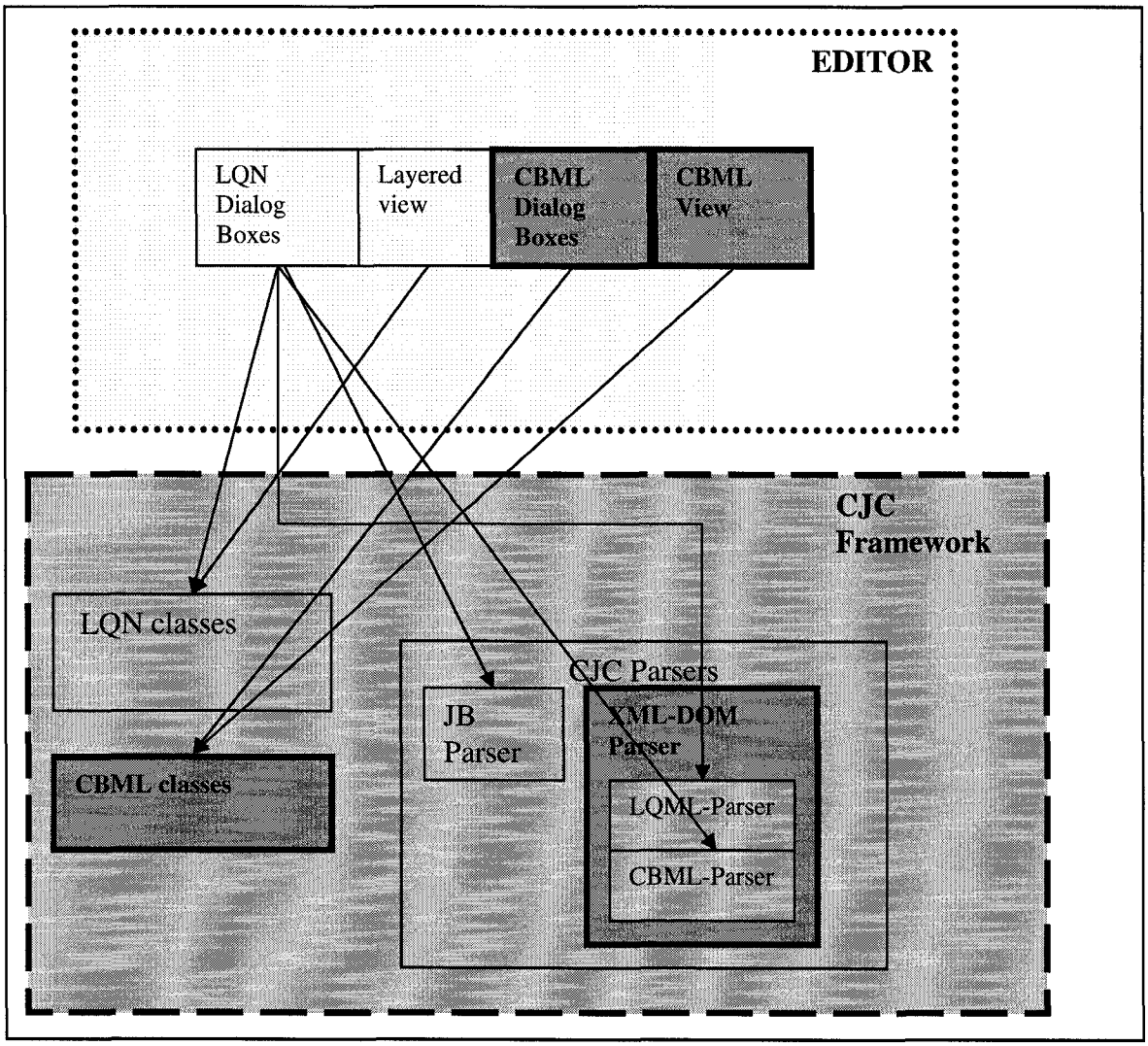


Figure 18 Block Diagram of Editor with introduced CBML related classes

3.5.1. Extending CJC framework for CBML

This section describes the internals of "CBML Classes" block depicted in Figure 18. The purpose of this block is to understand the information within Slot, Interface and binding by the CJC framework to support CBML. Thus, new Java Classes are introduced in CJC to handle CBML related notations. These classes are engineered to reflect the information of a Slot shown in Figure 15. However, they are not one to one mappings of XML tags represented by Figure 15. In the following, each new class is discussed in detail supported by their class diagrams.

- **Slot**

This class represents the Slot notation that holds sub-model interface and Binding information. Apart from utility attributes such as ID, it contains attributes to represent interface (name slotInterface) and collection of Binding Groups and Parameters. Each of these Attribute's data types, access methods and Class interaction with rest of the Editor components are clearly illustrated in the following class diagram.

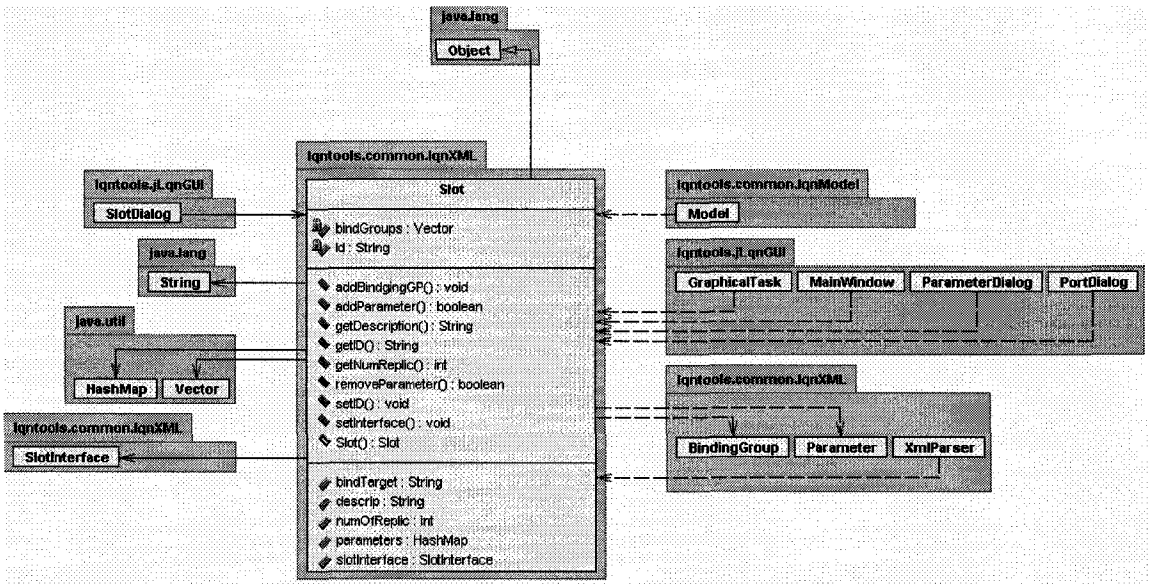


Figure 19 Class diagram: Slot

- **Interface**

Interface of a slot contains a collection of in-ports and out-ports that governs the connection between main model Entries and sub-model Entries. The SlotInterface class models the interface of a slot with list of in-ports and out-port attributes. Figure 20 depicts the self-explanatory class diagram for this SlotInterface Class.

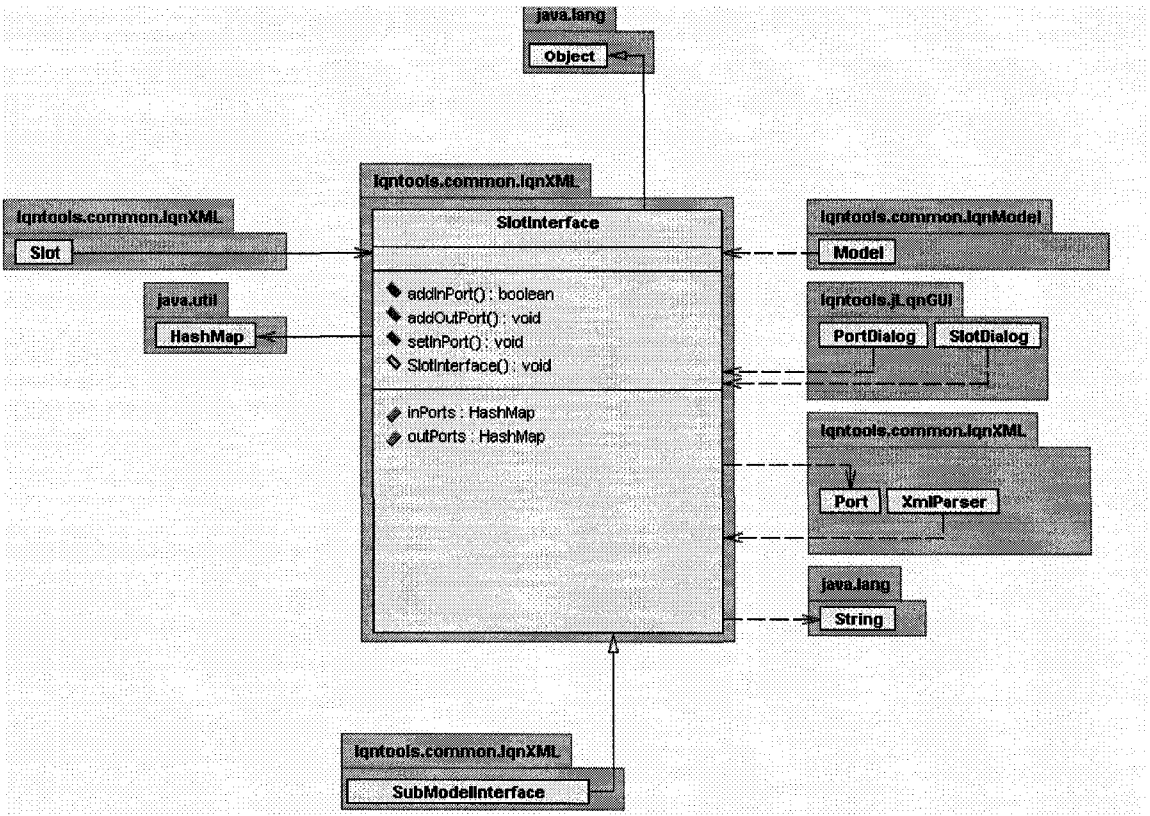


Figure 20 ClassDiagram: Slot Interface

- **Port**

As discussed earlier, there are two types of port notations in CBML (see Figure 15): (1) in-port and (2) out-port. They are differentiated by only one attribute, "connect-to" or "connection-from". In order to reduce the number of classes, rather than developing two separate classes to represent in-port and out-port, a common "Port" class is introduced. Any class that wishes to use a port class must be aware of how it wants to use this class. For example, in SlotInterface Class (see Figure 20), two separate port lists are maintained to hold in-port and out-port information. A Port class contains three attributes: (1) name, (2) description and (3) connection. The name attribute gives a name to the port and description attribute describe the purpose of the port. The third attribute, "connection" tells the name of the Entry that a port is connected-to (if it is in-port) or connected-from (if it is out-port). Figure 21 illustrate the class diagram of Port class.

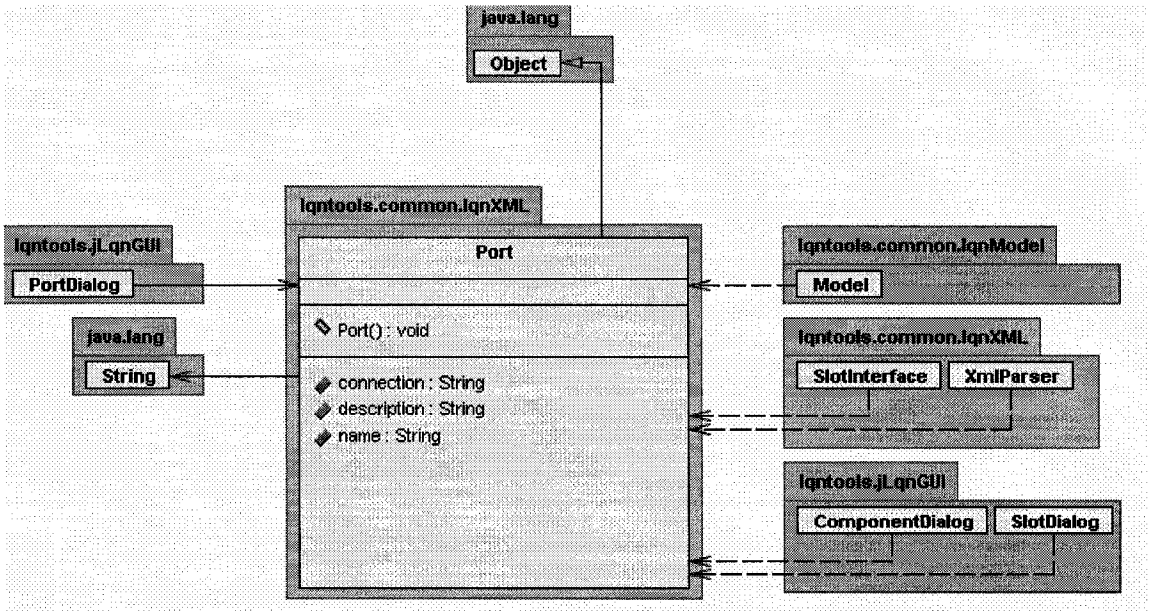


Figure 21 Class Diagram: Port

- **BindingGroup**

As its name implies, BindingGroup groups the binding information of a sub-model. It has parameter, port binding and processor binding lists and necessary utility methods. Figure 22 depicts the self-explanatory class diagram for BindingGroup.

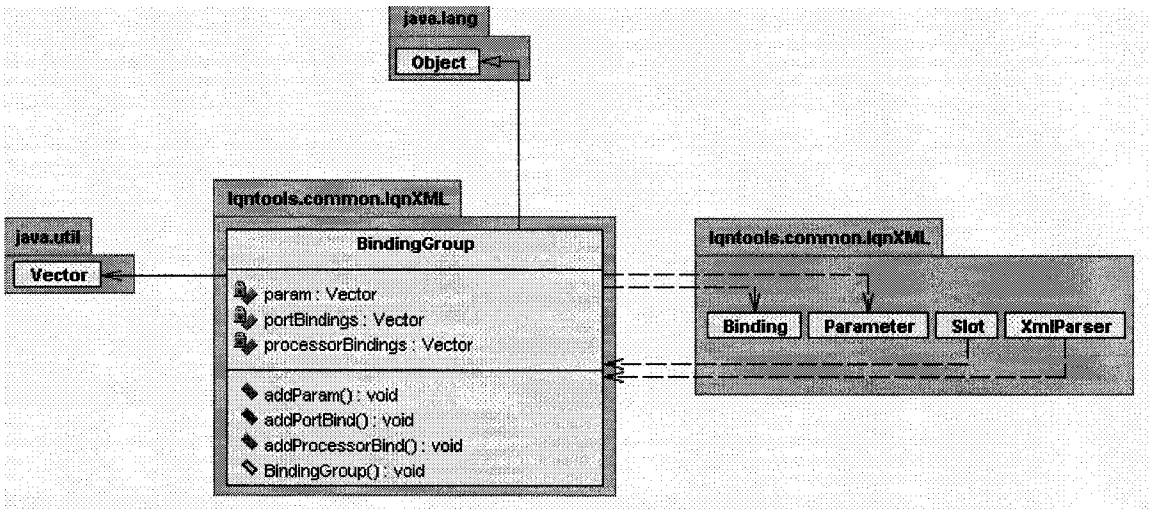


Figure 22 Class Diagram: BindingGroup

- **Binding**

This class represents the common binding information that can be used for Processor and Port bindings. It has source and target names of the bindings as its attribute. In its portBindings and processorBindings Vector, BindingGroup class will contain Binding class (see Figure 22). Figure 23 depicts the class diagram for Binding.

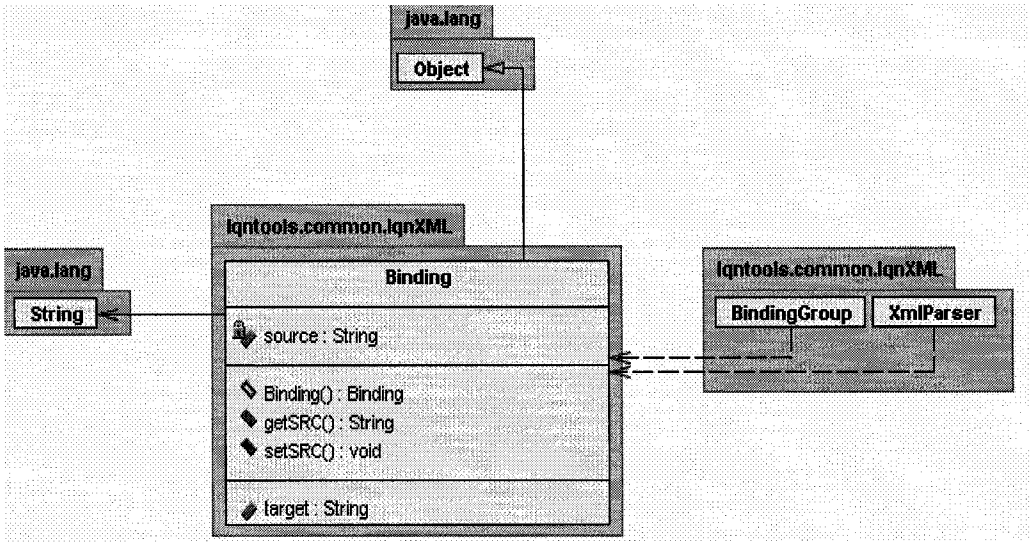


Figure 23 Class Diagram: Binding

3.5.2. Extending Main Window for CBML

In order to handle the CBML in main Window of the Editor, a "Slots" list is introduced with Add, Edit, and Delete buttons. This can be seen right side of Figure 24. If a main model includes one or many sub-model, they will be listed in Slots list. Just as for the Processors of a model, Slots can be added, changed or deleted using the available button. Note, although Slots have "Results" button as a result of reusing the GUI framework, it will not be active (indicating Results are not applicable for Slots).

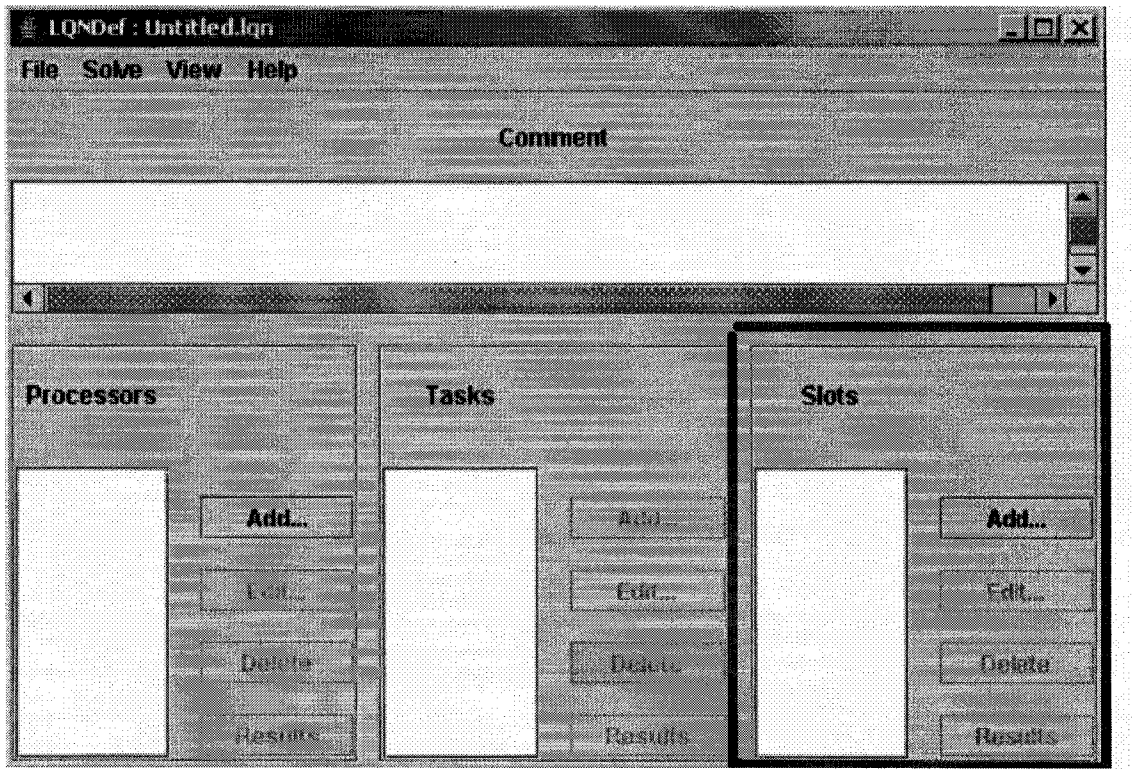


Figure 24 Main window of the latest Editor

3.5.3. Editor Dialog boxes for CBML

This section discusses the internal details of CBML Dialog Box block found in Figure 18. Dialog boxes are needed to create, modify and display CBML related information via Editor. There are four important Dialog boxes introduced that are discussed in the following:

I. Save as component

This dialog box allows a modeler to convert a normal LQN model into a component. In order to convert a model into component, the modeler needs ID (name of the component), in-ports, out-ports, parameters, and replaceable processors. Figure 25 depicts this dialog box with example values. It is named "Save as Component" and has in-ports, out-ports, replaceable processor and parameter lists. In addition, using this dialog box, existing component information can be modified or viewed.

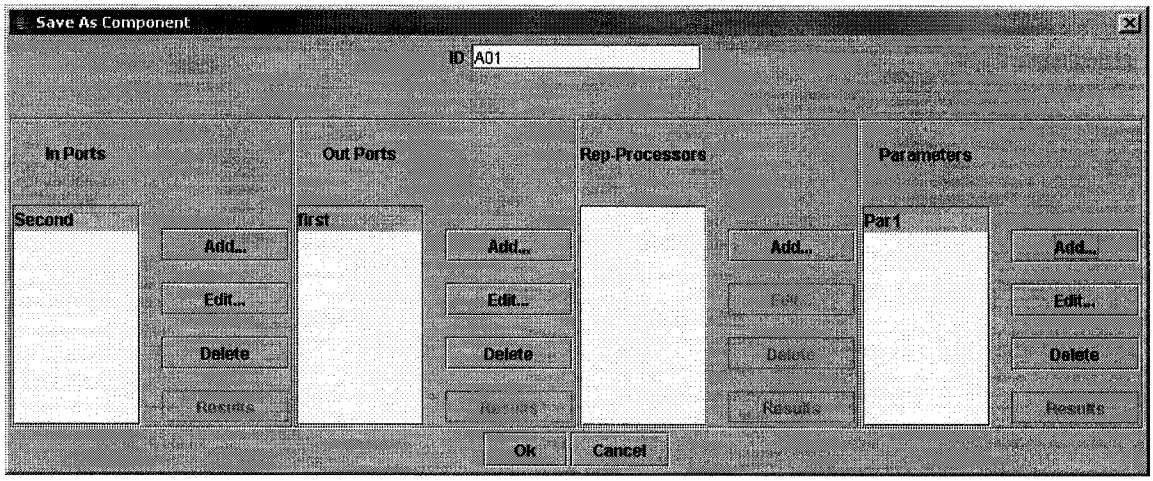


Figure 25Dialog box for saving a model as sub-model

II. Port Dialog

Port Dialog box enables one to create, modify or display in-port or out-port information of a component. It has two text fields to handle port comment and Port name. In addition, it has a pull down bar with available name of the Entries that this port is connected to (for in-port) or connected from (for out-port). Figure 26 depicts the port dialog box of a sample out-port.

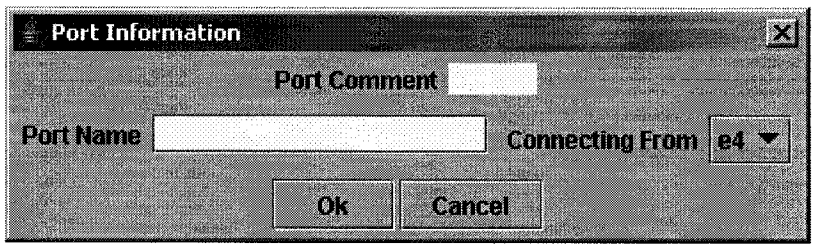


Figure 26 Dialog box for providing port information

III. Parameter Dialog

As depicted in Figure 27, the port dialog box has name and value text fields to display or change parameter information.

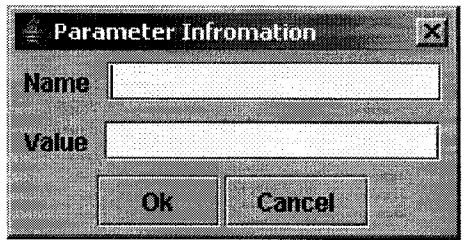


Figure 27 Dialog box for Parameter Information

IV. Slot Dialog box

The Slot Dialog box allows the user to create, modify or display a slot information. As seen earlier, the slot's name, in-ports, out-ports and parameters can be changed via this dialog box. Figure 28 depicts the slot dialog box.

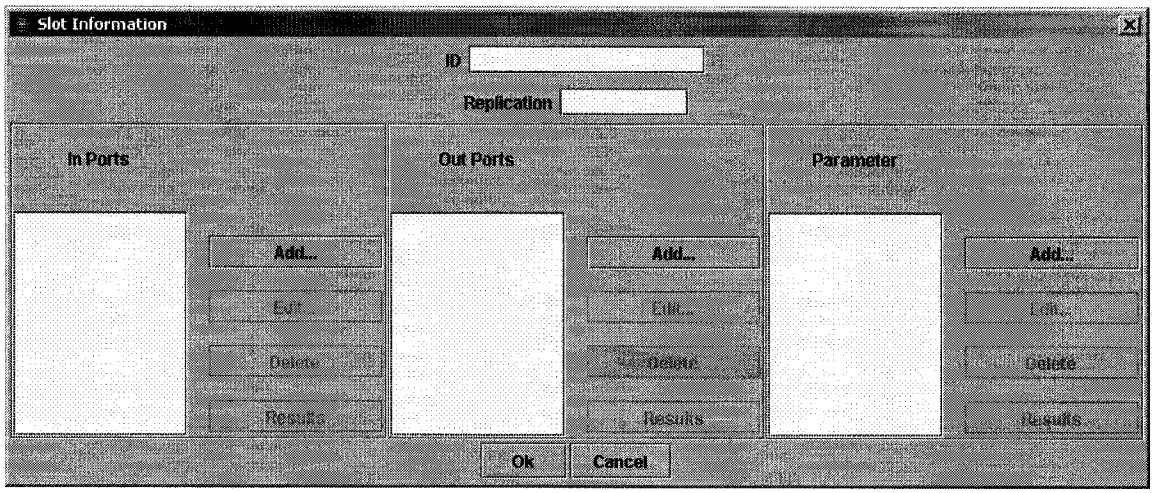


Figure 28 Dialog box to change Slot Information

3.5.4. Component Sub-Model Visualization

Visualizing a component sub-model is essential to see (1) how the ports are connected to the entries of the model and (2) the layered structure of the sub-model. Essentially, Editor should display a graphical representation of a sub-model similar to Figure 14. Since Editor is already capable of displaying layered view of ordinary models, enhancements were made as part of this research to handle Slot information in the graphical view. The "CBML View" block depicted in Figure 18 represents the details discussed in this section.

To simplify the design and reuse the existing graphical class, Slots are represented by pseudo tasks and Ports are represented by pseudo Entries. Thus, a sub-model in the

graphical view will have two pseudo Tasks and as many pseudo entries as ports. One pseudo Task will hold all the available in-ports and the other will hold the available out-ports in the form of pseudo Entries.

In order to mimic Figure 14 like layered view, in-port pseudo Task will be in the very first layer whereas the out-port pseudo Task will be in the very bottom. To avoid confusion with regular Tasks, pseudo Tasks will have unique names systematically assigned by Editor. In-port pseudo Task will have name "<sub-model name> + _INPorts_p" whereas Out-port pseudo Task will have name "<sub-model name> + _OUTPorts_p".

For example, sub-model depicted in Figure 29 had its sub-model named as "A01_slot". As you can see in Figure 29 , Layer 1 has the in-port pseudo Task named "A01_slot_INPorts_p"and Layer 5 (very last) has out-port pseudo Task named "A01_slot_OUTPorts_p". It is assumed that pseudo Tasks will contain only pseudo entries. Thus, maintaining a systematic name for a pseudo entry is not required. Figure 30 helps to associate Figure 29 with Figure 14 Graphical representation of an example CBML model [16p.27]. Note, the Layered Viewer does not show processors.

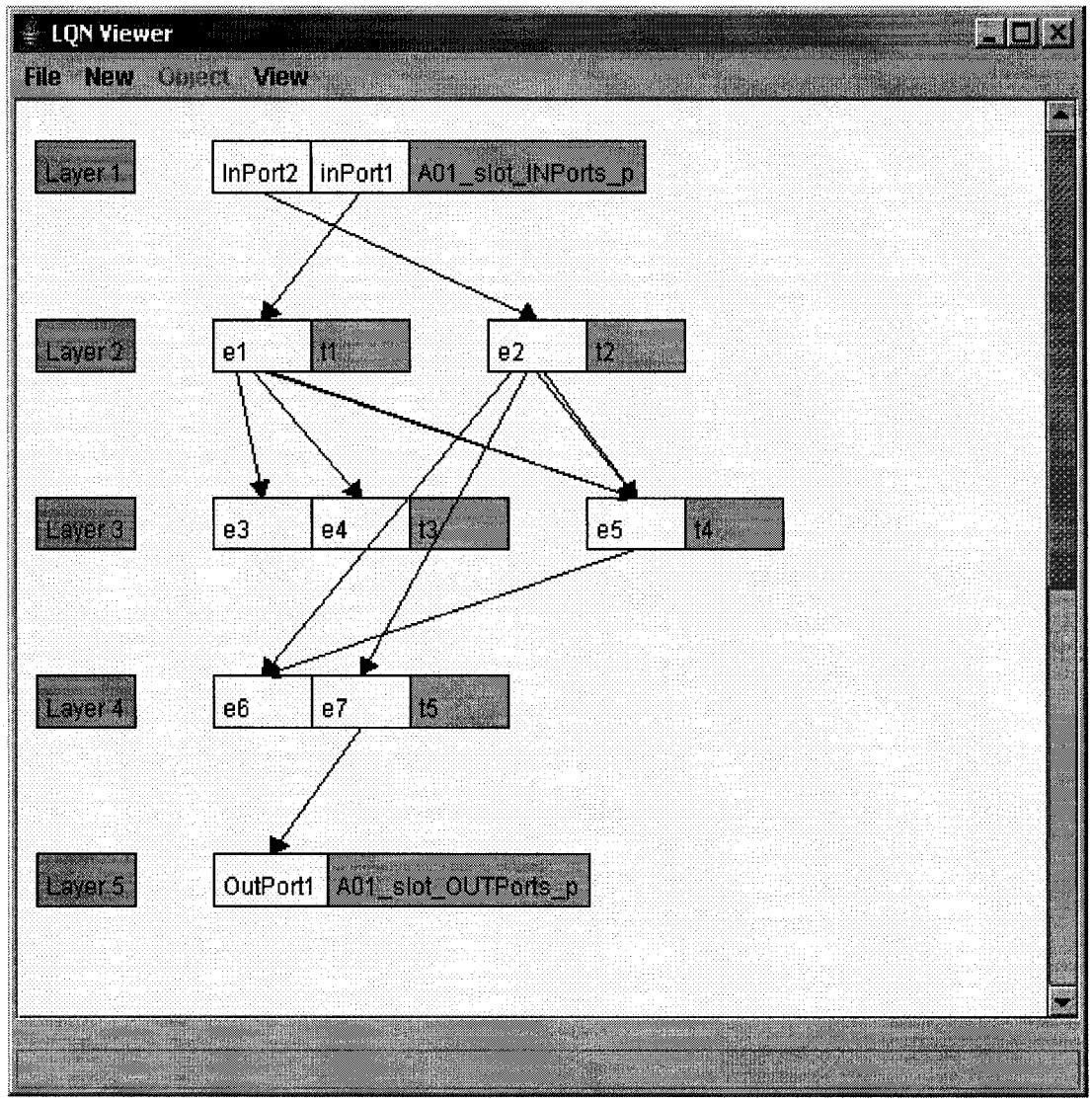


Figure 29 An example CBML layered view

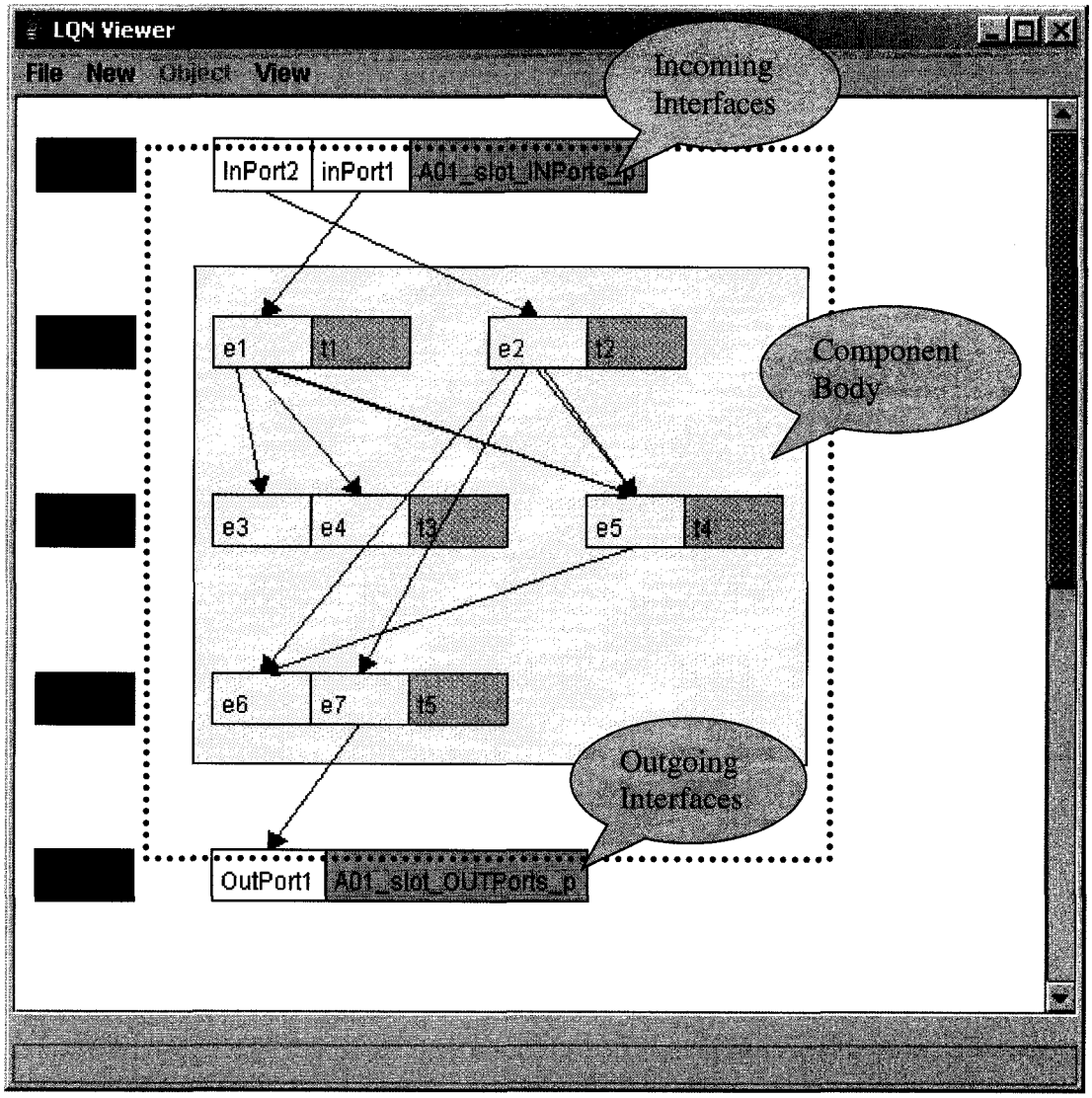


Figure 30Associating Figure 29 with Figure 14

Chapter 4. Identification of Software Bottlenecks with Saturation Breakdowns

Identifying and alleviating software bottlenecks in a system are crucial and complex aspects of SPE [8][39]. However, identifying it's location from the raw data is even difficult. When it is identified and root causes are clearly analyzed, appropriate mitigation techniques [15] can be applied to alleviate it. For example, adding resources at the bottleneck or shortening its service time are common mitigation techniques [39][41][42]. As discussed in the following sections of this chapter, it is important to realize that a software bottleneck cannot be eliminated. It can be either transferred to a less critical region or reduced its impact to an acceptable level. Thus, alleviating it in a system becomes an iterative process.

As mentioned earlier, a software bottleneck is a saturated server. A distributed system will have software bottleneck when a client spends excessive amounts of time waiting for the response from its servers. In other words, if a software task is fully utilized while the underlying resources are underutilized leading to system level performance constraints is known as software bottleneck [8].

The above mentioned characteristics of software bottleneck are well understood by SPE community. However, the process of (1) identifying, (2) analyzing and (3) iterative attempts to alleviate is very complex and tedious. In order to reduce the complexity and tediousness of this process, this research extends the Editor to (1) automatically identify the software bottleneck, (2) classify it and (3) visualize the location and saturation breakdown. This chapter provides the foundation for this research. Here, necessary theoretical background is presented followed by the detailed design of extending LQN Editor.

4.1 Software Bottleneck Strength Analysis

The concept of bottleneck strength and associated formula were introduced in Neilson [8] after the following observation and definitions:

Observation 1: Consider a task with utilization equal to 1 being saturated. A task is said to be a software bottleneck when it is saturated but its servers are not.

Observation 2: Saturated tasks or processors have a tendency to saturate or block its client tasks. Thus, the software bottleneck has the potential to spread saturation along the request arcs.

Definition 1: A software bottleneck occurs as a result of high task utilization respect to its servers' utilization. A server can be either direct or indirect.

Definition 2: Software Bottleneck Strength is a measurement of ratio of task utilization to the highly utilized server.

Definition 2 is governed by Equation 1:

$$B_b = \frac{U_b}{Max_{i \in \Theta}(U_i)} \qquad \textbf{Equation 1}$$

where B_b is the bottleneck strength, U_b is the utilization of candidate task and U_i is the utilization of its servers. Figure 31 depicts the utilization set Θ that is considered for bottleneck strength at Task b.

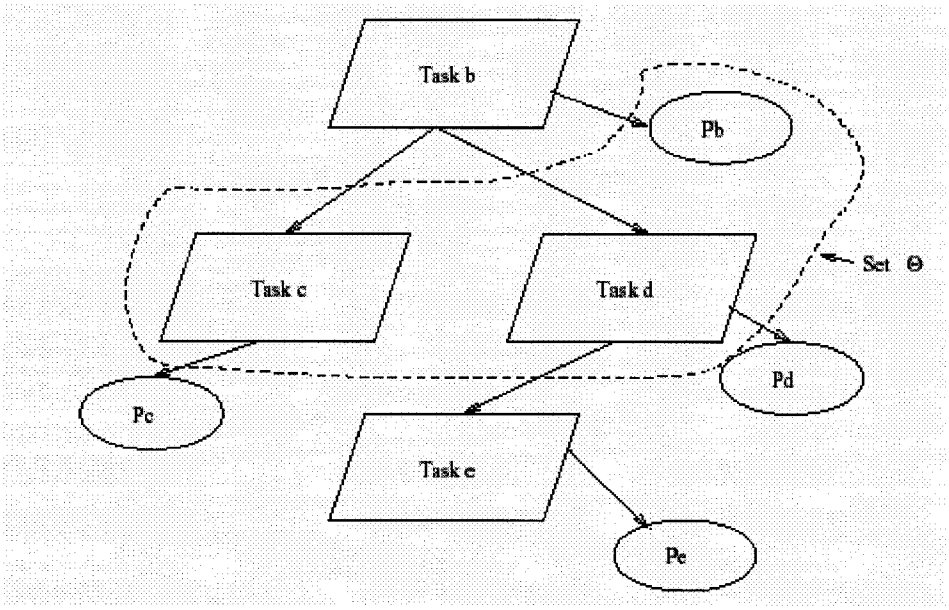


Figure 31 Set of Tasks, Θ taken from [15 p.31]

4.2 Software Bottleneck Detection Steps

Initially, Neilson in [8] developed definitions and mechanisms to informally identify software bottleneck through few examples. Tregunno in [15] formalized the detection by the following procedure:

1. Acquire utilization values for each task and processors in the system by analytic solution, measurement or simulation.
2. Discover the tasks that are saturated and form a set. As discussed in the introduction, a task is said to be saturated if its utilization is grater than the predefined threshold value, U_i
3. From the saturated tasks set in step 2, remove tasks that are pure servers because they cannot be a software bottleneck as they do not hold more than one resource [15].
4. For the remaining tasks in the set compute their bottleneck strength using Equation 1. The task that has the highest value is the software bottleneck.

4.3 Software Bottleneck Classifications

Once a software bottleneck is identified, it must be classified to select the appropriate mitigation strategies. Through experiments, [15]and [7] found two main variations: (1) server supported and (2) processor supported software bottleneck. Let U_θ be the utilization in the denominator of bottleneck strength equation. If U_θ is from a server task, it is server supported else if U_θ is from processor, it is processor supported. Tregunno in [15] introduces sub divisions to these servers and processors supported bottlenecks; declared as "Strong" or "Weak". Although quantifying strong or weak is arbitrary, [15] suggests strong is when bottleneck strength, B_b is grater than 2 (B_b >2), and weak if B_b is between 1 and 2 (1 < B_b <2). For example, in Figure 32 depicts a strong server supported bottleneck at task named COCO. Set Θ for COCO task is {H248IP, DB, SS7OutputIP, CPU_COCO}.

From the numerical value of the set Θ ({0.0885, 0.0206, 0.0550, 0.0838}), it is evident H248IP has the highest utilization. Thus using Equation 1, the bottleneck strength of COCO is11.2994:

$$BNS_{coco} = \frac{1.00}{0.0885} = 11.2994.$$

Using the same tasks, Figure 33 depicts an example weak server supported bottleneck case by changing the utilization of DB.

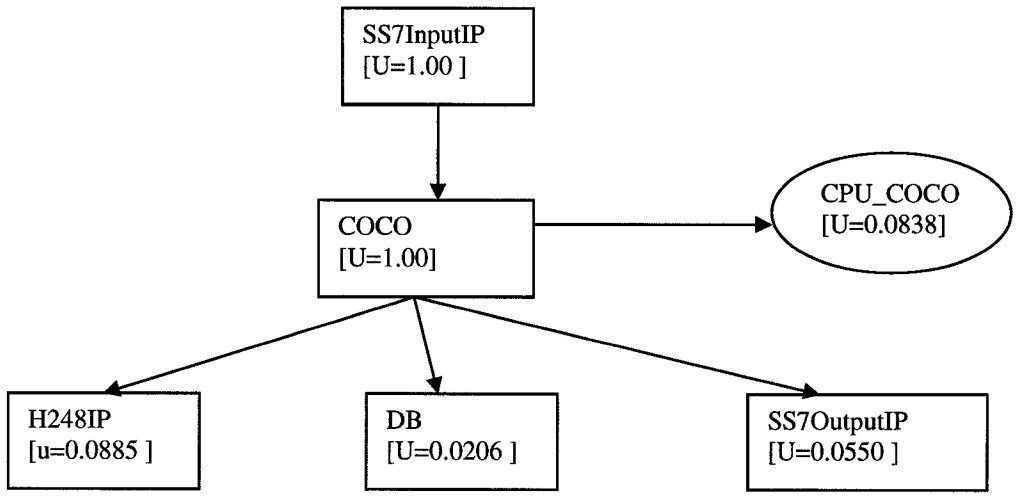


Figure 32 Example of Strong Server Supported Bottleneck

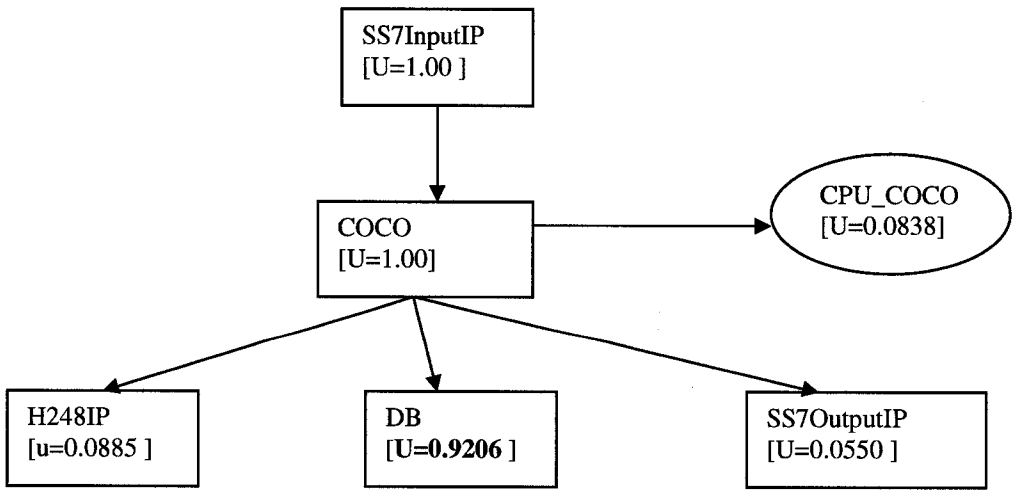


Figure 33 Example of Weak Server Supported Bottleneck

In summary, based on the origin of U_θ and value of B_b, software bottleneck can be classified into the following four classes:

1. Strong processor supported,
2. Weak processor supported,
3. Strong server supported and
4. Weak server supported.

4.4 Software Bottleneck Mitigation

Once a software bottleneck is discovered and classified, it must be mitigated to increase the overall system performance. Based on the bottleneck strength formula, mathematically, the bottleneck can be mitigated by increasing U_θ or decreasing U_b. Although this is not a straight forward task to accomplish, the following four strategies are proven to be successful [15]:

Strategy 1: Increase the number of threads at bottleneck task, T_b.

The important question to ask before trying this strategy is: how many threads will mitigate the bottleneck at T_b. According to [15], this technique has an upper limit on T_b's throughput, f_b. The upper limit is known to be $f_b \times B_b$. The disadvantage of this technique is that it will increase processor contention. Negative performance impacts of multithreading is a well known phenomenon in computer systems. In addition, it is important to note that this will possibly cause the server tasks to become bottlenecks [15]. Thus, this technique is suitable for strong server supported bottlenecks.

Strategy 2: Replication of T_b and its processor, P_b.

Addition of processors is beneficial in the case of a weak processor supported bottleneck. In this class of bottleneck, T_b is blocked waiting for processor time. Before attempting processor replication in a model, the modeler should consider the practical difficulties involved with it such as the need for hardware and firmware modification.

Strategy 3: Dropping processor demand for phase 1 of T_b.

This strategy attempts to gain performance by reducing execution demand to improve U_b. A well known mantra for this is to develop "faster code" to reduce execution time. However, this can be a suggestion by the modeler and cannot be guaranteed since modeling occurs at the architecture phase of the software development cycle. Obviously, this technique is not appropriate for server supported bottlenecks as they are the consequence of waiting for response from lower layer server. This can be useful for processor supported bottlenecks.

Strategy 4: Decreasing the calls T_b makes to lower layers

This technique reduces the waiting time for a resource by aggregating rendezvous calls. As seen earlier, waiting for resources are the root cause for software bottleneck. Thus, "batching" the calls will reduce the waiting time experienced by T_b and decrease U_b. After mathematical analysis, [15] concludes that batching rendezvous are effective when T_b is a weak server supported bottleneck.

4.4.1 Algorithm for Software Bottleneck Mitigation

1. Identify the software bottleneck task, T_b using the performance metrics from either simulation or analytical solution.
2. Determine the software bottleneck class. Four such classes are discussed above.
3. Choose appropriate strategy based on the bottleneck class.
    a. If (class == Strong Server or Processor supported)
        i. Try *Strategy 1*, increasing the threading level.
        ii. Go to step 4.
    b. Else if (class == weak processor supported)
        i. Try *Strategy 2*, replicate processors that T_b uses
        ii. Or try *Strategy 3*, reduce T_b execution time.
        iii. Go to step 4.
    c. Else if (class == week server supported && T_b has many rendezvous)
        i. Try *Strategy 4*, batch process the rendezvous from T_b
        ii. Go to step 4.

4. Solve the modified model and obtain new set of performance metric. Check if the bottleneck is mitigated. If the mitigation result is not satisfactory or it has migrated to a different task that is sensitive to bottleneck, go to step 1.

4.5 Introduction to Saturation Breakdown

Saturation Breakdown is the service time of a task broken down into three parts:

1. CPU utilization fraction,
2. call-blocked time fraction and
3. idle time fraction of a Task.

Visualizing and understanding this breakdown at a bottleneck task is beneficial to realize the root causes for the saturation. Once the root causes are identified, classification of bottleneck becomes easy. Therefore, appropriate mitigation can be applied to the bottleneck task to alleviate the saturations - the focal point of SPE.

The fraction of CPU utilization ($Fcpu_p$) and blocked time (Fcb_p) are cumulative over the Phases within the Entries of a Task. Note that $Fcpu_p$ and Fcb_p has index p representing a Phase. Figure 34 depicts saturation breakdown at an abstract level. This abstract level breakdown serves as a guideline to the reader for more detailed descriptions and to derive the mathematical equations. In addition, using this information, one can clearly see where a task spends most of its time plus its workload status. For example, if a task has idle fraction much higher than its CPU utilization and call-blocked time fractions, it indicates the capacity to handle more workload. On the other hand, having idle fraction much less than CPU utilization and call-blocked fraction indicates potential bottleneck.

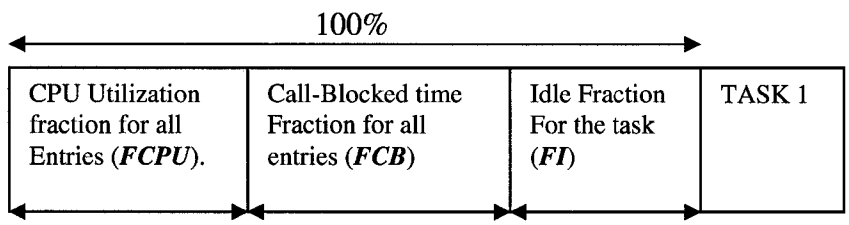


Figure 34 Abstract level saturation breakdown

In order to establish mathematical equations for fractions shown in Figure 34, the following general definition is used:

The fraction for any set *OP* of operation is

$$F_{OP} = \sum f_i \times x_i \qquad \textbf{Equation 2}$$

where f_i is the frequency of one operation in set *OP* and x_i is the mean duration of one operation in set *OP*.

Since sum of CPU utilization, call-block and idle fractions adds to one, idle fraction (*FI*) is given by:

$$FI = 1 - (FCPU + FCB) \qquad \textbf{Equation 3}$$

$$= 1 - (\sum_{p=1}^{n} Fcpu_p + \sum_{p=1}^{n} Fcb_p) \qquad \textbf{Equation 4}$$

Note that in LQN syntax, phases are contained in entries. In the above equation, all the phases (1...n) from all the entries of a task are considered together for simplification. Figure 35 correlates this equation to the abstract level fractions depicted in Figure 34.

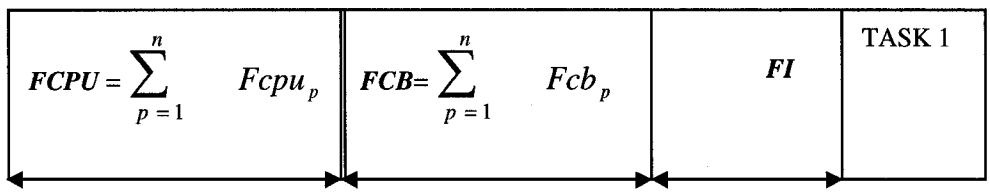


Figure 35 Abstract level faction of a Task

4.5.1 Step-By-Step Approach

The abstract level breakdown seen above cannot depict the individual contribution from each Entry, Phase or Call of a task. In order to understand the root cause of the bottleneck clearly, it is important to see what exactly contributes to higher fractions (CPU utilization or call-blocked time). To do this, one must zoom into each entry, phase and call of the Task. In the following, author explains the process of zooming in through step-by-step diagrams. This step-by-step approach serves two purposes: (1) to derive mathematical equations and (2) to help the visualization of the breakdown.

Step 1: Looking inside each entries

As seen earlier in the background chapter, an Entry can have maximum of three phases; and each phases can have many number of calls. As a first step, zoom into the entries of the Task and look at their phases. Figure 36 illustrates two Entries and five phases of a Task. Note Entry1 has three phases whereas Entry2 has two phases.

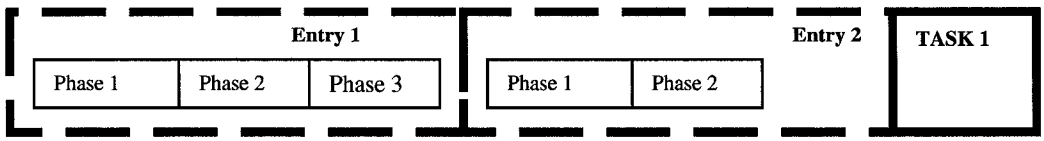


Figure 36 Phases of an Entry

Step 2: Looking at the calls made from each entries

The second step is to zoom inside a particular Entry of the task to see where the calls are destined. For example in Figure 37, we consider only Entry1 of task TASK1 which has three phases. Phase1 has two calls destined to E1 of task T2 and E5 of task T3. Phase2 also has two calls destined to E2 and E3 of task T2. Phase3 has only one call destined to E4 of task T2.

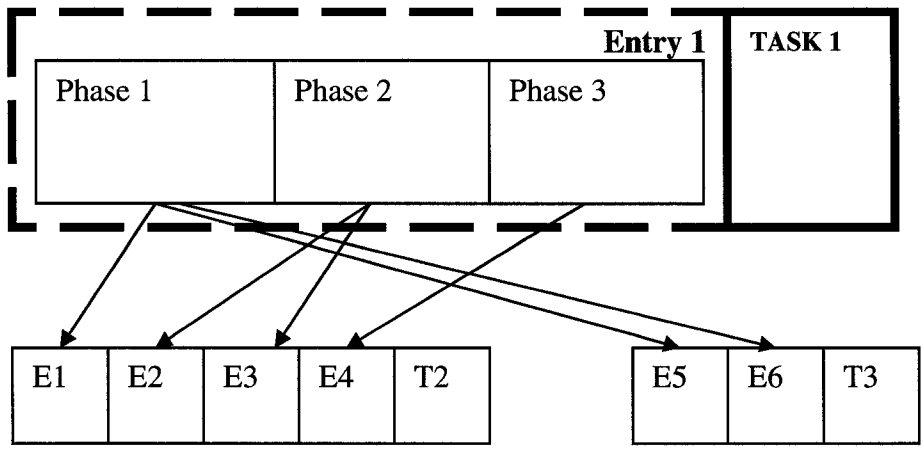


Figure 37 Phase of an Entry with calls

Step 3: Looking inside a phase

The third step is to focus into a particular phase of an entry. This allows seeing the details of the calls inside a phase. Figure 38 illustrates the Phase1 of Entry1 from Figure 37 with three calls:

1. Call C1, makes y1 mean number of calls to E1 of task T2,
2. Call C2, makes y2 mean number of calls to E5 of task T3 and
3. Call C3 makes y3 mean number of calls E6 of task T3.

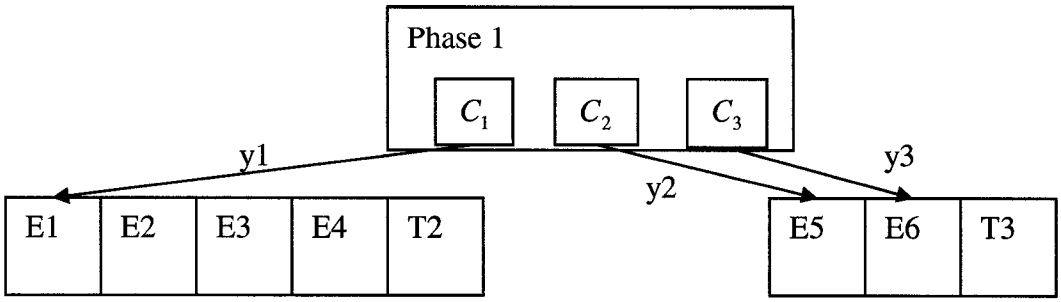


Figure 38 Phase and the calls

The information in Figure 38 is the focal point of the saturation breakdown. Each phase has its own CPU utilization fraction and call-blocked time fraction. Call-blocked fraction, Fcb_p, of a phase is the sum of the call-blocked fractions experienced by its calls. Thus it is given by

$$Fcb_p = \sum_{i=1}^{n} \alpha_{ep,i} \qquad \textbf{Equation 5}$$

where the index e represents the destination Entry (e= 1..m), i represents call number (i= 1…n) and p represents the phase (p =1..3).

The fraction of a call-blocked ($\alpha_{ei,p}$) experienced by a Phase is given by

$$\alpha_{ep,i} = U_p \times \frac{y_{ep,i} \times b_{ep,i}}{X_p} \qquad \textbf{Equation 6}$$

where y is the mean number of calls made to a particular destination entry, b is the mean blocking per call, X_p is the mean service time and U_p is the utilization of a phase. Note that U_p / X_p is the throughput (f) of the phase. The mean blocking time per call is

$$b_p = t_r + t_s \qquad \textbf{Equation 7}$$

where t_r is the rendezvous delay to destination entry and t_s is the phase one service time of a call. Figure 39 depicts graphical representation of CPU utilization and call-blocked fractions for Phase 1 of Figure 38.

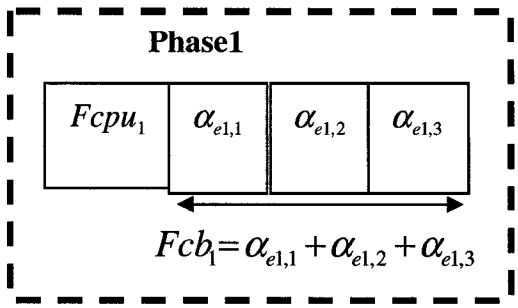


Figure 39 CPU utilization and Call-Blocked time fractions of Phase 1

Step 4: Putting it all together and visualizing saturation breakdown

Figure 40 depicts the saturation breakdown of Task1 by combining information from Figure 36 to Figure 39 discussed in earlier steps. Note that only Phase1 and Phase2 of Entry1 are shown for simplicity. Also, assume that the fraction values for rest of the Entries and Phases of Task1 are smaller. Phase 1 and phase 2 show their execution times ($Fcpu_p$) and call-blocked time ($\alpha_{ep,i}$) fractions. Here, author introduces a convention that the width of the rectangle representing $Fcpu_p$, $\alpha_{ep,i}$ and *FI* are proportional to their numerical values. Thus, saturation breakdown can be easily interpreted using the size of the rectangles. For example, according to Figure 40, Task1 spends most of its time blocked on call $\alpha_{e1,3}$ of Phase1. Furthermore, it indicates CPU utilization fraction for Phase2 is higher than Phase1. Similar rationalization for the remaining rectangles will provide sufficient insights about the service time breakdown for Task1. Using such insights will help one to propose a suitable mitigation strategy for any bottleneck task. For example, one can increase the idle fraction, *FI* for Task1 by reducing $\alpha_{e1,3}$ of Phase1 using the mitigation strategies discussed earlier.

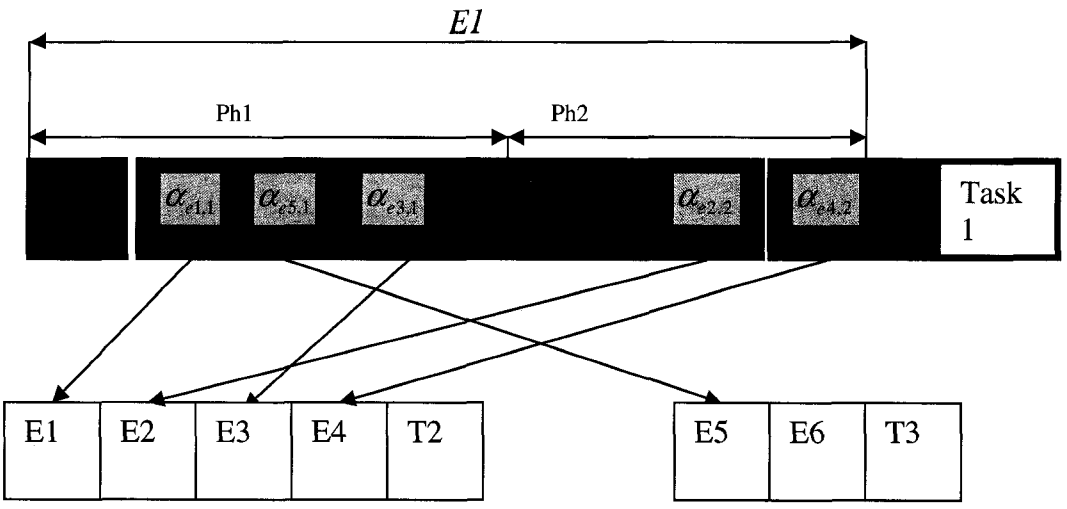


Figure 40 Service time breakdown of a Task

4.5.2 Challenges

Since bottleneck mitigation is a repetitive process, manual computation of these fractions becomes tedious and error prone. Apart from the tediousness, manual computation of saturation breakdown becomes impossible for large models. Imagine a model with fifty tasks, one hundred and fifty entries and four hundred calls. It is impractical to manually read the result values for each calls and phases to produce the saturation breakdown for all the tasks. To avoid these issues, this research presents an approach to automatically compute and display saturation breakdown using the Editor (discussed in section 5.2).

Chapter 5 Implementation of Visualization

The visualization support in the Editor for bottleneck identification and saturation breakdown includes several functionalities:

1. load results of a solution in a form usable by the display,
2. filter and obtain bottleneck related result values,
3. execute algorithms to compute bottleneck strength and saturation breakdowns and
4. Construct the visualization of all the tasks.

Implementation details of these functionalities are discussed in the following subsections.

5.1 Software Bottleneck Identification and Categorization

This research enables Editor to identify and classify software bottlenecks by (1) retrieving and storing result values from LQML models, (2) executing algorithm to traverse through Tasks, Entries and Calls to compute BNS (Bottleneck Strength) equation and (3) modifying the Task Dialog box to display BNS related information.

These points are discussed in detail in the following sections:

- **Retrieving and Storing Result parameters in CJC objects**

Software bottleneck identification and categorization can only be possible if Editor is capable of reading the results of a solved model. Thus, as a first step, a mechanism to read and store the result parameters for each LQN model element was provided. This mechanism (1) reads LQML files with results, (2) parses it and (3) stores the DOM *result-node* in respective CJC object. Results are stored as raw DOM elements since Editor will only do read operation, not modification. All LQN nodes such as Processors, Tasks, Entries, Activities, Calls and Phases must have this *result-node* as an attribute. Since these nodes are subclasses of *ModelNode* classes, *result-node* attribute and associated methods are introduced in ModelNode class. This is depicted in the class diagram (Figure 42).

It is important to note that *result-node* contains a collection of standard performance output parameters. If an output parameter is not applicable to a LQN node, it will have a default value in it. So, its Editor's responsibility to dynamically select suitable attributes from each node. At the same time, Editor is designed to support *result-nodes* with 99% and 95% confidence intervals.

- **Algorithm 5: Software Bottleneck Identification and Classification**

This algorithm is responsible for retrieving a Task's utilization, its server tasks' utilization and its processor's utilization. Once these sets of utilization are available it will execute the BNS equation and classify the bottleneck based on the discussions from section 4.3 to 4.5. Again, assume that T_b is the target task for which we want to know the bottleneck strength.

1. Check if Task T_b has entries and results available
2. Get the list of Entries in T_b, *entryList*
3. for (all Entries in *entryList*)
   {
   a. Get the list of Call-Outs, *callList*
   b. For (all Synchronous calls in *callList*)
      {
      i. Get the destination Task, T_{dest}
      ii. From T_{dest}, get its utilization and name and add them destination task hash map.
      }
   }//End of For: Entries
4. Check if T_b has Activities. If yes, retrieve them and form an *activityList*.
5. for(all Activities in *activityList*)
   {
   a. Get the list of Call-Outs, *callList*
   b. For (all Synchronous calls in *callList*)
      {

i. Get the destination Task, T_{dest}

ii. From T_{dest}, get its utilization and name

iii. if (T_{dest} not present in hash map, $HM_{T_{dest}}$)

- Add them to destination task hash map.

}

}

6. from $HM_{T_{dest}}$ identify the Task with highest utilization, $U_{T_{dest}}$
7. from T_b, get the process utilization, U_P
8. if($U_{T_{dest}} > U_P$)
   a. Set bottleneck classification as "server supported"
   b. Set $U_{\max} = U_{T_{dest}}$
9. if ($U_P > U_{T_{dest}}$)
   a. Set bottleneck classification as "Processor supported"
   b. Set $U_{\max} = U_P$
10. calculate bottleneck strength: $BNS = U_{T_b} / U_{\max}$
11. if ($1.0 < BNS \;\&\&\; BNS < BNS_{threshold}$)
    a. Set bottleneck type as "weak"
12. if ($BNS_{threshold} < BNS$)
    a. Set bottleneck type as "strong"

- **Displaying Bottleneck related attributes in Task Dialog box**

Particular Task related information can be viewed in Editor using the Task Dialog box. This dialog box can be accessed from the main window or from double clicking on any task rectangular box from the layered view window. This research extended the Task Dialog box to incorporate software bottleneck related information. When this dialog box is launched, it will dynamically execute Algorithm 5 to compute bottleneck strength, type and classification. It will then populate these values to the respective text fields in the top

right hand corner (see Figure 41). The following indications are provided to get a user's attention:

1. The bottleneck information has a status field that indicates the status of the presented information as valid or not valid. The status will become invalid when there are Tasks or processors that do not have a result-node present in the model. In such cases, the model will have to be solved again. This situation may also arise when a solved model with results have been modified (i.e. adding new Task or Processor). If the status is invalid, bottleneck related information should be ignored.
2. When the software bottleneck type is strong (i.e. BNS >2 according to [15]), the text field color will changed to red and the string "STRONG" will be bold. This gives indication to the modeler that this Task as a candidate for mitigation.
3. In the absence of *result-node* for this task, its server tasks or processors, the bottleneck information will be set to the default values are listed in Table 5.

Attributes	**Default value**
BNS value	**0.0**
Status	**INVALID**
Type	**NONE**
Classification	**UNKNOWN**

Table 5 Default value for bottleneck information

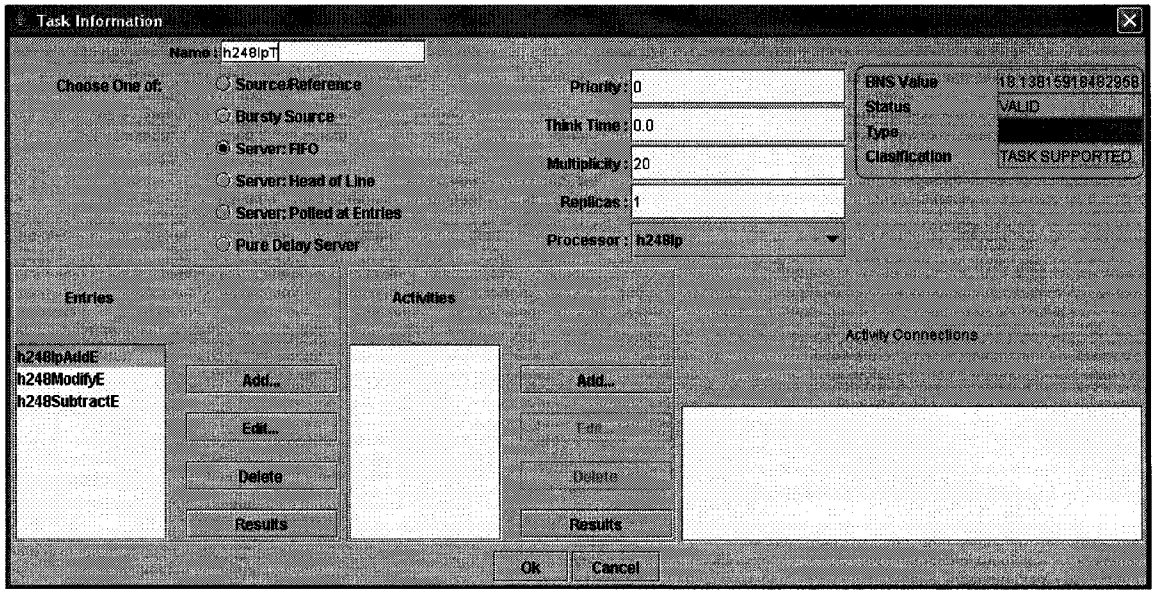


Figure 41 Task Dialog box with Bottleneck Information

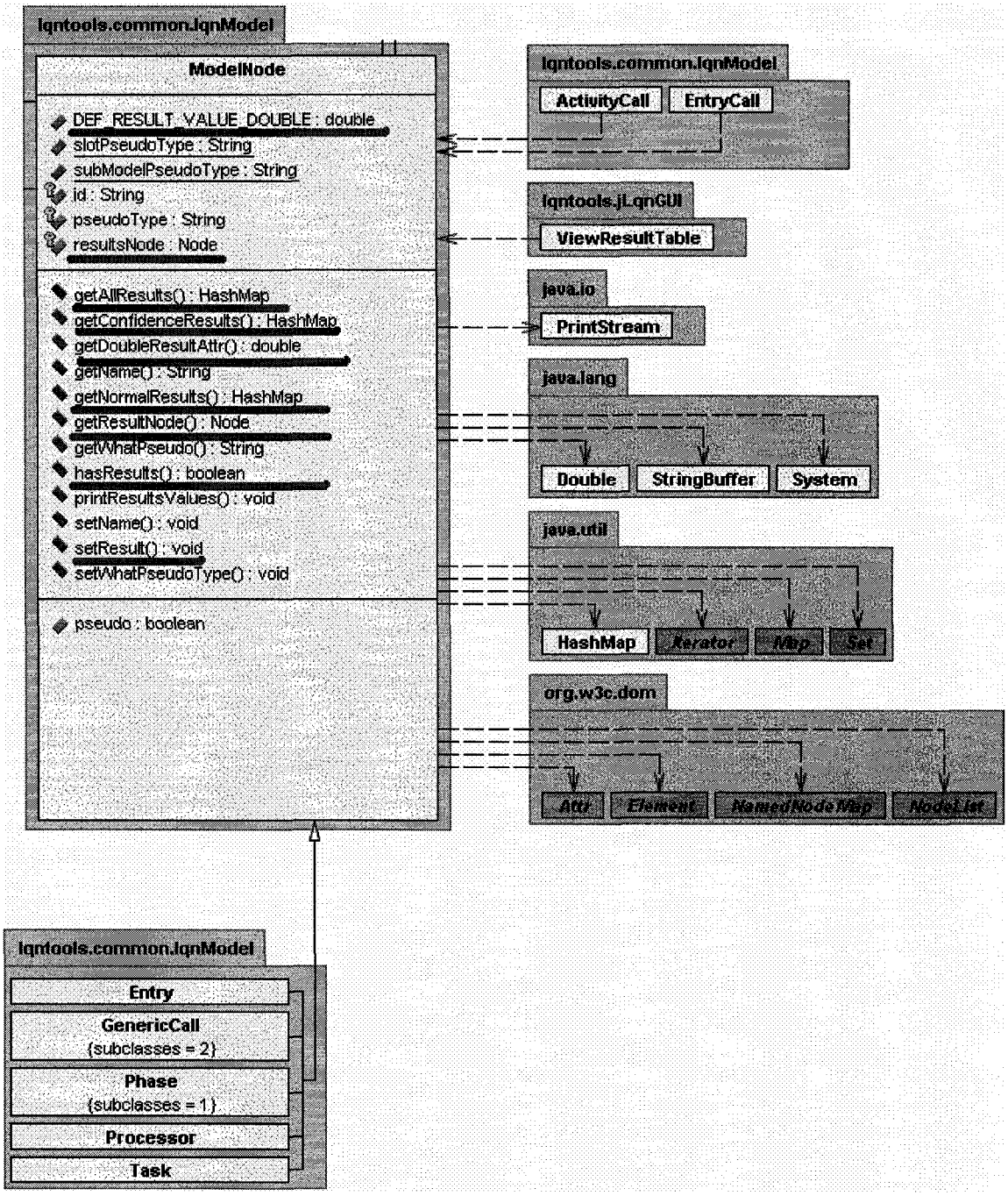


Figure 42 Class Diagram: ModelNode

5.2 Software Bottleneck and Saturation breakdown Visualization

5.2.1 Significance of software bottleneck visualization

Generally, once a LQN model is solved, results are stored in a text file; the modeler must manually fetch desired result values and compute the BNS and associated values. This process is cumbersome and error prone. In addition, the modeler must

create a layered picture of the system either manually or by tools and propose a bottleneck migration. Being able to visualize bottleneck and saturation breakdown of a system provides intuitive understanding of the bottlenecks. Using the architecture of the Layered View, BNS algorithm and Saturation breakdown algorithm from this research automates the software bottleneck visualization. Hereafter, this is referred as the Bottleneck View.

The Bottleneck View displays all the Tasks of the model and their calls to the respective server tasks in a layered manner. It shows the bottleneck strength of a task in red. The width of the reddishness in a Task rectangle demonstrates how strong the bottleneck is. If a Task's rectangle is all red, that indicates that it is a bottleneck. If it is located in a performance sensitive area of the system, a modeler can pay extra attention to it. By using the saturation breakdown capability, a modeler can easily zoom in to a bottleneck and investigate the root cause. Based on the result of this investigation, an appropriate mitigation strategy can be applied. Mitigation that involves changing the parameters of a model is made easy by Editor Dialog boxes. This process of solving a model, visualizing bottleneck and applying mitigation is repeated until a suitable solution is found. Automating such process adds value by saving time and eliminating the complexity and errors.

5.2.2 High Level Design Requirements

- In a separate window, the user must be able to visualize a Layered view consist of only Tasks and Calls. This window should be named as "BNS View".
- In BNS view, user must be able to visualize and identify Tasks that are potential bottlenecks of a system.
- On a particular Task in BNS view, the user must be able to expand and see the service time break down. To achieve this, interested Task's Entry, Phase and Calls must be displayed.
- For a particular Task from BNS view, the user must be able to open its Task information dialog box.
- User must be able to launch a bottleneck view window from the "View" menu bar of the Editor.

- Legacy Layered and bottleneck views must be able to run concurrently.

5.2.3 Detailed Design of Bottleneck View

Design fundamentals of the Bottleneck View are similar to that of the Layered View. The major difference is that bottleneck view has two levels of display:
(1) Abstract BNS display - showing only the tasks and their calls with reddish rectangles to indicate bottleneck strength and
(2) Detail BNS display- zooming into a particular task to display its Entries, phases and calls with service time breakdown. Layers are formed based on the calls going out of tasks.

The abstract BNS display, referred to as first level visualization hereafter, is designed to show all the potential bottleneck tasks in a system. It enables a modeler to quickly scan through the layered model for potential bottlenecks by looking at the reddish task rectangles. Depending on the bottleneck strength, a graphical Task can be full (very strong) red or partial red.

The detail BNS display, referred to as second level visualization hereafter, is designed to provide more insight into a selected Task by displaying service time breakdown. If there is a potential bottleneck in the selected Task, the modeler can gain insight using the saturation breakdown. Acquiring this insight is vital for proposing an appropriate mitigation strategy.

In order to achieve the above mention design, this research introduced four new classes to GUI package of the Editor: (1) BNSView, (2) BNSDrawArea (3) GraphicalPhase and (4) PhaseCallFractionBox. The following subsections are dedicated for detailed description of these classes and their interaction with rest of the CJC and GUI classes.

- **BNSView Class**

When a user launches the bottleneck view from the main window, an instance of BNSView will be instantiated to provide the frame for bottleneck display. This frame, a subclass of JFrame class available from JAVA SWING components, listens for standard user events. Inside this BNSView, a BNSDrawArea (discussed in a subsequent section) containing layered Tasks and calls with bottleneck information will be populated. It also

has a backward reference to the MainWindow to collect necessary model information and to forward it to BNSDrawArea object. Figure 43 depicts the class diagram for BNSView. All the attributes and methods in Figure 43 are self-explanatory.

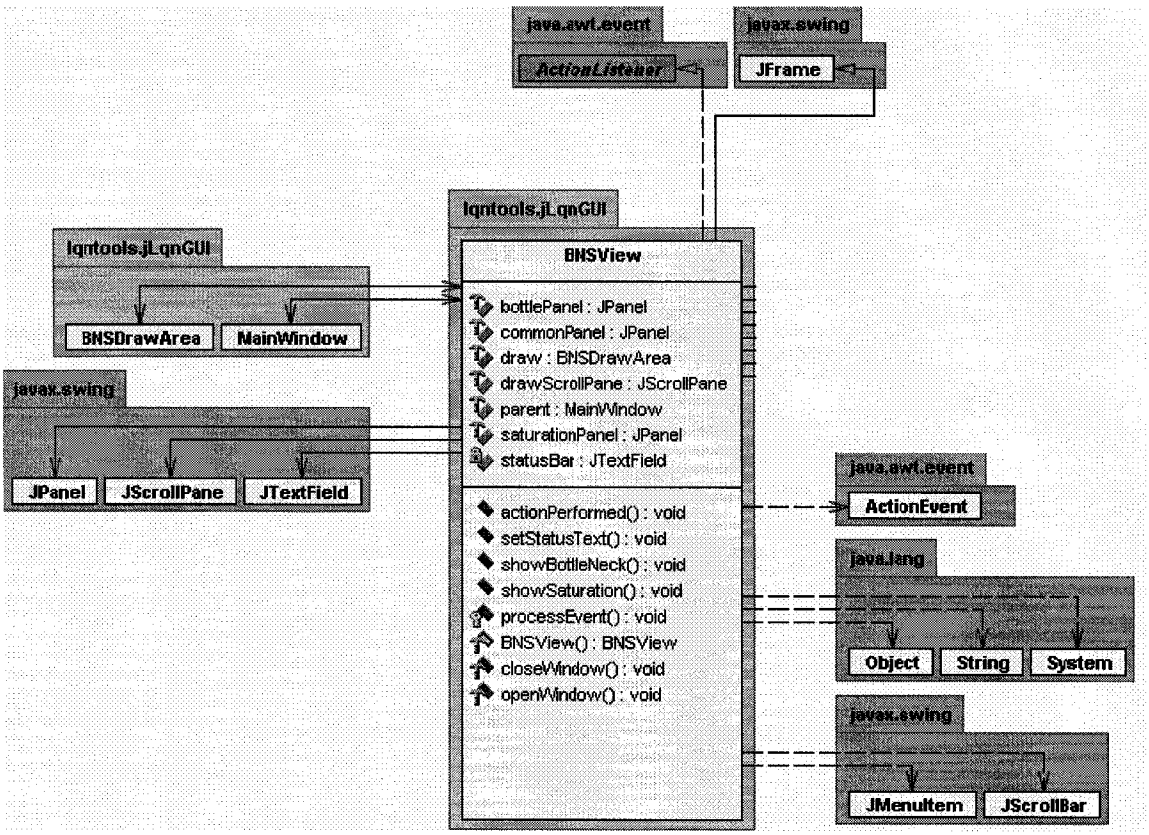


Figure 43 BNSView Class diagram

o BNSDrawArea Class

BNSDrawArea is the core of bottleneck visualization window. Its major task is to display graphical representation of Tasks and their calls with bottleneck strength. Graphically, this is known as the first level of bottleneck visualization mentioned above. If a Task is selected to display its saturation or service time breakdown, then it will display a graphical representation of Task, Entry, Phase, Phase execution fractions and calls. This is known as the second level visualization discussed above.

First, this class draws all the Tasks available in Model object in layered Manner. Then it draws the calls between the Tasks. It is important to realize the fact that conceptually Tasks don't make calls. Entries or Activities within a Task make the call. So, there

exists a need to aggregate all the calls originating from a Task and destined to a particular Task. Algorithm 7 describes this aggregation in detail. Figure 44 depicts the conceptual calls format that legacy Layered View draws and Figure 45 depicts the aggregated version of Figure 44 displayed in BNS view. For example, calls *C1* and *C2* originated from T1 and destined to T2 are aggregated into call *C1'*.

Second, it computes BNS for each task and fills red color in Graphical Task according to the classification discussed in Algorithm 5. BNS value and corresponding red fill ratio for Graphical Task rectangle is available in Table 6.

BNS Value	Red Color Fill	Image Representation
1.0 to 1.25	25 %	
Greater than 1.25 to 1.50	50 %	
Greater than 1.50 to 1.75	75 %	
Greater than 1.75	100%	

Table 6 BNS value and Red fill ratio

Finally, it listens for mouse events. Apart from standard mouse reaction, BNSDrawArea is designed to pop-up a menu bar when right clicked within the boundaries of a Graphical Task. By selecting the corresponding menu items, a modeler can (1) open a Task Dialog box (Figure 41), (2) open a Result Dialog box with performance matrix and (3) display the service time breakdown.

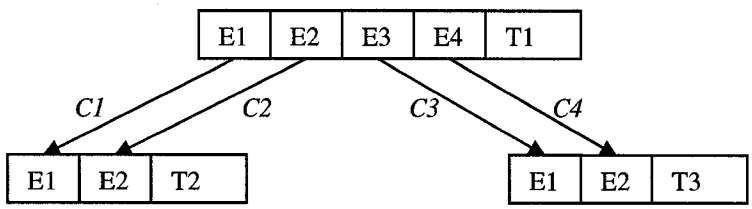


Figure 44 Layered View: Tasks with Entries and Calls

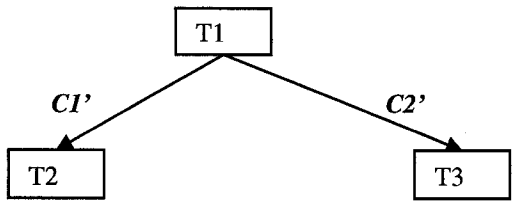


Figure 45 Bottleneck View: After aggregating calls destined to a server Tasks

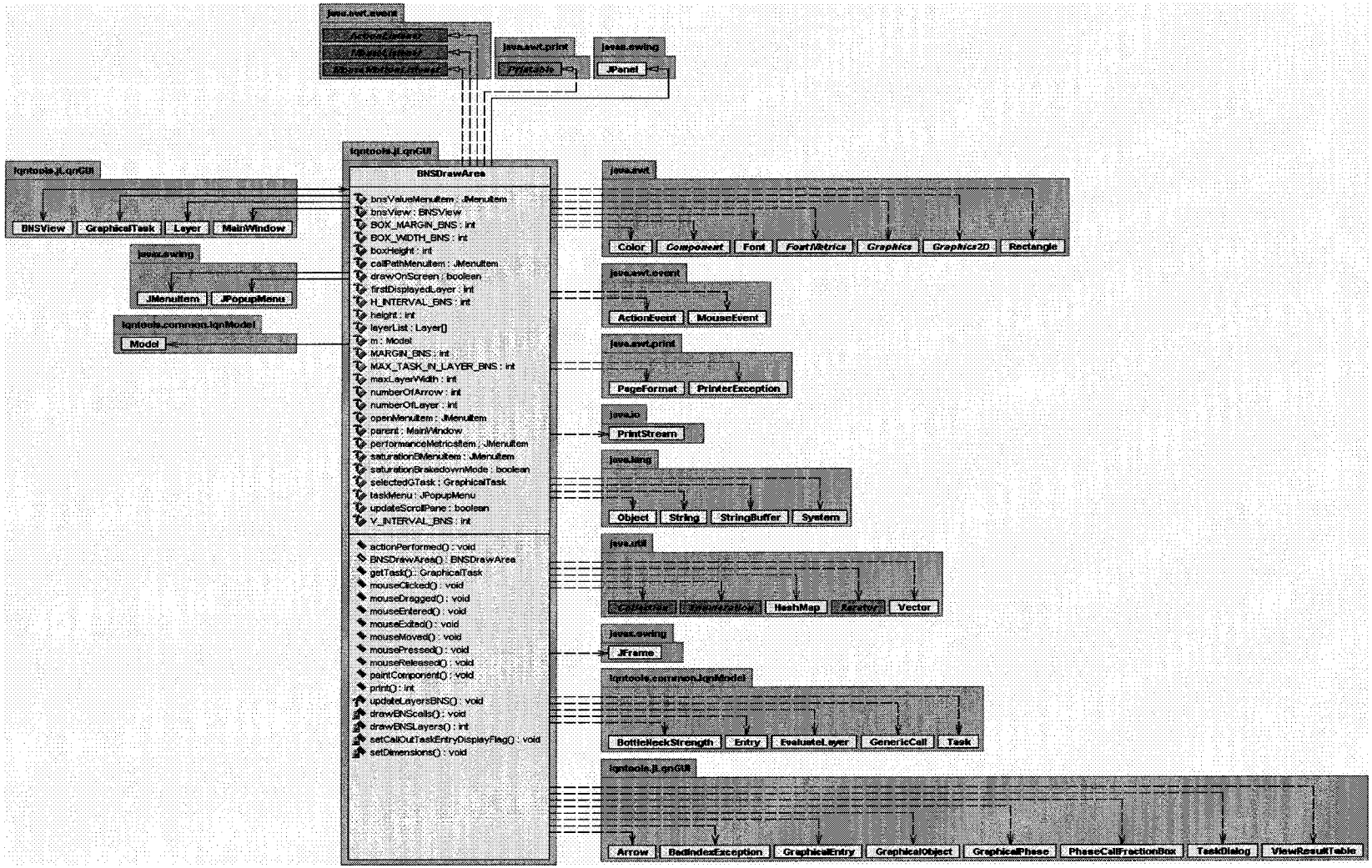


Figure 46 BNSDrawArea class diagram which support bottleneck strength visualization

- **GraphicalPhase Class**

Since the legacy Layered View does not show the phases, Editor's GUI package did not have a graphical representation of a Phase. This research introduced this class for the purpose of second level bottleneck display. Like other graphical classes, it is designed to be subclass of GraphicalObject and has a CJC Phase reference. For easy access, it has reference to its Entry and Task classes.

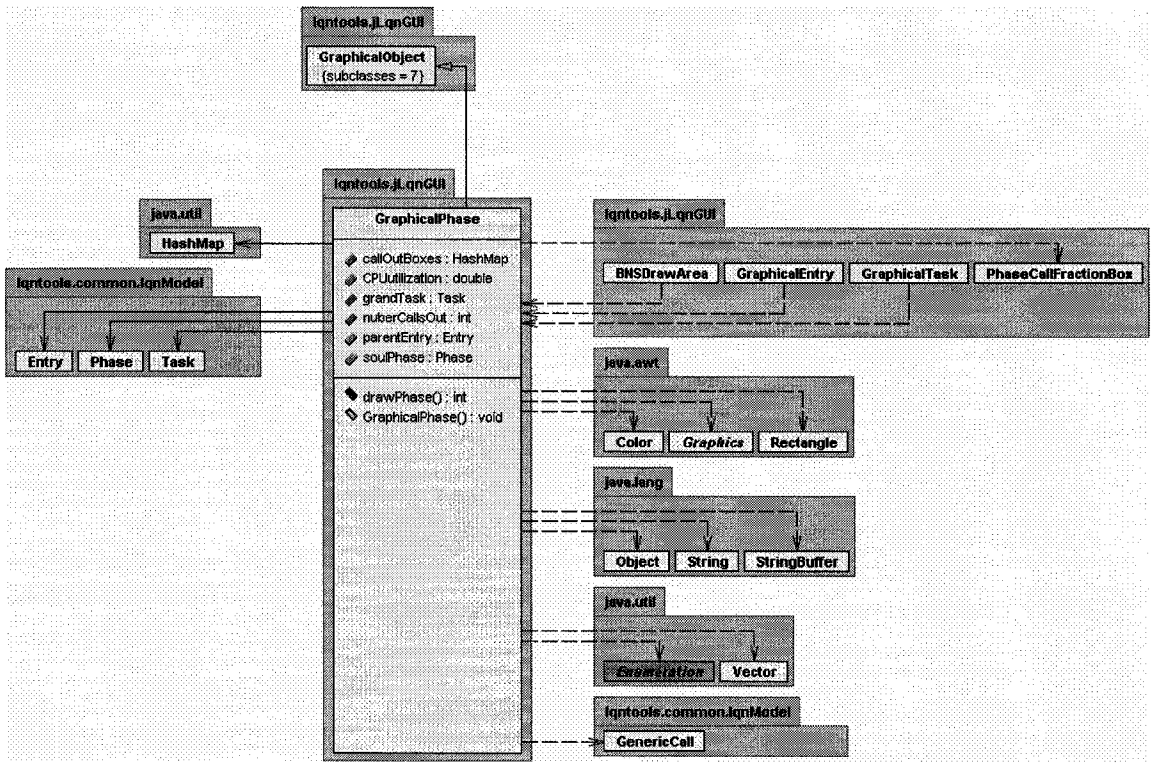


Figure 47 Class Diagram for GraphicalPhase

- PhaseCallFractionBox

This class is also introduced for the purpose of second level bottleneck display. This box represents the fraction of time consumed by a particular call of a phase. The width of the box varies in proportion to the time it consumes. Figure 48 depicts the class diagram for PhaseCallFractionBox. All the attributes and methods in it are self-explanatory.

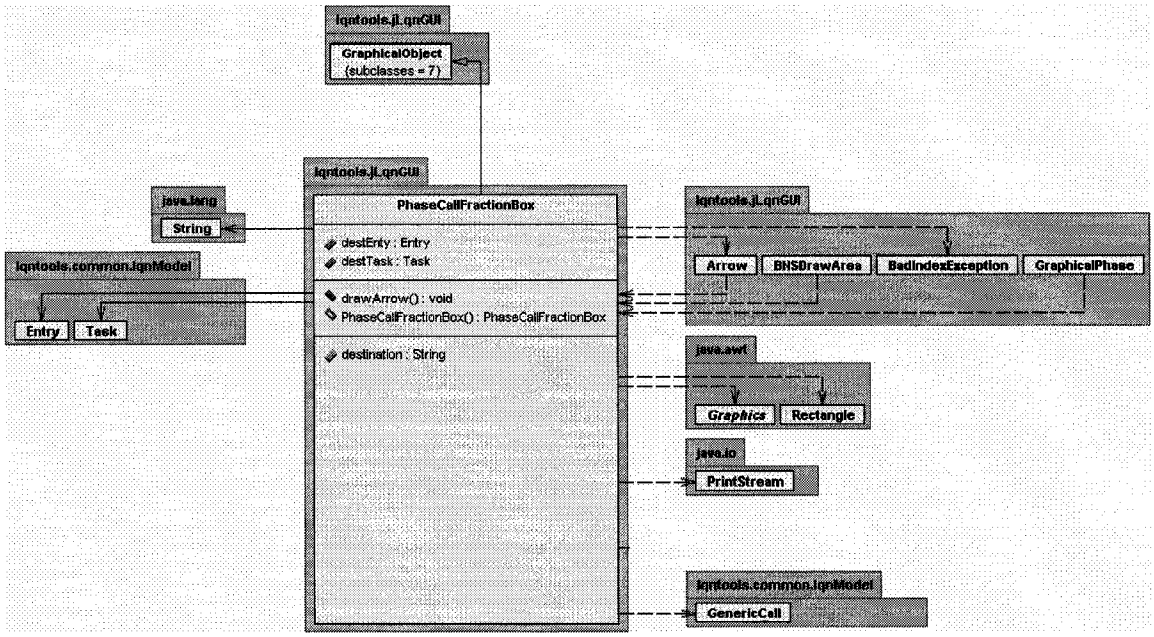


Figure 48 Class Diagram for PhaseCallFractionBox

- **Algorithm 6: Saturation Breakdown**

1. Using coordinates from mouse event, identify the target Task CJC object, T_{SB}, for which saturation breakdown is needed.
2. From T_{SB}, retrieve its Entries
    a. For each entry of T_{SB}, $E_{T_{SB}}$ retrieve its Phases objects:
        i. For each phase of $E_{T_{SB}}$:
            1. Calculate CPU utilization fraction that phase.
            2. Calculate the Blocking fraction due to calls made in this phase.
            3. Based on the Blocking Fraction (BF) value, create proportional graphical boxes.
        ii. Calculate $W_{E_{TSB}}$, new width of the Entry that includes boxes representing its phases.
    b. Calculate the idle fraction of T_{SB}.
    c. Calculate $W_{T_{SB}}$, new width of T_{SB}.
3. For all the Tasks that are right to T_{SB}, shift them to right by $W_{E_{TSB}}$.
4. List all the Tasks that are being called from T_{SB} and display their Entries.

5. If the width of all the tasks in the Layer exceeds the Maximum width of Layer, breakdown the Layer into to many Layers.
6. Refresh the Bottleneck View window.

- **Algorithm 7: Calls aggregation**

1. From CJC Model object, retrieve a list of available Task objects, L_T
2. For each Task, $T_{current}$,
    a. Create an empty list to store known Server Task Objects, $L_{T_{Server}}$
    b. Retrieve it's list of Entries (L_E)and Activities(L_A)
    c. For each Entry($E_{current}$) in L_E, retrieve the list of Phase objects, L_P
        i. For each Phase in L_P get a list of calls, L_C
            1. For each calls in L_C,
                a. get the destination Entry object, E_{Dest}
                b. Using E_{Dest}, find the sever Task, T_{Dest}
                c. If T_{Dest} is not in known servers list $L_{T_{Server}}$, add it to the list. Else, ignore it.
    d. For each Activity ($A_{current}$), in L_A, get a list of calls, L_C:
            1. For each calls in L_C,
                a. get the destination Entry object, E_{Dest}
                b. Using E_{Dest}, find the sever Task, T_{Dest}
                c. If T_{Dest} is not in known servers list $L_{T_{Server}}$, add it to the list. Else, ignore it.

    //Note: By now $L_{T_{Server}}$ should have some elements if $T_{current}$ is not a pure serve

    e. For each server task, T_{Dest} in $L_{T_{Server}}$:
        i. Find the respective GraphicalTask object, GT_{Dest}
        ii. Create a Graphical Call (Arrow) object making the source as as $GT_{current}$ and destination as GT_{Dest}

iii. Draw the Graphical Call in BNSDrawArea.

Chapter 6 Case Study: A Distributed Telecom Switch

Telecommunication switches are excellent example of performance sensitive systems where SPE has been successfully applied. The performance of these switches is highly regulated by national and international standard bodies such as Canadian Radio and Telecommunications Commissions (CRTC) and Telecordia [43][44][45]. For example, according to CRTC standard the "post dialing delay" should not exceed 0.4 seconds for 99.9% of attempted calls [15]. Thus, meeting these strict performance standards is vital for the success of any Telecom product.

This research reuses the LQN model developed by Tregunno in [15] for Voice over Packet (VoP) switching network. Automatic Bottleneck identification and Saturation breakdown concepts discussed in Chapter 4 are applied to this model through the use of Editor. The goal of this case study is to arrive at the same conclusion as Tregunno faster by using Editor's automated capability and visualization features. It is important to note that although Tregunno used the LQN solver to solve the model, all the bottleneck related calculations are done manually. In following subsections, brief background information needed to understand the model, details of the model and the application of this research are presented.

6.1 Background

The Model selected for this case study represents a Voice over Packet (VoP) switch. This switch evolved from classical voice switches that are structured into a two level hierarchy. Figure 49, depicts an abstract view of the classical voice switch network. Class 5 switches contain subscriber line and network trunk interfaces where as Class 4 only have trunk interfaces. This network also has a Database connected to it through the "Signaling Channel" commonly known as SS7. This Database is responsible for providing modern features such as "800" services and call display.

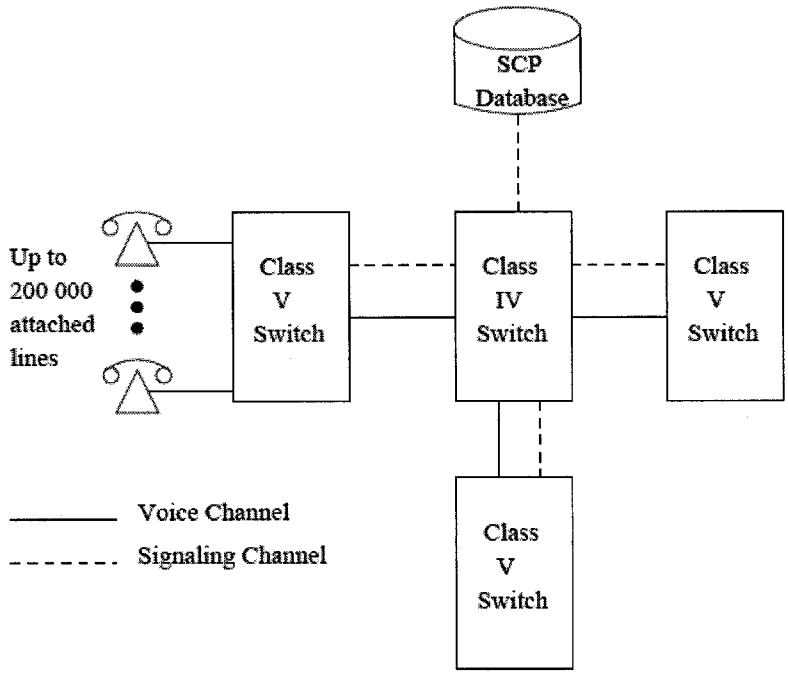


Figure 49 Current Telephone network view [4 pg.97]

The VoP switch evolved from the classical voice switch by replacing the class 4 switches with "VoP switches, VoP control points and packet based transport networks" [15]. Figure 50 depicts the abstract view of the evolved switch. As depicted in Figure 50, class 4 switch is replaced by four major components of VoP switches: (1) Call Connection Agent, (2) Originating VoP Trunking Gateway, (3) Terminating VoP Trunking Gateway and (4) Core Packet Switch. A detail list of distributed servers in the VoP switch can be found in [4].

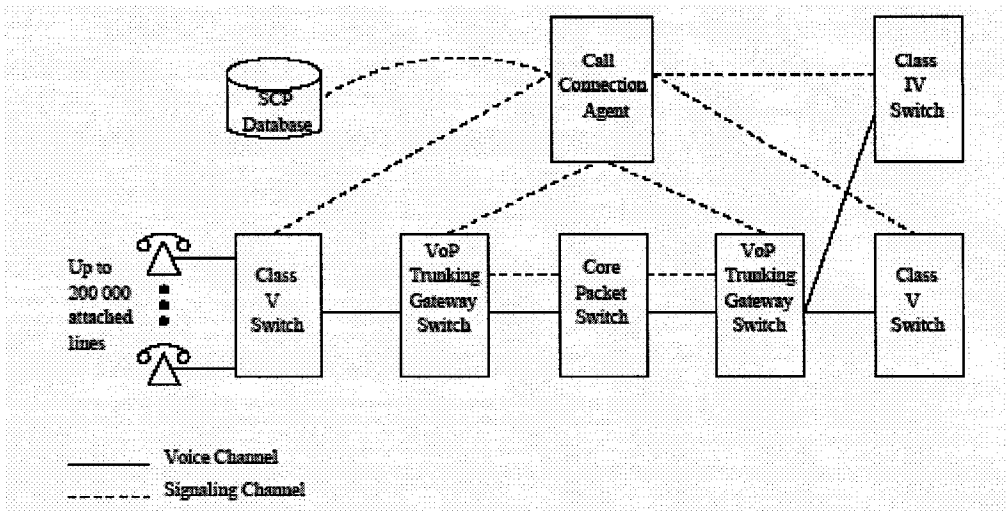


Figure 50 Next generation VoP networks [4 pg.99]

6.2 LQN Model for VoP Switch

Tregunno in [15] built the LQN model reflecting the four components of a certain VoP switch. Figure 51 depicts the task level LQN model. There are four dashed-line rectangle boxes representing (1) Call connection Agent, (2) Termination Gateway, (3) Originating gateway and (4) Core Switch.

As seen in LQN background section (chapter 2), building any LQN model requires having details of interaction between tasks, processor speeds, and execution times. Tregunno used design artifacts such as sequence diagrams and Kahkipuro's [15] ad-hoc techniques to construct this model. Execution times are estimated based on experience.

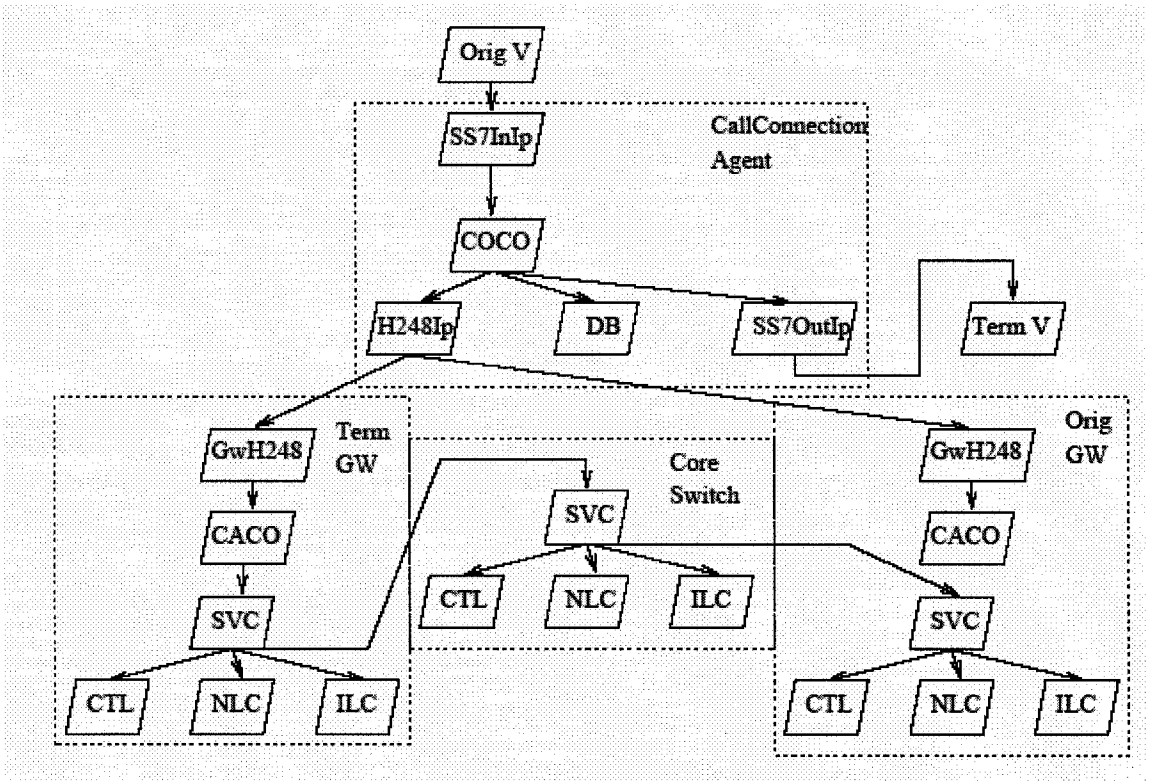


Figure 51 Task level LQN model for VoP switch [4 pg.102]

6.2.1 Internal Details of the Model

In this model, Class 5 switches are represented by the 'Orig V' task. This task is the reference task of the model where call throughput and setup delay are measured. As seen in Figure 51, there is another task, Term V, representing a terminating Class 5 switch. It is modeled as a pure delay server with the number of threads equal to the number of concurrent call attempts. These two tasks represent the edge Class 5 switches of VoP seen in Figure 50. All other tasks belong to the four major components of the VoP switch and their internal details are documented in [4].

6.3 Analyzing VoP Switch model in Enhanced Editor

The contribution of this thesis allows a modeler to analyze XML based LQN models in the editor. The enhanced editor has the capability of displaying (1) Layered view, (2) Bottleneck Strength View, (3) Saturation Breakdown View and (4) Result Values of LQML models. Having these sophisticated functionalities integrated into one tool allows modelers to create a model, understand it and diagnose bottlenecks faster

without doing any manual calculations. Using Tregunno's model, the following subsections demonstrate enhanced Editor's capabilities.

6.3.1 Layered View of VoP model

Although the Layered View was an existing functionality of Editor discussed in section [2.4.6], it was designed to work for legacy LQN models. The DOM parsers introduced in Chapter 3 enables LQML models to be converted into respective CJC objects. Editor uses these CJC objects to from the Layered View. Figure 52 depicts the Layered View of the VoP model generated by Editor. As seen in the left hand side of Figure 52, VoP model has eleven layers starting from Class5SwithchT_1 Task that represent the surrounding Class 5 switches and ending with gwIIcT_2 and gwNIc_2 tasks representing Line Interface and Network Interface Cards respectively of the VoP switch. In addition, it displays all the entries and calls within a Task. Being able to see and modify this information (by double clicking on any rectangle box) in a Layered manner enables modelers to maintain the integrity of the system.

Generating such a view for LQML version of VoP models demonstrate that the DOM parsers discussed in Chapter 3 work successfully.

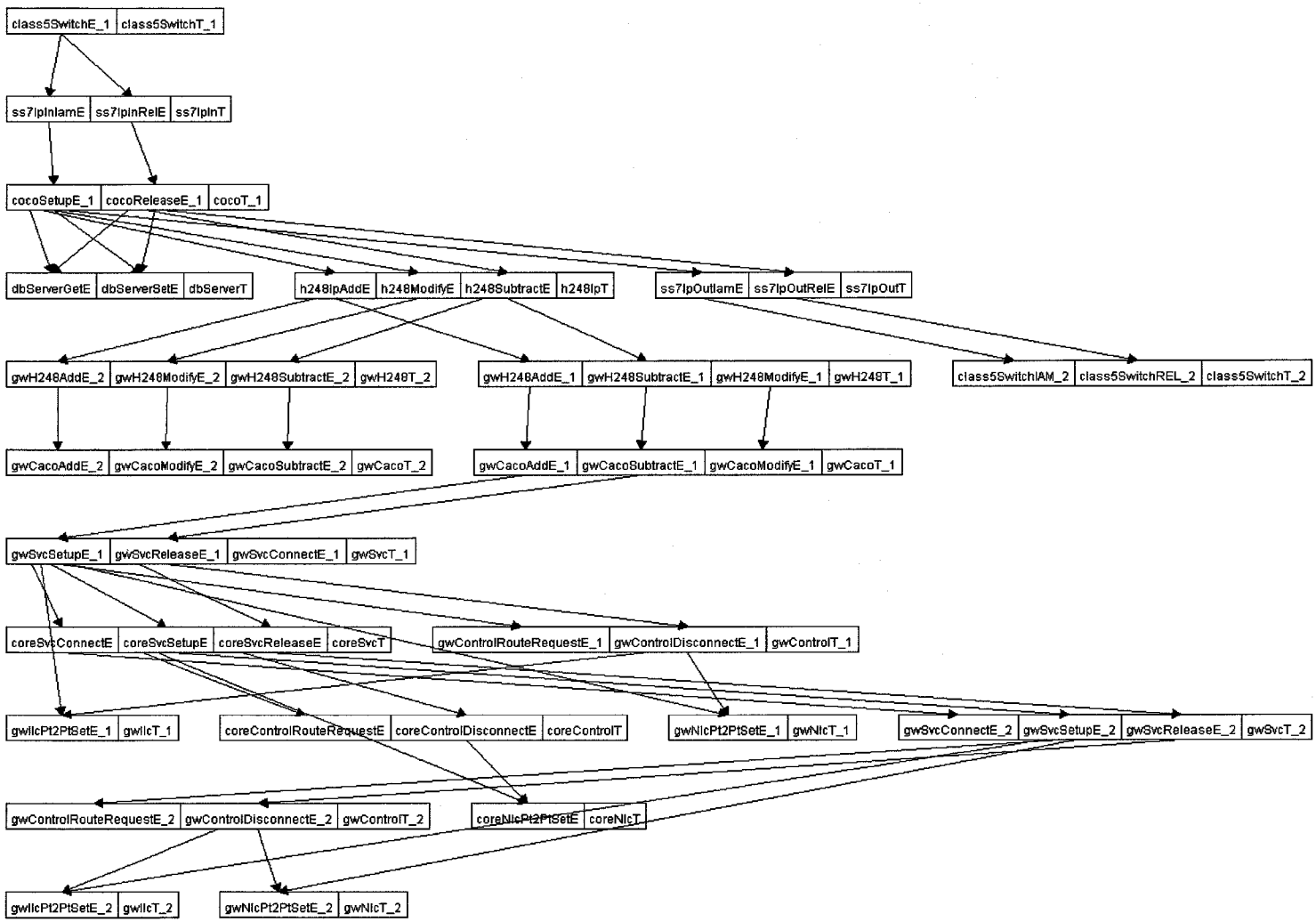


Figure 52 Layerd View of VoP model

6.3.2 Bottleneck Strength View of VoP System

This section identifies the potential bottleneck points in VoP system. Once the model is solved, results are embedded in the LQML file by the solver. Figure 53 depicts the bottleneck strength view for VoP switch with the utilization saturation threshold set to 75%. This view was produced by running bottleneck strength algorithms discussed in section [5.1] and applying graphical representation conventions discussed in section [5.2.3]. Hereafter, for simplicity, this view is referred to as the BNS view.

- **Benefit**

The major benefit of analyzing this model with the BNS viewer is to easily identify the tasks that are fully or partially saturated. It also allows quick traceability of software bottleneck "push-back" affects. Easy and rapid identification of bottleneck tasks and "push-back" paths are necessary for software bottleneck mitigation which is a recursive process. Being able to quickly scan for bottleneck tasks and their migration patterns after each run, during mitigation without manual calculation, reduces the overall SPE time.

- **Preconditions and Procedures**

The preconditions to view any models in BNS viewer are:

1. successfully solve the model with solvers or simulators and
2. models must have embedded results.

The Procedures are:

1. Open the model using the file menu of the editor,
2. set the saturation threshold utilization from the view menu,
3. select "Bottleneck Strength" to launch the viewer.
4. search for full reddish task boxes.

- **Interpretation of the View**

The BNS view shows only tasks and their layers hiding the details of entries. As discussed in section 5.2.3, redness width of the task boxes indicates the bottleneck strength. Thus, to identify the bottleneck task, search for the most reddish boxes.

The VoP model's bottleneck strength is depicted in Figure 53 with the threshold utilization set to 75%. From Figure 53, it can be seen that there are five fully reddish

rectangles indicating "Strong" bottleneck tasks and five partial reddish rectangles indicating "weak" bottleneck tasks. Note, one of the partially reddish tasks is the reference task (class5SwitchT_1) that should be ignored from the bottleneck strength analysis.

Note, the BNS value of a task can be quickly retrieved from the Bottleneck Strength Viewer by opening (right mouse click and selecting "open") its dialog box. Further, all the results values for a task can be displayed in the BNS viewer (see Figure 55) by selecting "Performance Metrics". Table 7 and Table 8 list these tasks with their BNS values. From Table 7, it is clear that ss7IpInT task has highest BNS value (21.75591) followed by gwSvcT_2 (5.74537) and gwSvcT_1 (2.111892). Evidently, this system has more than one bottleneck point. Classification of the bottleneck (i.e. task supported or processor supported) can also be retrieved from task dialog box. All of the five strong bottlenecks of VoP system are task supported. Thus, mitigation will not involve processor replacement (hardware change).

In addition, the BNS viewer can be used to recognize bottleneck "push-back" paths. Figure 54 illustrates such bottleneck "push-back" path for VoP switching model. The push back effect goes from layer 9 task gwSvcT_2 to all the way up to layer 2 task ss7IpOutT.

- **Summary**

In summary, analyzing the VoP model in the Bottleneck Strength Viewer provides the following three information without any manual computations: (1) identifies the saturated tasks, (2) classify them as task or processor supported (3) allows visualizing the "push-back" path. However, this analysis does not explain the root causes for the identified bottlenecks. In Section 6.3.3, by going one step further, the root causes for the five strong bottlenecks using the saturation breakdown capabilities of the editor are investigated further.

Table 7 Strong bottleneck Tasks

Task Name	Layer Number	Bottleneck Strength
ss7IpInT	Layer 2	21.75591
gwSvcT_1	Layer 7	2.111892
coreSvcT	Layer 8	2.067627

gwSvcT_2	Layer 9	5.74537

Table 8 Weak Bottleneck Tasks

Task Name	Layer Number	Bottleneck Strength
cocoT_1	Layer 3	1.253789
ss7IpOutT	Layer 4	1.003889
gwH248T_1	Layer 5	1.012029
gwCacoT_1	Layer 6	1.049084

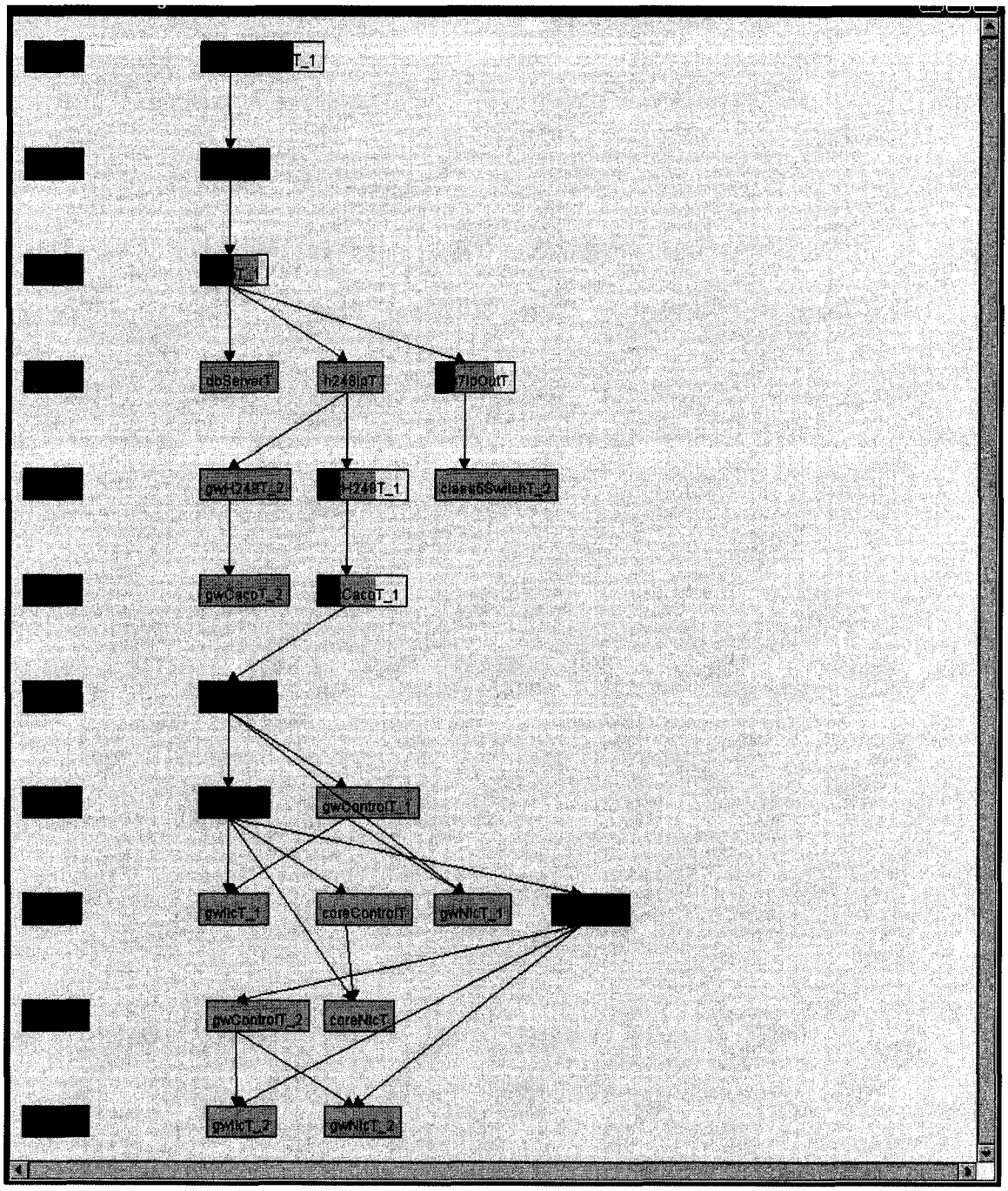


Figure 53 Bottleneck Strengths of VoP system

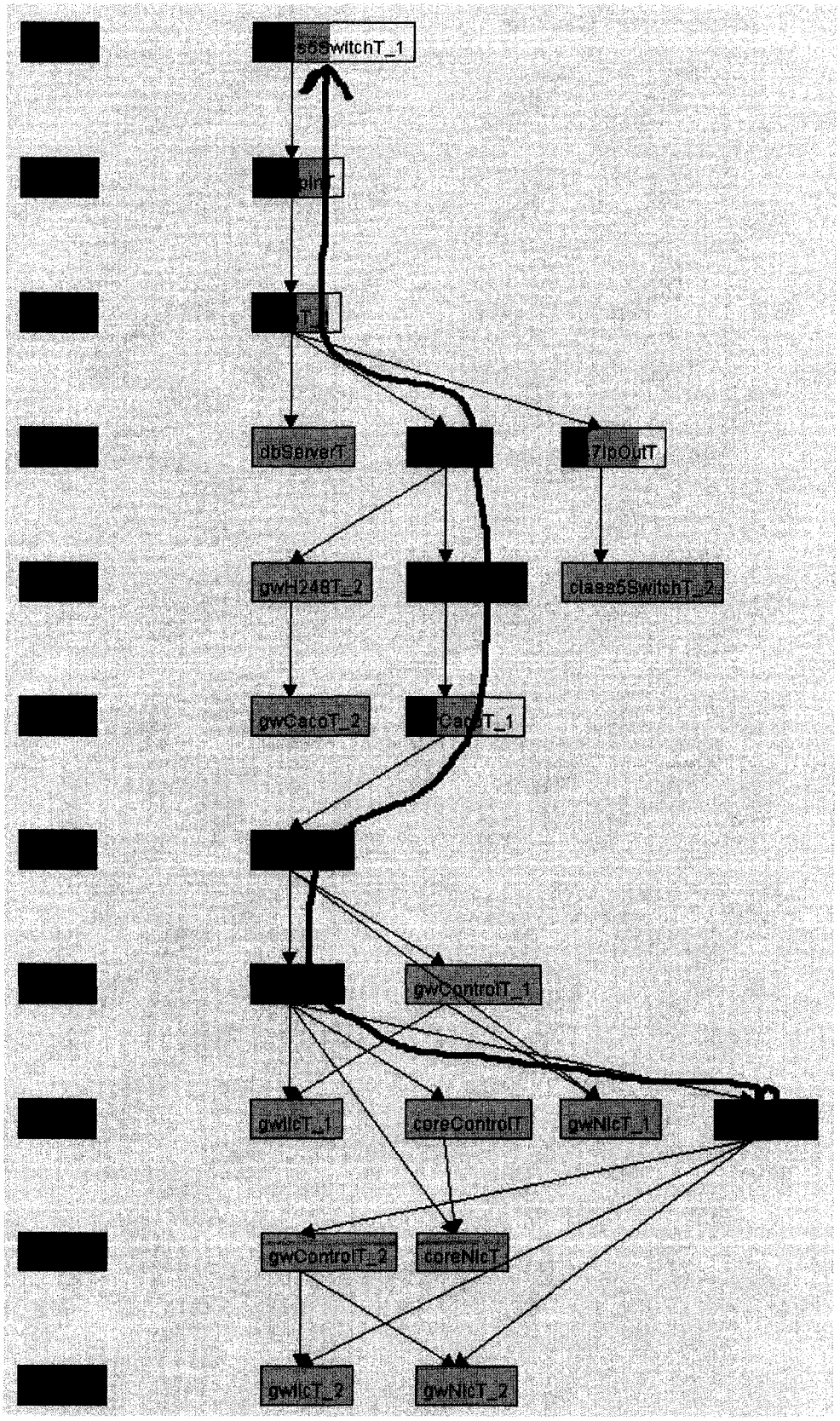


Figure 54 Bottlneck Call Path of VoP system

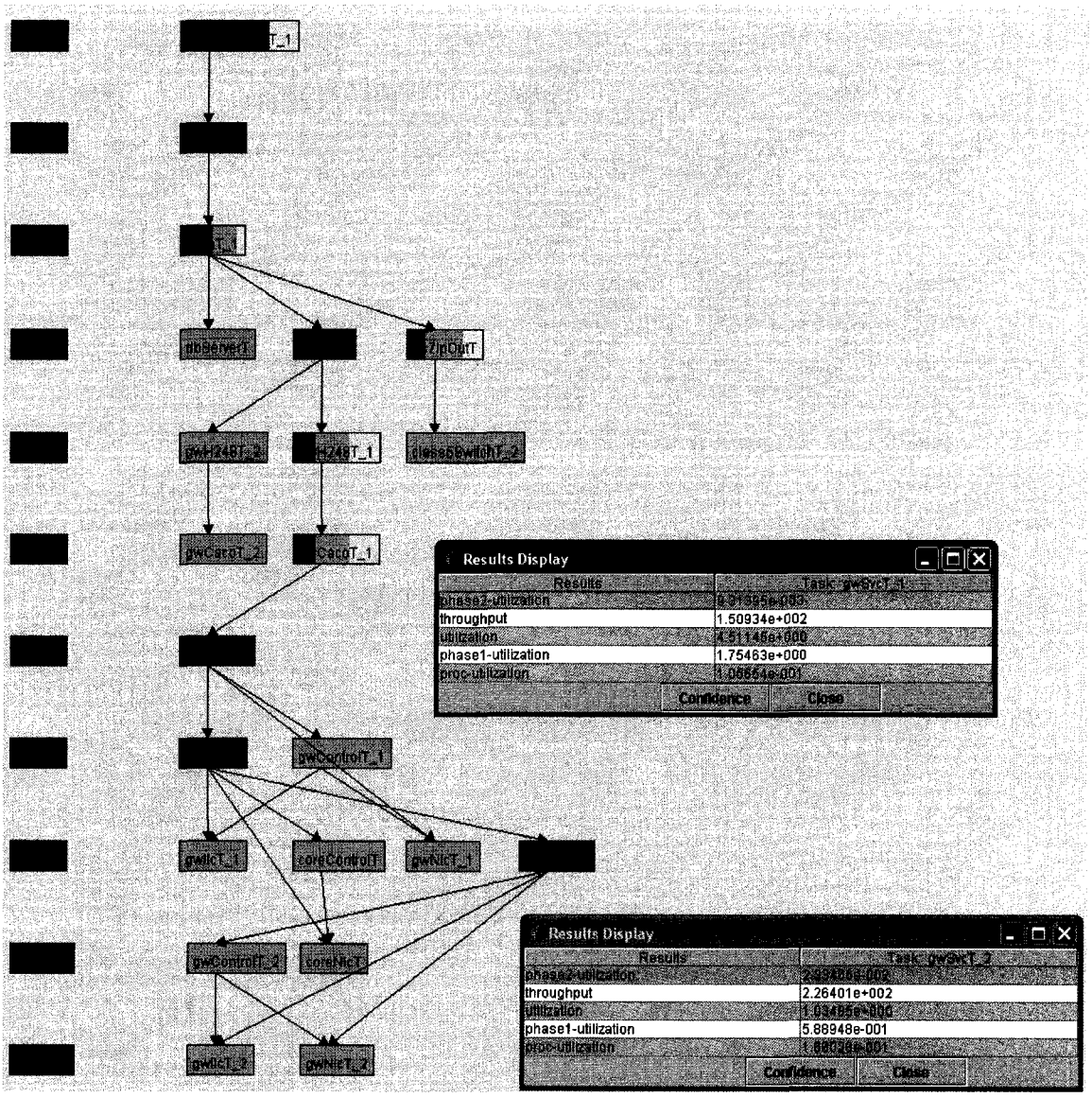


Figure 55 BNS Viewer showing the task results

6.3.3 Saturation Breakdown View

In the previous section, using the bottleneck strength viewer, a set of strong bottleneck tasks were identified. Out of the five tasks in this set, three have relatively high BNS value (BNS value grater than 15). Obviously, as the next step, one must gain insights into the root causes for such high saturation. As discussed in section 5.2, this can be achieved by investigating where and why a task spends most of its time. Here, we examine the root causes using the "saturation breakdown view".

- **Benefit**

The major benefit of analyzing the VoP model with the Saturation Breakdown viewer is to easily identify where and why a task spends most of its time. Since this functionality is integrated into the Bottleneck Strength Viewer (section 6.3.2), traceability of software bottleneck "push-back" is maintained. Again, being able to quickly breakdown the task service times after each run during mitigation without manual calculation reduces the overall SPE time.

- **Preconditions and Procedures**

The Preconditions to analyze the saturation breakdowns of any model in the BNS viewer are:

1. successfully solve the model with solvers or simulators
2. models have embedded results.
3. successfully opened the BNS view.

The Procedures are:

1. Open the model using the file menu of the editor.
2. Set the saturation threshold utilization from the view menu.
3. Select "Bottleneck Strength" to launch the BNS viewer.
4. Select a Task to see the saturation breakdown.
5. Right click and select "Saturation Breakdown"
6. Now, the selected task will be expanded showing entries, phases and calls of a task.

- **Interpreting saturation breakdown view**

Here, for demonstration, the highest BNS task of VoP model is expanded and interpreted. Since the interpretation is similar for the rest of the tasks, it is not necessary to discuss them here. Nevertheless, Figure 58 is included to illustrate the Editor's capability to expand multiple tasks for saturation breakdown analysis.

As seen earlier in Table 7, the gWSvcT_1 task has the highest BNS value. Figure 56 shows the expanded gWSvcT_1 task with their entries, phases and calls along with their associated fractions (graphical representation) discussed in section 5.2. Note, there is a white color divider box between each phase and entry. It is shown only for the purpose of visual differentiation (i.e. indicating end of a phase or an entry). Task gWSvcT_1 in Layer 7 has three entries: (1) gwSvcSetupE_1, (2) gwSvcReleaseE_1 and

(3) gwSvcConnectE_1. Note, entry gwSvcConnectE_1 should be ignored since it doesn't have any calls. It was designed in [15 p.103] for future use. So, only gwSvcSetupE_1 and gwSvcReleaseE_1 are contributing to high BNS value. By looking at the fraction boxes, it is evident that phase one of entry gwSvcReleaseE_1 contributes most to the BNS value. Red color boxes indicate that it has the same CPU utilization fraction ($Fcpu_p$) as Phase one of gwSvcReleaseE_1. However, according to the blue boxes, it has significantly higher fraction of accumulated call-blocked time (Fcb_p). Thus, it is clear gWSvcT_1 task spends most of its time blocked on five calls made from gwSvcSetupE_1entry.

Similarly, from here on, one can continue to expand the higher or lower layer tasks of gwSvcSetupE_1 for causality trace. Figure 57 depicts one of the lower layer saturation breakdowns (for coreSvcT) whereas Figure 58 depicts one of the upper layer breakdowns (for h248IpT). For example, consider the lower layers in Figure 57, gWSvcT_1 makes calls to already saturated task coreSvcT that in turn calls another saturated task (gWSvcT_2). Clearly, making synchronous calls to already saturated tasks increases the call-blocked time (α_c) at gWSvcT_1. Obviously, insights gained from such causality trace are sufficient to mitigate bottleneck at gwSvcSetupE_1.

- **Limitations**

Known limitations of Editor's saturation breakdown view are:

1. expanding tasks with many entries in the same layer leads to a very wide viewer,
2. origin and destinations of call arrows are not aligned to center of the box and
3. Expanding many tasks in all the layers for causality trace can result in crowded and messy viewer.

- **Summary**

In summary, the saturation breakdown view helps to visualize the causality trace of a model. Visualizing and understanding such causality trace is important to propose any mitigation strategies. In addition, this view eliminates the tedious and erroneous manual computations of equations discussed in section 5.2.

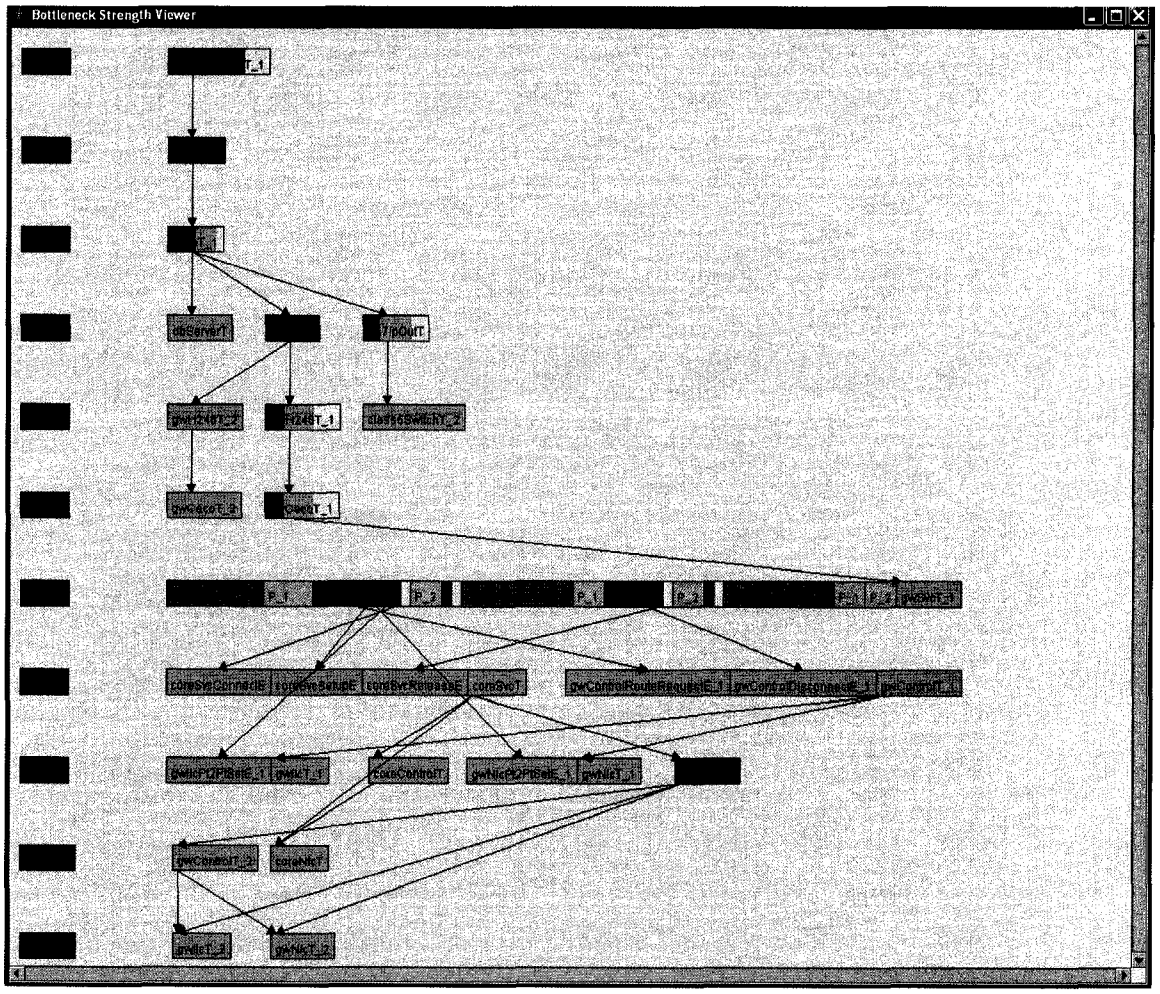


Figure 56 Saturation Breakdown View for gwSvCT_1

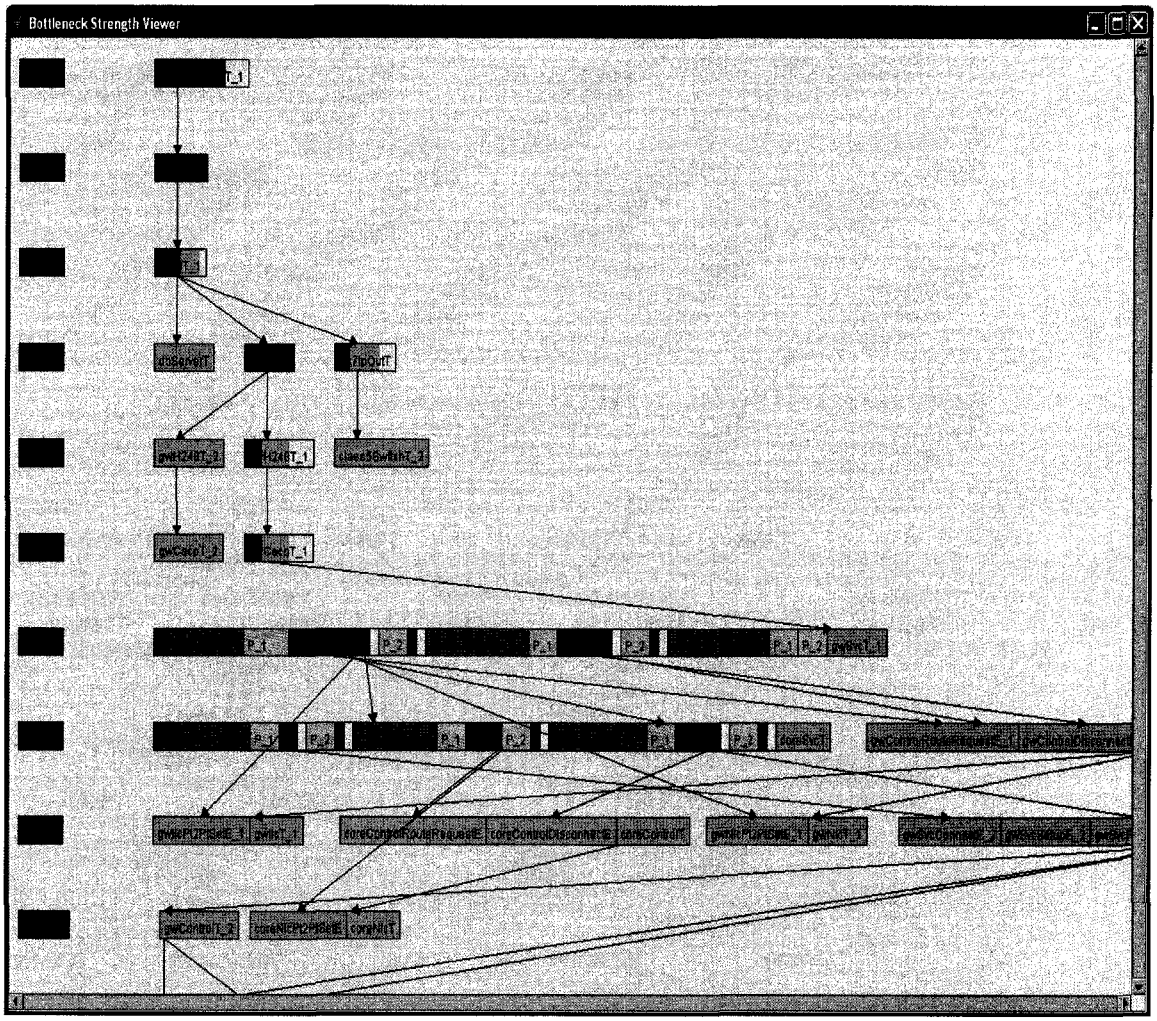


Figure 57 Downward causality trace of VoP model

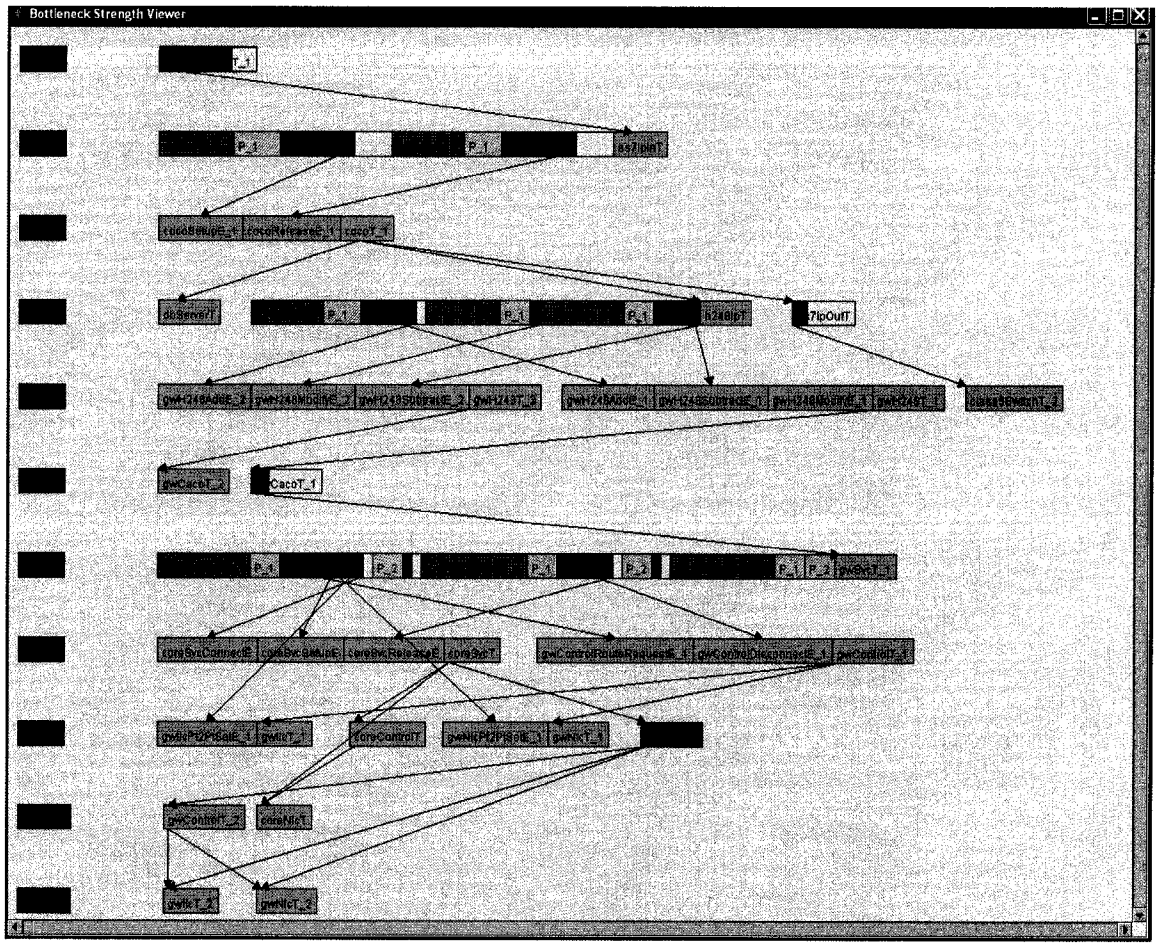


Figure 58 Upward causality trace of VoP model

6.3.4 Mitigation Strategy Demonstration

As discussed in the previous sections, the BNS viewer and Saturation break down views can help mitigate a bottleneck task. In this section, this capability is demonstrated by applying the task multiplication strategy mentioned in section 4.4.

From Table 7, ss7IpInT is the strongest bottleneck in the system with BNS value of 21.75591. Obviously, mitigation strategies should be applied to this task first.

Step 1: Multiplicity at the Bottleneck Task

The task multiplicity of the ss7IpInT task has been increased from 25 to 300. Table 9 summarizes the BNS value, task utilization and effects on cocoT_1 for each multiplication step. Figure 59 depicts the linear relationship between the number of thread and BNS value for ss7IpInT task. It suggests that the task multiplication will not

help in reducing the BNS value. In addition, saturation breakdown from Figure 58 indicates ss7IpInT has higher call-blocked time fraction. Thus, Step 2 attempts to reduce the call-blocking fraction.

Table 9 ss7IpInT Task Multiplication

	ss7IpInT			cocoT_1	
Num Task	Utilization	BNS	Num Task	Utilization	BNS
25	2.50E+01	2.499995	10	1.00E+01	1.285958
50	5.00E+01	4.99995	10	1.00E+01	1.285542
75	7.50E+01	7.499917	10	1.00E+01	1.285072
100	1.00E+02	9.99986	10	1.00E+01	1.285328
125	1.25E+02	12.49972	10	1.00E+01	1.286146
150	1.50E+02	14.99971	10	1.00E+01	1.284842
160	1.60E+02	15.99952	10	1.00E+01	1.285939
170	1.70E+02	16.99942	10	1.00E+01	1.28523
180	1.80E+02	17.99942	10	1.00E+01	1.284819
190	1.90E+02	18.99922	10	1.00E+01	1.285976
200	2.00E+02	19.99936	10	1.00E+01	1.284912
250	2.50E+02	24.99867	10	1.00E+01	1.285378
275	2.75E+02	27.49835	10	1.00E+01	1.285629
300	3.00E+02	29.99723	10	9.99999	1.285442

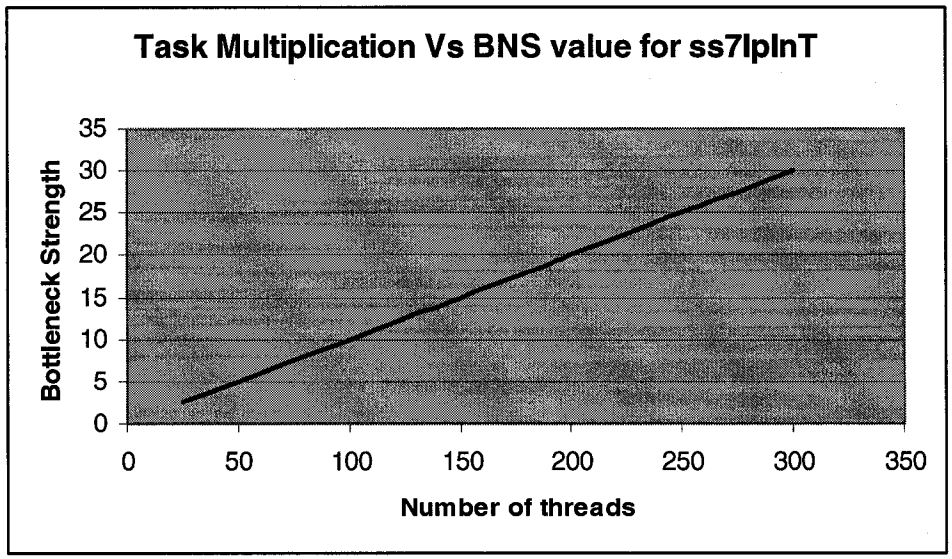


Figure 59 BNS value Vs Multiplication for ss7IpInT

Step 2: Reduce Call-blocking at the Bottleneck

A second approach for mitigation is to reduce the call-blocked time at the bottleneck. This can be accomplished by introducing task multiplication to the cocoT_1 task (server for ss7IpInT). Table 10 lists the task multiplication number introduced at cocoT_1 and its impact on utilization and BNS for ss7IpInT and h248IpT. Figure 60 illustrate the BNS value changes at cocoT_1, ss7IpInT and h248IpT as a result of introducing task multiplication at cocoT_1. Figure 61 to Figure 68 depict BNS view and Saturation breakdown for a sequence of higher task multiplication levels.

Figure 61 indicates that increasing the multiplication at cocoT_1 from 10 to 35 reduces its bottleneck. The ss7IpInT box in BNS view is only half reddish indicating a "weak" software bottleneck. Saturation breakdown (right hand potion of Figure 61) confirms this by the significant reduction in call-blocked time fraction boxes (in blue). Also, note that the ss7IpInT task starts to show a green box indicating available idle time (compare to Figure 58). Thus, visualization of BNS and saturation breakdown suggests a significant reduction of bottleneck at ss7IpInT when the task multiplication at cocoT_1 is 35. Numerical values in Table 10 and graph in Figure 60 validates this conclusion.

Step 3: Reduce Call-Blocking Further

Next, assume there is a strict requirement for a bottleneck free ss7IpInT task, reducing the bottleneck strength below 1. How and at what cost can this requirement be fulfilled? As seen earlier, increasing the task multiplication at cocoT_1 from 10 to 35 brought ss7IpInT from a strong to a weak bottleneck. To go further, task multiplication at cocoT_1 was increased in steps of 25 and the changes to ss7IpInT were observed. When the multiplication at cocoT_1 was 150, a bottleneck free ss7IpInT task was achieved (see Figure 66). However, as seen in Figure 60 and Figure 66, it has made h248IpT and cocoT_1 very "strong" bottlenecks with BNS values of 7.93 and 4.379 respectively. This result confirms the fundamentals of software bottlenecks:

1. Practically, software bottlenecks cannot be eliminated from a system and
2. Software bottlenecks can migrate to different locations.

Once more, figures from Figure 61 to Figure 68 demonstrate that the visualization techniques established in this research work helped to quickly identify the bottleneck

migration. In general, knowledge of such bottleneck migration pattern in a system can help to push the bottlenecks to less critical area.

Further Steps

Figure 67 and Figure 68 suggest that any task multiplication at cocoT_1 beyond 150 will not help to reduce the BNS value at ss7IpInT. In fact, the BNS value for ss7IpInT started to increase slightly. It is clear (from Table 1 and Figure 60) the bottleneck was migrating to h248IpT as cocoT_1 multiplication is increased. Thus, if any further performance improvement is desired at ss7IpInT, mitigation strategies used in Step 1, 2 and 3 should be applied to h248IpT.

Table 10 Task Multiplication at cocoT_1

	cocoT_1			ss7IpInT			h248IpT	
Num Task	Utilization	BNS	Num Task	Utilization	BNS	Num Task	Utilization	BNS
10	1.341306	1.52E+00	170	2.55E+02	16.77907	20	3.81E+01	0.915423
35	5.49E+01	1.441603	170	8.07E+01	1.471418	20	3.81E+01	1.815423
50	7.76E+01	2.310878	170	2.82E+02	3.632627	20	3.36E+01	5.005726
75	1.17E+02	2.377542	170	2.52E+02	2.155345	20	3.71E+01	5.402733
100	1.56E+02	3.82702	170	2.69E+02	1.717384	20	4.09E+01	6.161557
125	1.95E+02	4.661369	170	2.75E+02	1.41282	20	4.18E+01	6.308363
150	2.32E+02	4.76284	170	2.30E+02	0.996841	20	4.26E+01	6.36057
175	2.69E+02	6.413128	170	2.93E+02	1.088618	20	4.20E+01	6.333333
200	2.69E+02	6.413118	170	2.93E+02	1.088614	20	4.20E+01	6.333323

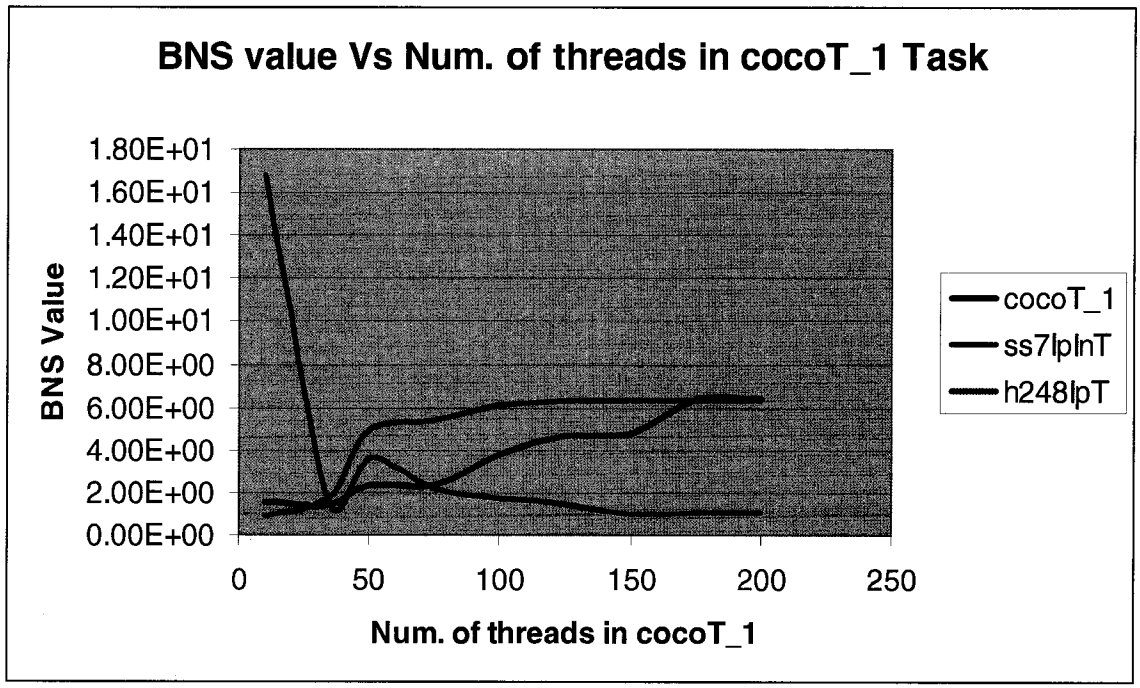


Figure 60 Num. of task multiplication at cocoT_1 Vs BNS values

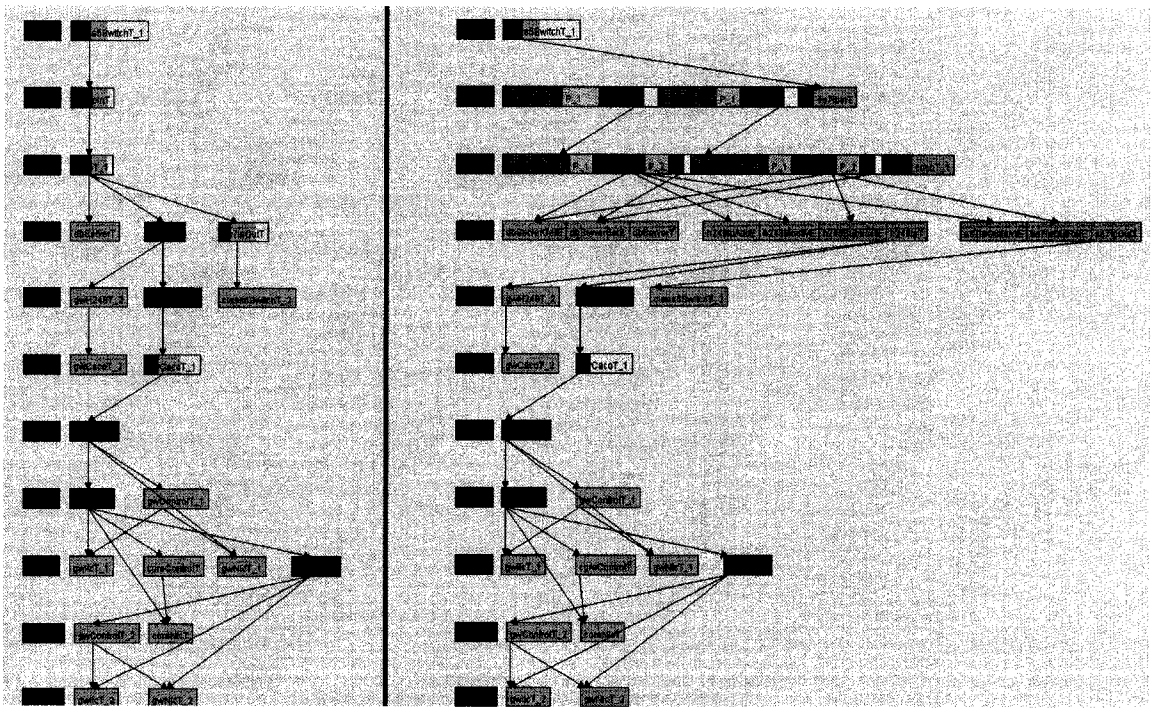


Figure 61 BNS View and Saturation breakdown for m=35 at cocoT_1 task.

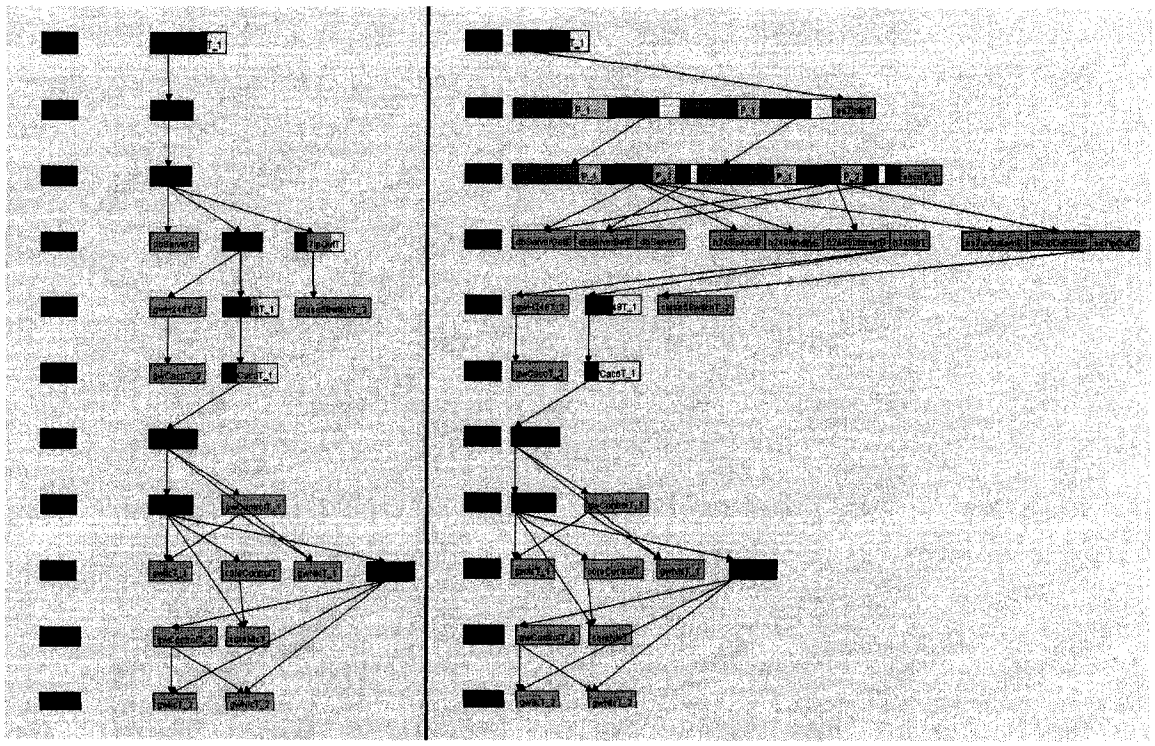


Figure 62 BNS View and Saturation breakdown for m=50 at cocoT_1 task.

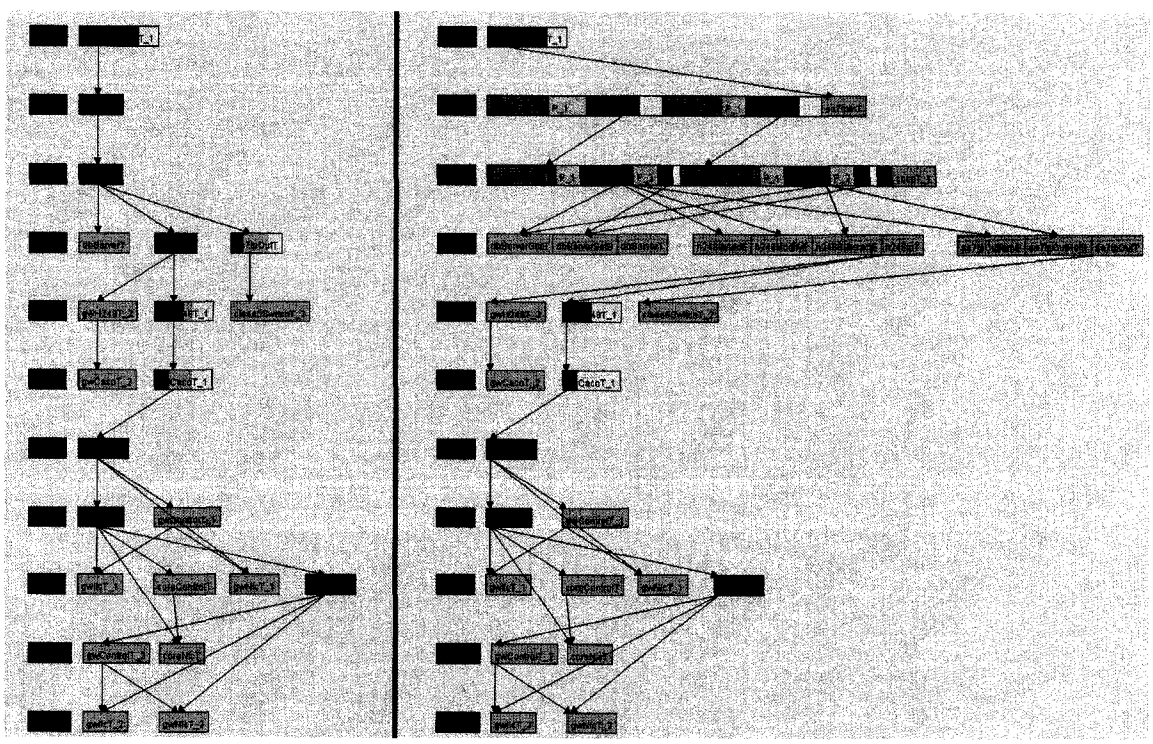


Figure 63 BNS View and Saturation breakdown for m=75 at cocoT_1 task.

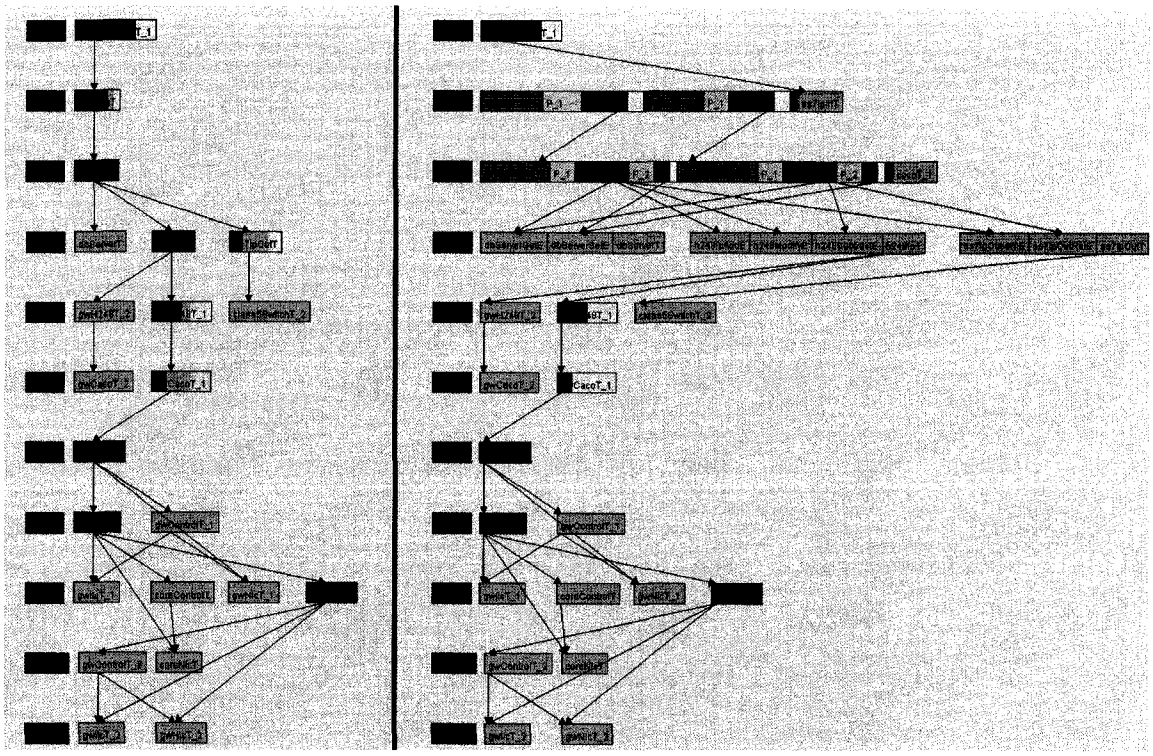


Figure 64 BNS View and Saturation breakdown for m=100 at cocoT_1 task.

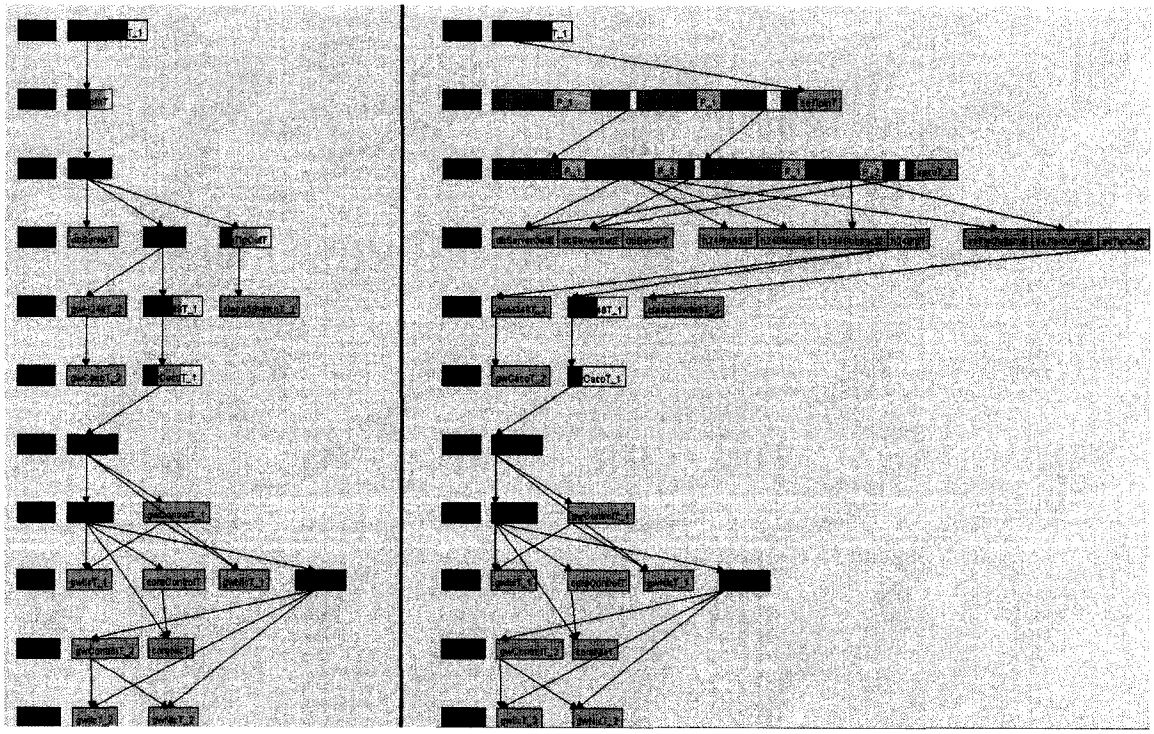


Figure 65 BNS View and Saturation breakdown for m=125 at cocoT_1 task.

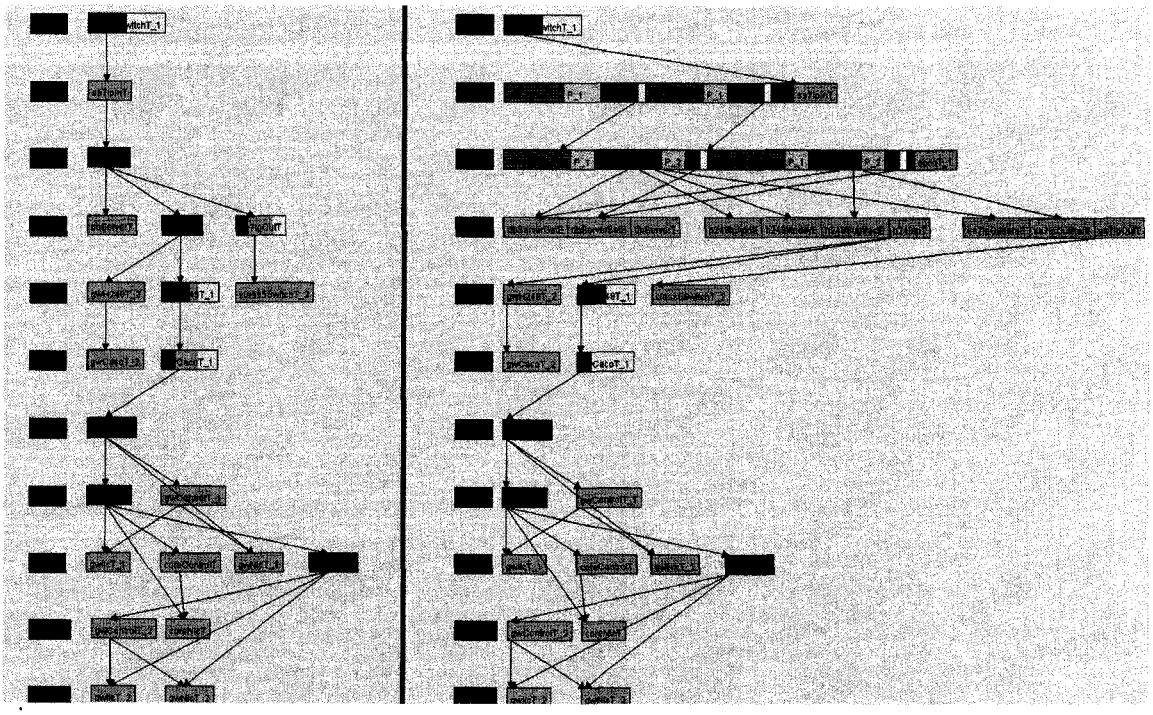


Figure 66 BNS View and Saturation breakdown for m=150 at cocoT_1 task.

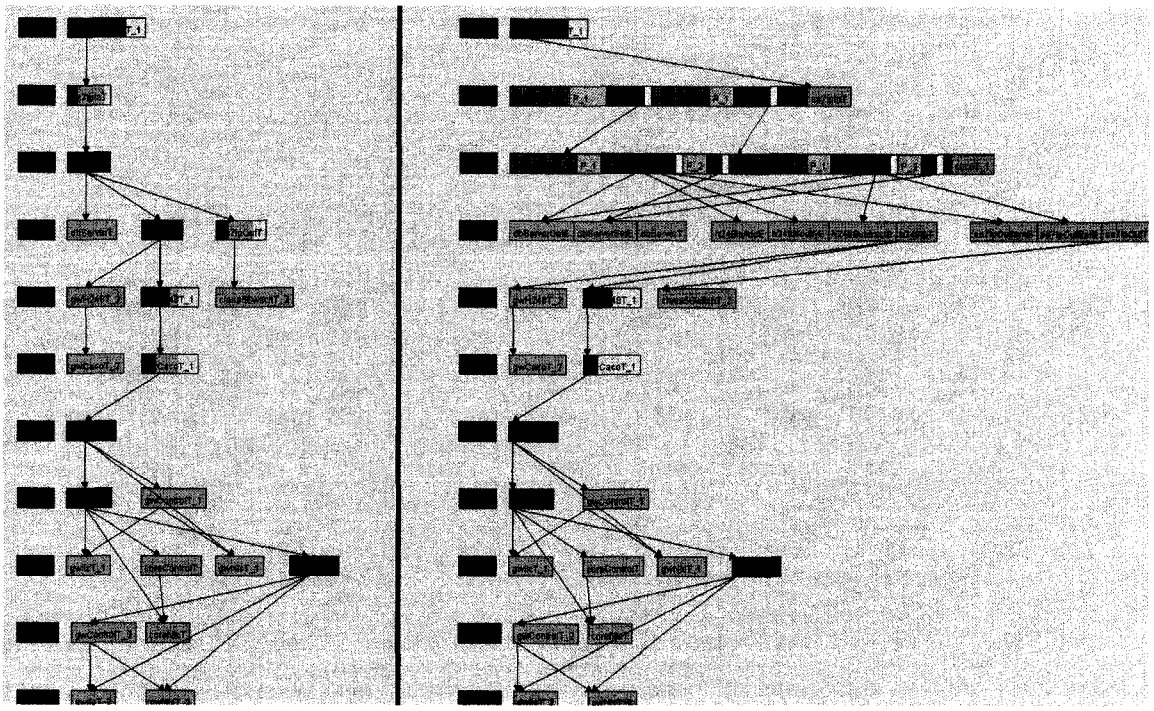


Figure 67 BNS View and Saturation breakdown for m=175 at cocoT_1 task.

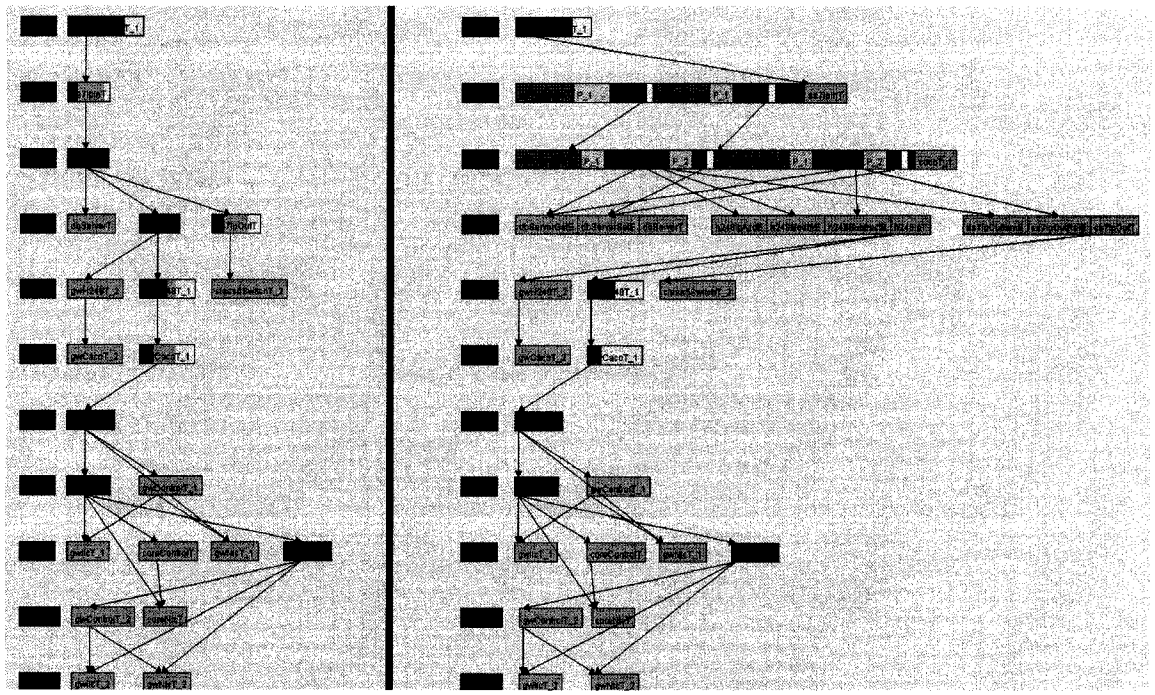


Figure 68 BNS View and Saturation breakdown for m=200 at cocoT_1 task.

Chapter 7 Conclusion

7.1 Contribution

The specific contributions of this thesis are:

1. **Enhanced editor with a DOM-XML parser for LQML and CBML.**

   This parser is the foundation for this research. Designing and developing it involved many complicated algorithms that translate XML representation to LQN-Core Java Classes (CJC) and vise versa. Design decisions and the algorithms are described in Chapter 3. Having this parser and embedded results as part of CJC framework enables effective development of valuable LQML or CBML related tools.

2. **Enhanced Editor with Embedded Result capabilities**

   Unlike the legacy Editor which holds the results in a text file, the latest editor stores the result values for each Processor, Task, Entry etc. as a DOM node. It can display result values for a specific Task or Entry. Having the results as DOM nodes under each LQN elements allows efficient filtering and abstractions using standard APIs. Capabilities of efficient filtration and abstraction of results from the raw data enables elegant presentation. Thus, it reduces the tediousness of SPE.

3. **Bottleneck identification, Saturation Breakdown and Visualization Capabilities**

   The basic algorithm for Software Bottleneck identification established in [15] by Peter Tregunno is encapsulated into a new automation algorithm. This new algorithm is implemented using the CJC framework. Furthermore, a visual display is provided to show the system level bottleneck strength of each server. An approach to saturation breakdown is established in section 4.5 followed by extending editor for its visualization. Such a breakdown helps to understand the reasons for a bottleneck server/task.

4. **Component Based Model Creation, Manipulation and Visualization**

   Editor is enhanced to support component driven modeling developed by Wu in [16]. In the conclusion of [16], Wu claims that introduction of XML into

LQN modeling is a limitation due to the extensive amount of tags to represent a model and the need for third party tools. This limitation is eliminated by this research as a result of LQML/CBML aware Editor.

5. **A case study demonstrating the capability of enhanced Editor**

   A Voice over Packet (VoP) switch model is used in Chapter 6 to demonstrate the capability of the enhanced Editor.

7.2 Significance

The significance of this research work can be seen from three different angles:

1. **Usability**

   Successful application of SPE demands expertise in building models, understanding solvers and simulation and proficiency in interpreting results.

   The enhanced graphical interfaces of this research increase usability by:

   i. allowing easy model creation and eliminating the need to master LQN syntax,
   ii. providing validation of a model via the layered viewer and
   iii. Displaying results (if available).

2. **Traceability**

   The bottleneck strength and saturation breakdown viewer address SPE's fundamental quest: identifying software bottleneck and tracing its migration.

3. **Automation**

   Introducing a DOM parser with the capability of parsing results of a solved model enabled automation of bottleneck strength and saturation breakdown algorithms. This automation significantly reduces overall SPE analysis time and eliminates error-prone manual calculations.

7.3 Generalizations

Using the contributions of this research as a foundation, this Editor can be further enhanced to automatically mitigate bottlenecks. In addition, the Editor can be modified to pick and choose different LQN solvers or simulators depending on the outcome of the

runs. For example, if a model doesn't converge after few hundred iterations in LQNS, Editor can automatically switch to run the simulator. Furthermore, it can be converted into an SPE centric IDE (Integrated Development Environment).

References:

[1] Connie U. Smith, *"Performance Engineering of Software Systems"*, Addison-Wesley, 1990.

[2] Connie U. Smith, "The evolution of software performance engineering: a survey", *Proceedings of ACM Fall joint computer conference,* Dallas, Texas, pp.778-783, 1986

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

[4] Catalina M. Llado,Connie U. Smith , Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation, *Proceedings of the Quantitative Evaluation of Systems*, First International Conference on (QEST'04), pp.38-47, September 27-30, 2004

[5] Andreas Schmietendorf, André Scholz, Claus Rautenstrauch, "Evaluating the performance engineering process", *Proceedings of the 2nd international workshop on Software and performance*, 2000

[6] Dorina C.Petriu, Dorin B. Petriu, Hui Shen, Jose Merseguer, Murray Woodside, Toqueer Israr, "Performance by Unified Model Analysis (PUMA)", *Proceedings of the 5th international workshop on Software and performance*, July 12-14, 2005, Illes Balears, Spain.

[7] José Merseguer,Simona Bernardi , Susanna Donatelli , From UML sequence diagrams and statecharts to analysable petri net models, *Proceedings of the 3rd international workshop on Software and performance*, July 24-26, 2002, Rome, Italy

[8] Giuliana Franceschinis,Susanna Donatelli , The PSR Methodology: Integrating Hardware and Software Models, *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, pp.133-152, June 24-28, 1996

[9] Javier Campos, José Merseguer ,Juan Pablo López-Grao , From UML activity diagrams to Stochastic Petri nets: application to software performance engineering, *Proceedings of the 4th international workshop on Software and performance*, January 14-16, 2004, Redwood Shores, California

[10] C. Canevet , J. Hillston , L. Kloul , P. Stevens, S. Gilmore , Analysing UML 2.0 activity diagrams in the software performance engineering process, *Proceedings of the 4th international workshop on Software and performance*, January 14-16, 2004, Redwood Shores, California

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

[12] Dorina C. Petriu, John E. Neilson, C. Murray Woodside, and Shikharesh Majumdar, "The stochastic rendezvous network model for performance of synchronous client-server-like distributed systems," *IEEE Transactions on Computers*, vol. 44, no. 1, pp. 20 – 34, Jan. 1995.

[13] Murray Woodside, Tutorial Introduction to Layered Modeling of Software Performance, Edition 3.0, *http://www.sce.carleton.ca/rads/lqn/lqn-documentation/tutorialg.pdf*, May 2, 2002

[14] Mark H. Klein, "State of the practice report: Problems in the practice of software engineering," Tech. *Rep. CMU/SEI-95-TR-020*, Software Engineering Institute, Carnegie Mellon University, Feb. 1996.

[15] Peter Tregunno, *"Practical Analysis of Software Bottlenecks"*, Master's Thesis, Department of System and Computer Engineering, Carleton University, Ottawa. 5th May 2003

[16] Xiuping Wu, *"An Approach to Predicting Performance for Component Based Systems"*, Master's Thesis, Department of System and Computer Engineering, Carleton University, Ottawa. July 20th, 2003.

[17] Nikhil Barthwal, *"An Approach for building and evaluating performance models for component based models"*, Master's Thesis, Department of System and Computer Engineering, Carleton University, Ottawa. Nov. 2004.

[18] Greg Franks, *"Performance Analysis of Distributed Server Systems"*, Report OCIEE-00-01, Department of System and Computer Engineering, Carleton University, Ottawa. Jan. 2000

[19] H. Harreld, NASA Delays Satellite Launch After Finding Bugs in Software Program, *Federal Computer Week*, (1998).

[20] Robert F. Dugan, "Performance lies my professor told me: the case for teaching Software Performance Engineering to undergraduates", *Proceedings of the 4th international workshop on Software and performance*, January 14-16, 2004, Redwood Shores, California.

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

[22] Vittorio Cortellessa, "How far are we from the definition of a common software performance ontology?" *Proceedings of the 5th international workshop on Software and performance*, pp.195-204, July 12-14, 2005, Palma, Illes Balears, Spain

[23] Carlos Juiz, Günter Haring, Isaac Lera, Joachim Zottl, Ramon Puigjaner, "Performance assessment on ambient intelligent applications through ontologies", *Proceedings of the 5th international workshop on Software and performance*, p.205-216, July 12-14, 2005, Palma, Illes Balears, Spain

[24] Edward D.Lazowska, G.Scott Graham, John Zahorjan, Kenneth C. Sevick, *Quantitative System Performance*, 1984

[25] D. A. Menasce. "Two-Level Iterative Queuing Modeling of Software Contention", *Proceedings of MASCOTS 2002*, October, Fort Worth, Texas, USA. pp267-280

[26] C.M. Woodside, "Throughput Calculation for Basic Stochastic Rendezvous Networks", *Performance Evaluation*, Vol. 9, No. 2, April 1989, pp143-160

[27] Aninisca Di Marco, Marta Simeoni, Paola Inverardi, Simonetta Balsamo, "Software performance: State of the art and perspectives," *Technical Report MIUR SAHARA Project TR SAH/04*, Jan. 2003.

[28] C.M. Woodside, O.Das, "Dependable LQNS: A Performability Modeling Tool for Layered Systems" *13th Int Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS 2003)*, Urbana, Illinois, USA, Sept 2003.

[29] Dennis Heimbigner, "*jb -- Parser and Lexer Generation for Java*", Department of Computer Science, University of Colorado, http://www.cs.colorado.edu/serl/software , 7 November 2000.

[30] Cai M. R. Lyu, K. F. Wong R. Ko, "Component-based software engineering: technologies, development frameworks, and quality assurance schemes", Proceedings of APSEC 2000: *Software Engineering Conference*, 2000. Seventh Asia-Pacific, 5-8 Dec. 2000, pp 372 –379

[31] A. Liu, I. Gorton, S. Chen, Y. Liu, "Performance Prediction of COTS Component-based Enterprise Applications", *Proceedings of 5th ICSE Workshop on Component-Based Software Engineering: Benchmarks for Predictable Assembly*, http://www.sei.cmu.edu/pacc/CBSE5/liu-cbse5-29.pdf, Orlando, Florida USA, May 19-20, 2002

[32] J.Q. Ning, "Component-based software engineering (CBSE)", *Proceedings of the Fifth International Symposium on Assessment of Software Tools and Technologies*, 1997, 2-5 June 1997 pp 34 -35

[33] A.L.N. Reddy, G. Kulczycki, J. Krone, Sitaraman, W. F. Ogden "Performance Specification of Software Components", *Proceedings of SSR '01, pp. 3-10. ACM/SIGSOFT*, May 2001

[34] D. McMullan, M. Woodside and X. Wu. "Component Based Performance Prediction", *Proceedings of 6th ICSE Workshop on Component-Based Software Engineering; Automated Reasoning and Prediction*, pp13-18, Portland, Oregon, USA, May 3-4, 2003

[35] S. Yacoub. "Performance Analysis of Component-Based Applications", *Proceedings of the Second Software Product Line Conference, pp.299-315, San Diego*, CA, USA, August 2002

[36] A. Mili, H. Ammar and S. Yacoub, "Characterizing a Software Component", *http://www.sei.cmu.edu/cbs/icse99/papers/34/34.htm*, May 1999

[37] Al Sagaich and Michael C. Daconta , "*XML Development with JAVA 2*", October 2000.

[38] Piroz Mohseni, "Choose Your Java XML Parser", http://www.devx.com

[39] D. C. Petriu, C. M. Woodside, J. E. Neilson, and S. Majumdar, "Software bottlenecking in client-server systems and rendezvous networks," *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 776 – 782, Sept. 1995.

[40] Connie U. Simth and Lloyd G. Williams, "Software performance antipatterns,"in *Proceedings of the Second International Workshop on Software and Performance, WOSP'2000*, Ottawa, Ontario, Canada, Sept. 2000, pp.127 – 136a.

[41] Greg Hills, Giuseppe Serazzi, and Jerome Rolia"Performance engineering of distributed software process architecures," in *8th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation'95*, Heidelberg, Germany, Sept. 1995.

[42] Giuseppe Serazzi, Marin Litoiu and Jerome Rolia "Designing process replication and activation: A quantitative approach," *IEEE Transactions on Software Engineering*, vol. 26, no. 12, pp. 1168 – 1178, Dec. 2000.

[43] Telcordia Technologies, "LSSGR: service standards," *Generic Requirement GR-511-CORE*, June 2000.
[44] Telcordia Technologies, "LSSGR: traffic capacity and environment," *Generic Requirement GR-517-CORE*, Dec. 1998.
[45] Canadian Radio and Telecommunications Commission, *"Telecom decision 97-16: Quality of service indicators for use in telephone company regulation"* July 1997.

Appendix A:

Legacy LQN Model for VoP Switch

G

"Case Study"

0.0001

200

-1

P 0

p class5Switch_1 f i

p class5Switch_2 f i

Processors in the Call Connection Agent

p ss7Ip f

p h248Ip f

p dbServer f

p coco_1 f

Processors in the Gateway

p gwControl_1 f

p gwSvc_1 f

p gwH248_1 f

p gwNlc_1 f

p gwIlc_1 f

p gwControl_2 f

p gwSvc_2 f

p gwH248_2 f

p gwNlc_2 f

p gwIlc_2 f

```
# Processors in the Core Switch
p coreControl f

p coreSvc f

p coreNlc f

-1

T 0

t class5SwitchT_1 r class5SwitchE_1 -1 class5Switch_1 m 1000

t class5SwitchT_2 n class5SwitchIAM_2 class5SwitchREL_2 -1 class5Switch_2 m 500

t ss7IpInT n ss7IpInIamE ss7IpInRelE -1 ss7Ip m 150

t ss7IpOutT n ss7IpOutIamE ss7IpOutRelE -1 ss7Ip m 150

t h248IpT n h248IpAddE h248ModifyE h248SubtractE -1 h248Ip m 20

t cocoT_1 n cocoSetupE_1 cocoReleaseE_1 -1 coco_1 m 10

t dbServerT n dbServerGetE dbServerSetE -1 dbServer

# One control, one Voice CallP, one SVC, 3 network line, 6 interface line

t gwControlT_1 n gwControlRouteRequestE_1 gwControlDisconnectE_1 -1 gwControl_1

t gwSvcT_1 n gwSvcSetupE_1 gwSvcConnectE_1 gwSvcReleaseE_1 -1 gwSvc_1 m 20

t gwH248T_1 n gwH248AddE_1 gwH248ModifyE_1 gwH248SubtractE_1 -1 gwH248_1 m 10

t gwCacoT_1 n gwCacoAddE_1 gwCacoModifyE_1 gwCacoSubtractE_1 -1 gwH248_1 m 4

t gwNlcT_1 n gwNlcPt2PtSetE_1 -1 gwNlc_1

t gwIlcT_1 n gwIlcPt2PtSetE_1 -1 gwIlc_1

# One control, one Voice CallP, one SVC, 3 network line, 6 interface line

t gwControlT_2 n gwControlRouteRequestE_2 gwControlDisconnectE_2 -1 gwControl_2

t gwSvcT_2 n gwSvcSetupE_2 gwSvcConnectE_2 gwSvcReleaseE_2 -1 gwSvc_2 m 20

t gwH248T_2 n gwH248AddE_2 gwH248ModifyE_2 gwH248SubtractE_2 -1 gwH248_2 m 10

t gwCacoT_2 n gwCacoAddE_2 gwCacoModifyE_2 gwCacoSubtractE_2 -1 gwH248_2 m 4

t gwNlcT_2 n gwNlcPt2PtSetE_2 -1 gwNlc_2 m 3

t gwIlcT_2 n gwIlcPt2PtSetE_2 -1 gwIlc_2 m 6

# One control, one svc, 12 line cards
```

t coreControlT n coreControlRouteRequestE coreControlDisconnectE -1 coreControl

t coreSvcT n coreSvcSetupE coreSvcConnectE coreSvcReleaseE -1 coreSvc m 125

t coreNlcT n coreNlcPt2PtSetE -1 coreNlc m 12

-1

E 0

Gateway 1 is the terminating gateway

Gateway 2 is the originating gateway

we have 500 000 users, each making 1 call per hour. That means that

there is 3600 seconds of sleep time

for practical reasons, we have 1000 users, with 7.2 seconds of sleep time

s class5SwitchE_1 0 3.6 3.6 -1

y class5SwitchE_1 ss7IpInIamE 0 0 1 -1

y class5SwitchE_1 ss7IpInRelE 0 1 0 -1

set the service time for the IAM at the terminating switch to

be 50 msec

s class5SwitchIAM_2 0.050 0 -1

s class5SwitchREL_2 0.050 0 -1

s ss7IpInIamE 0.000030 0 -1

y ss7IpInIamE cocoSetupE_1 1 0 -1

s ss7IpInRelE 0.000030 0 -1

y ss7IpInRelE cocoReleaseE_1 1 0 -1

s ss7IpOutIamE 0.000030 0 -1

y ss7IpOutIamE class5SwitchIAM_2 1 0 -1

s ss7IpOutRelE 0.000030 0 -1

y ss7IpOutRelE class5SwitchREL_2 1 0 -1

s h248IpAddE 0.000050 0 -1

y h248IpAddE gwH248AddE_1 0.5 0 -1

y h248IpAddE gwH248AddE_2 0.5 0 -1

s h248ModifyE 0.000025 0 -1

y h248ModifyE gwH248ModifyE_2 1 0 -1

s h248SubtractE 0.000030 0 -1

y h248SubtractE gwH248SubtractE_1 0.5 0 -1

y h248SubtractE gwH248SubtractE_2 0.5 0 -1

s cocoSetupE_1 0.000800 0.000050 -1

y cocoSetupE_1 h248IpAddE 2 0 -1

y cocoSetupE_1 h248ModifyE 1 0 -1

y cocoSetupE_1 dbServerGetE 1 0 -1

y cocoSetupE_1 dbServerSetE 0 1 -1

y cocoSetupE_1 ss7IpOutIamE 1 0 -1

s cocoReleaseE_1 0.000200 0.000050 -1

y cocoReleaseE_1 h248SubtractE 2 0 -1

y cocoReleaseE_1 dbServerGetE 1 0 -1

y cocoReleaseE_1 dbServerSetE 0 1 -1

y cocoReleaseE_1 ss7IpOutRelE 1 0 -1

s dbServerGetE 0.000025 0 -1

s dbServerSetE 0.000100 0 -1

s gwControlRouteRequestE_1 0.000040 0 -1

s gwControlDisconnectE_1 0.000080 0 -1

y gwControlDisconnectE_1 gwNlcPt2PtSetE_1 1 0 -1

y gwControlDisconnectE_1 gwIlcPt2PtSetE_1 1 0 -1

s gwSvcSetupE_1 0.000800 0.000050 -1

y gwSvcSetupE_1 gwControlRouteRequestE_1 1 0 -1

y gwSvcSetupE_1 coreSvcSetupE 1 0 -1

y gwSvcSetupE_1 gwNlcPt2PtSetE_1 1 0 -1

y gwSvcSetupE_1 gwIlcPt2PtSetE_1 1 0 -1

y gwSvcSetupE_1 coreSvcConnectE 1 0 -1

s gwSvcReleaseE_1 0.000500 0.000050 -1

y gwSvcReleaseE_1 gwControlDisconnectE_1 1 0 -1

y gwSvcReleaseE_1 coreSvcReleaseE 1 0 -1

s gwSvcConnectE_1 0.000750 0.000050 -1

s gwH248AddE_1 0.000020 0 -1

y gwH248AddE_1 gwCacoAddE_1 1 0 -1

s gwH248ModifyE_1 0.000010 0 -1

y gwH248ModifyE_1 gwCacoModifyE_1 1 0 -1

s gwH248SubtractE_1 0.000020 0 -1

y gwH248SubtractE_1 gwCacoSubtractE_1 1 0 -1

s gwCacoAddE_1 0.000800 0.000100 -1

y gwCacoAddE_1 gwSvcSetupE_1 1 0 -1

s gwCacoModifyE_1 0.000100 0.000050 -1

s gwCacoSubtractE_1 0.000100 0.000050 -1

y gwCacoSubtractE_1 gwSvcReleaseE_1 1 0 -1

s gwIlcPt2PtSetE_1 0.001000 0 -1

#End of Gateway number 1

#Start of Gateway number 2

s gwControlRouteRequestE_2 0.000040 0 -1

s gwControlDisconnectE_2 0.000080 0 -1

y gwControlDisconnectE_2 gwNlcPt2PtSetE_2 1 0 -1

y gwControlDisconnectE_2 gwIlcPt2PtSetE_2 1 0 -1

s gwSvcSetupE_2 0.000800 0.000050 -1

y gwSvcSetupE_2 gwControlRouteRequestE_2 1 0 -1

y gwSvcSetupE_2 gwNlcPt2PtSetE_2 1 0 -1

y gwSvcSetupE_2 gwIlcPt2PtSetE_2 1 0 -1

s gwSvcReleaseE_2 0.000500 0.000050 -1

y gwSvcReleaseE_2 gwControlDisconnectE_2 1 0 -1

s gwH248AddE_2 0.000020 0 -1

y gwH248AddE_2 gwCacoAddE_2 1 0 -1

s gwH248ModifyE_2 0.000010 0 -1
y gwH248ModifyE_2 gwCacoModifyE_2 1 0 -1

s gwH248SubtractE_2 0.000020 0 -1

y gwH248SubtractE_2 gwCacoSubtractE_2 1 0 -1

s gwCacoAddE_2 0.000800 0.000050 -1

s gwCacoModifyE_2 0.000100 0.000050 -1

s gwCacoSubtractE_2 0.000100 0.000050 -1

#y gwCacoSubtractE_2 gwControlDisconnectE_2 1 0 -1

s gwNlcPt2PtSetE_2 0.001000 0 -1

s gwIlcPt2PtSetE_2 0.001000 0 -1

start of the core switch

s coreControlRouteRequestE 0.000040 0 -1

s coreControlDisconnectE 0.000080 0 -1

s coreSvcSetupE 0.000800 0.000050 -1

y coreSvcSetupE coreControlRouteRequestE 1 0 -1

y coreSvcSetupE coreNlcPt2PtSetE 2 0 -1

y coreSvcSetupE gwSvcSetupE_2 1 0 -1

s coreSvcConnectE 0.000750 0.000050 -1

y coreSvcConnectE gwSvcConnectE_2 1 0 -1

s coreSvcReleaseE 0.000500 0.000050 -1

y coreSvcReleaseE coreControlDisconnectE 1 0 -1

y coreSvcReleaseE gwSvcReleaseE_2 1 0 -1

s coreNlcPt2PtSetE 0.001000 0 -1

-1

Appendix B:

XML Version VoP Model

```
<?xml version="1.0"?>
<!-- Invoked as: lqn2xml C:\lqnmodels\tempExp\caseStudy.lqn -->
<!-- Sun Sep 11 08:17:56 2005 -->

<lqn-model name="C:\lqnmodels\tempExp\caseStudy" description="Generated by: lqn2xml, version 3.6"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///Program Files/LQN Solvers/lqn.xsd">
  <solver-params comment="Case Study" conv_val="0.0001" it_limit="200" print_int="10"
underrelax_coeff="0.9"/>
  <processor name="class5Switch_1" multiplicity="0" scheduling="fcfs">
    <task name="class5SwitchT_1" multiplicity="1000" scheduling="ref">
      <entry name="class5SwitchE_1" type="PH1PH2">
        <entry-phase-activities>
          <activity name="class5SwitchE_1_ph2" phase="2" host-demand-mean="3.6">
            <synch-call dest="ss7IpInRelE" calls-mean="1"/>
          </activity>
          <activity name="class5SwitchE_1_ph3" phase="3" host-demand-mean="3.6">
            <synch-call dest="ss7IpInIamE" calls-mean="1"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="coco_1" scheduling="fcfs">
    <task name="cocoT_1" multiplicity="10" scheduling="fcfs">
      <entry name="cocoSetupE_1" type="PH1PH2">
        <entry-phase-activities>
          <activity name="cocoSetupE_1_ph1" phase="1" host-demand-mean="0.0008">
            <synch-call dest="h248IpAddE" calls-mean="2"/>
            <synch-call dest="h248ModifyE" calls-mean="1"/>
            <synch-call dest="dbServerGetE" calls-mean="1"/>
            <synch-call dest="ss7IpOutIamE" calls-mean="1"/>
          </activity>
          <activity name="cocoSetupE_1_ph2" phase="2" host-demand-mean="5e-005">
```

```
          <synch-call dest="dbServerSetE" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="cocoReleaseE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="cocoReleaseE_1_ph1" phase="1" host-demand-mean="0.0002">
          <synch-call dest="h248SubtractE" calls-mean="2"/>
          <synch-call dest="dbServerGetE" calls-mean="1"/>
          <synch-call dest="ss7IpOutRelE" calls-mean="1"/>
        </activity>
        <activity name="cocoReleaseE_1_ph2" phase="2" host-demand-mean="5e-005">
          <synch-call dest="dbServerSetE" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="dbServer" scheduling="fcfs">
  <task name="dbServerT" scheduling="fcfs">
    <entry name="dbServerGetE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="dbServerGetE_ph1" phase="1" host-demand-mean="2.5e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="dbServerSetE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="dbServerSetE_ph1" phase="1" host-demand-mean="0.0001"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="h248Ip" scheduling="fcfs">
  <task name="h248IpT" multiplicity="20" scheduling="fcfs">
    <entry name="h248IpAddE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="h248IpAddE_ph1" phase="1" host-demand-mean="5e-005">
```

```
            <synch-call dest="gwH248AddE_1" calls-mean="0.5"/>
            <synch-call dest="gwH248AddE_2" calls-mean="0.5"/>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="h248ModifyE" type="PH1PH2">
        <entry-phase-activities>
          <activity name="h248ModifyE_ph1" phase="1" host-demand-mean="2.5e-005">
            <synch-call dest="gwH248ModifyE_2" calls-mean="1"/>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="h248SubtractE" type="PH1PH2">
        <entry-phase-activities>
          <activity name="h248SubtractE_ph1" phase="1" host-demand-mean="3e-005">
            <synch-call dest="gwH248SubtractE_1" calls-mean="0.5"/>
            <synch-call dest="gwH248SubtractE_2" calls-mean="0.5"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="ss7Ip" scheduling="fcfs">
    <task name="ss7IpInT" multiplicity="150" scheduling="fcfs">
      <entry name="ss7IpInIamE" type="PH1PH2">
        <entry-phase-activities>
          <activity name="ss7IpInIamE_ph1" phase="1" host-demand-mean="3e-005">
            <synch-call dest="cocoSetupE_1" calls-mean="1"/>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="ss7IpInRelE" type="PH1PH2">
        <entry-phase-activities>
          <activity name="ss7IpInRelE_ph1" phase="1" host-demand-mean="3e-005">
            <synch-call dest="cocoReleaseE_1" calls-mean="1"/>
          </activity>
        </entry-phase-activities>
```

```
    </entry>
  </task>
  <task name="ss7IpOutT" multiplicity="150" scheduling="fcfs">
    <entry name="ss7IpOutIamE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="ss7IpOutIamE_ph1" phase="1" host-demand-mean="3e-005">
          <synch-call dest="class5SwitchIAM_2" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="ss7IpOutRelE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="ss7IpOutRelE_ph1" phase="1" host-demand-mean="3e-005">
          <synch-call dest="class5SwitchREL_2" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="class5Switch_2" multiplicity="0" scheduling="fcfs">
  <task name="class5SwitchT_2" multiplicity="500" scheduling="fcfs">
    <entry name="class5SwitchIAM_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="class5SwitchIAM_2_ph1" phase="1" host-demand-mean="0.05"/>
      </entry-phase-activities>
    </entry>
    <entry name="class5SwitchREL_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="class5SwitchREL_2_ph1" phase="1" host-demand-mean="0.05"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwH248_1" scheduling="fcfs">
  <task name="gwH248T_1" multiplicity="10" scheduling="fcfs">
    <entry name="gwH248AddE_1" type="PH1PH2">
      <entry-phase-activities>
```

```
      <activity name="gwH248AddE_1_ph1" phase="1" host-demand-mean="2e-005">
        <synch-call dest="gwCacoAddE_1" calls-mean="1"/>
      </activity>
    </entry-phase-activities>
  </entry>
  <entry name="gwH248SubtractE_1" type="PH1PH2">
    <entry-phase-activities>
      <activity name="gwH248SubtractE_1_ph1" phase="1" host-demand-mean="2e-005">
        <synch-call dest="gwCacoSubtractE_1" calls-mean="1"/>
      </activity>
    </entry-phase-activities>
  </entry>
  <entry name="gwH248ModifyE_1" type="PH1PH2">
    <entry-phase-activities>
      <activity name="gwH248ModifyE_1_ph1" phase="1" host-demand-mean="1e-005">
        <synch-call dest="gwCacoModifyE_1" calls-mean="1"/>
      </activity>
    </entry-phase-activities>
  </entry>
</task>
<task name="gwCacoT_1" multiplicity="4" scheduling="fcfs">
  <entry name="gwCacoAddE_1" type="PH1PH2">
    <entry-phase-activities>
      <activity name="gwCacoAddE_1_ph1" phase="1" host-demand-mean="0.0008">
        <synch-call dest="gwSvcSetupE_1" calls-mean="1"/>
      </activity>
      <activity name="gwCacoAddE_1_ph2" phase="2" host-demand-mean="0.0001"/>
    </entry-phase-activities>
  </entry>
  <entry name="gwCacoSubtractE_1" type="PH1PH2">
    <entry-phase-activities>
      <activity name="gwCacoSubtractE_1_ph1" phase="1" host-demand-mean="0.0001">
        <synch-call dest="gwSvcReleaseE_1" calls-mean="1"/>
      </activity>
      <activity name="gwCacoSubtractE_1_ph2" phase="2" host-demand-mean="5e-005"/>
    </entry-phase-activities>
  </entry>
```

```
    <entry name="gwCacoModifyE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwCacoModifyE_1_ph1" phase="1" host-demand-mean="0.0001"/>
        <activity name="gwCacoModifyE_1_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwH248_2" scheduling="fcfs">
  <task name="gwH248T_2" multiplicity="10" scheduling="fcfs">
    <entry name="gwH248AddE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwH248AddE_2_ph1" phase="1" host-demand-mean="2e-005">
          <synch-call dest="gwCacoAddE_2" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="gwH248ModifyE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwH248ModifyE_2_ph1" phase="1" host-demand-mean="1e-005">
          <synch-call dest="gwCacoModifyE_2" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="gwH248SubtractE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwH248SubtractE_2_ph1" phase="1" host-demand-mean="2e-005">
          <synch-call dest="gwCacoSubtractE_2" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
  <task name="gwCacoT_2" multiplicity="4" scheduling="fcfs">
    <entry name="gwCacoAddE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwCacoAddE_2_ph1" phase="1" host-demand-mean="0.0008"/>
        <activity name="gwCacoAddE_2_ph2" phase="2" host-demand-mean="5e-005"/>
```

```
      </entry-phase-activities>
    </entry>
    <entry name="gwCacoModifyE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwCacoModifyE_2_ph1" phase="1" host-demand-mean="0.0001"/>
        <activity name="gwCacoModifyE_2_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="gwCacoSubtractE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwCacoSubtractE_2_ph1" phase="1" host-demand-mean="0.0001"/>
        <activity name="gwCacoSubtractE_2_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwSvc_1" scheduling="fcfs">
  <task name="gwSvcT_1" multiplicity="20" scheduling="fcfs">
    <entry name="gwSvcSetupE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwSvcSetupE_1_ph1" phase="1" host-demand-mean="0.0008">
          <synch-call dest="gwControlRouteRequestE_1" calls-mean="1"/>
          <synch-call dest="coreSvcSetupE" calls-mean="1"/>
          <synch-call dest="gwNlcPt2PtSetE_1" calls-mean="1"/>
          <synch-call dest="gwIlcPt2PtSetE_1" calls-mean="1"/>
          <synch-call dest="coreSvcConnectE" calls-mean="1"/>
        </activity>
        <activity name="gwSvcSetupE_1_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="gwSvcReleaseE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwSvcReleaseE_1_ph1" phase="1" host-demand-mean="0.0005">
          <synch-call dest="gwControlDisconnectE_1" calls-mean="1"/>
          <synch-call dest="coreSvcReleaseE" calls-mean="1"/>
        </activity>
        <activity name="gwSvcReleaseE_1_ph2" phase="2" host-demand-mean="5e-005"/>
```

```
      </entry-phase-activities>
    </entry>
    <entry name="gwSvcConnectE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwSvcConnectE_1_ph1" phase="1" host-demand-mean="0.00075"/>
        <activity name="gwSvcConnectE_1_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="coreSvc" scheduling="fcfs">
  <task name="coreSvcT" multiplicity="125" scheduling="fcfs">
    <entry name="coreSvcConnectE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="coreSvcConnectE_ph1" phase="1" host-demand-mean="0.00075">
          <synch-call dest="gwSvcConnectE_2" calls-mean="1"/>
        </activity>
        <activity name="coreSvcConnectE_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="coreSvcSetupE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="coreSvcSetupE_ph1" phase="1" host-demand-mean="0.0008">
          <synch-call dest="coreControlRouteRequestE" calls-mean="1"/>
          <synch-call dest="coreNlcPt2PtSetE" calls-mean="2"/>
          <synch-call dest="gwSvcSetupE_2" calls-mean="1"/>
        </activity>
        <activity name="coreSvcSetupE_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="coreSvcReleaseE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="coreSvcReleaseE_ph1" phase="1" host-demand-mean="0.0005">
          <synch-call dest="coreControlDisconnectE" calls-mean="1"/>
          <synch-call dest="gwSvcReleaseE_2" calls-mean="1"/>
        </activity>
        <activity name="coreSvcReleaseE_ph2" phase="2" host-demand-mean="5e-005"/>
```

```
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwControl_1" scheduling="fcfs">
  <task name="gwControlT_1" scheduling="fcfs">
    <entry name="gwControlRouteRequestE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwControlRouteRequestE_1_ph1" phase="1" host-demand-mean="4e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="gwControlDisconnectE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwControlDisconnectE_1_ph1" phase="1" host-demand-mean="8e-005">
          <synch-call dest="gwNlcPt2PtSetE_1" calls-mean="1"/>
          <synch-call dest="gwIlcPt2PtSetE_1" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwSvc_2" scheduling="fcfs">
  <task name="gwSvcT_2" multiplicity="20" scheduling="fcfs">
    <entry name="gwSvcConnectE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwSvcConnectE_2_ph1" phase="1" host-demand-mean="0.00075"/>
        <activity name="gwSvcConnectE_2_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="gwSvcSetupE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwSvcSetupE_2_ph1" phase="1" host-demand-mean="0.0008">
          <synch-call dest="gwControlRouteRequestE_2" calls-mean="1"/>
          <synch-call dest="gwNlcPt2PtSetE_2" calls-mean="1"/>
          <synch-call dest="gwIlcPt2PtSetE_2" calls-mean="1"/>
        </activity>
        <activity name="gwSvcSetupE_2_ph2" phase="2" host-demand-mean="5e-005"/>
```

```
      </entry-phase-activities>
    </entry>
    <entry name="gwSvcReleaseE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwSvcReleaseE_2_ph1" phase="1" host-demand-mean="0.0005">
          <synch-call dest="gwControlDisconnectE_2" calls-mean="1"/>
        </activity>
        <activity name="gwSvcReleaseE_2_ph2" phase="2" host-demand-mean="5e-005"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="coreControl" scheduling="fcfs">
  <task name="coreControlT" scheduling="fcfs">
    <entry name="coreControlRouteRequestE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="coreControlRouteRequestE_ph1" phase="1" host-demand-mean="4e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="coreControlDisconnectE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="coreControlDisconnectE_ph1" phase="1" host-demand-mean="8e-005">
          <synch-call dest="coreNlcPt2PtSetE" calls-mean="2"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwIlc_1" scheduling="fcfs">
  <task name="gwIlcT_1" scheduling="fcfs">
    <entry name="gwIlcPt2PtSetE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwIlcPt2PtSetE_1_ph1" phase="1" host-demand-mean="0.001"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
```

```
<processor name="gwNlc_1" scheduling="fcfs">
  <task name="gwNlcT_1" scheduling="fcfs">
    <entry name="gwNlcPt2PtSetE_1" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwNlcPt2PtSetE_1_ph1" phase="1" host-demand-mean="0.001"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwControl_2" scheduling="fcfs">
  <task name="gwControlT_2" scheduling="fcfs">
    <entry name="gwControlRouteRequestE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwControlRouteRequestE_2_ph1" phase="1" host-demand-mean="4e-005"/>
      </entry-phase-activities>
    </entry>
    <entry name="gwControlDisconnectE_2" type="PH1PH2">
      <entry-phase-activities>
        <activity name="gwControlDisconnectE_2_ph1" phase="1" host-demand-mean="8e-005">
          <synch-call dest="gwNlcPt2PtSetE_2" calls-mean="1"/>
          <synch-call dest="gwIlcPt2PtSetE_2" calls-mean="1"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="coreNlc" scheduling="fcfs">
  <task name="coreNlcT" multiplicity="12" scheduling="fcfs">
    <entry name="coreNlcPt2PtSetE" type="PH1PH2">
      <entry-phase-activities>
        <activity name="coreNlcPt2PtSetE_ph1" phase="1" host-demand-mean="0.001"/>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwIlc_2" scheduling="fcfs">
  <task name="gwIlcT_2" multiplicity="6" scheduling="fcfs">
```

```
      <entry name="gwIlcPt2PtSetE_2" type="PH1PH2">
        <entry-phase-activities>
          <activity name="gwIlcPt2PtSetE_2_ph1" phase="1" host-demand-mean="0.001"/>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="gwNlc_2" scheduling="fcfs">
    <task name="gwNlcT_2" multiplicity="3" scheduling="fcfs">
      <entry name="gwNlcPt2PtSetE_2" type="PH1PH2">
        <entry-phase-activities>
          <activity name="gwNlcPt2PtSetE_2_ph1" phase="1" host-demand-mean="0.001"/>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
</lqn-model>
```

Appendix C: XML VoP Model with Embedded.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?><!-- Invoked as: lqn2xml caseStudy.lqn --><!-
- Wed Jun 29 18:33:10 2005 --><lqn-model description="Generated by: lqn2xml, version 3.6"
name="caseStudy" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///Program Files/LQN Solvers/lqn.xsd">
  <solver-params comment="Case Study" conv_val="0.0001" it_limit="250" print_int="10"
underrelax_coeff="0.9">
  <result-general conv-val="6.57171e-004" elapsed-time="0:00:00.10" iterations="2.50000e+002" solver-
info="lqns 3.6" valid="NO"/>
  </solver-params>
  <processor multiplicity="0" name="class5Switch_1" scheduling="fcfs">
  <result-processor utilization="6.22422e+002"/>
    <task multiplicity="1000" name="class5SwitchT_1" scheduling="ref">
    <result-task phase2-utilization="4.99783e+002" phase3-utilization="5.00217e+002" proc-
utilization="6.22422e+002" throughput="8.64475e+001" utilization="1.00000e+003"/>
      <entry name="class5SwitchE_1" type="PH1PH2">
      <result-entry proc-utilization="6.22422e+002" squared-coeff-variation="6.35036e-001"
throughput="8.64475e+001" throughput-bound="1.36591e+002" utilization="1.00000e+003"/>
        <entry-phase-activities>
```

```
<activity host-demand-mean="3.6" name="class5SwitchE_1_ph2" phase="2">
<result-activity proc-waiting="0.00000e+000" service-time="5.78135e+000" service-time-variance="4.24375e+001" utilization="4.99783e+002"/>
<synch-call calls-mean="1" dest="ss7IpInRelE">
<result-call waiting="1.19030e+000"/>
</synch-call>
</activity>
<activity host-demand-mean="3.6" name="class5SwitchE_1_ph3" phase="3">
<result-activity proc-waiting="0.00000e+000" service-time="5.78636e+000" service-time-variance="4.25380e+001" utilization="5.00217e+002"/>
<synch-call calls-mean="1" dest="ss7IpInIamE">
<result-call waiting="1.19030e+000"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
</task>
</processor>
<processor name="coco_1" scheduling="fcfs">
<result-processor utilization="8.30350e-002"/>
<task multiplicity="10" name="cocoT_1" scheduling="fcfs">
<result-task phase1-utilization="9.97156e+000" phase2-utilization="2.54424e-002" proc-utilization="8.30350e-002" throughput="1.50973e+002" utilization="1.51847e+001"/>
<entry name="cocoSetupE_1" type="PH1PH2">
<result-entry proc-utilization="6.41634e-002" squared-coeff-variation="3.07237e+000" throughput="7.54864e+001" throughput-bound="1.58983e+002" utilization="5.18772e+000"/>
<entry-phase-activities>
<activity host-demand-mean="0.0008" name="cocoSetupE_1_ph1" phase="1">
<result-activity proc-waiting="5.11677e-005" service-time="6.85554e-002" service-time-variance="1.45107e-002" utilization="5.17500e+000"/>
<synch-call calls-mean="2" dest="h248IpAddE">
<result-call waiting="0.00000e+000"/>
</synch-call>
<synch-call calls-mean="1" dest="h248ModifyE">
<result-call waiting="0.00000e+000"/>
</synch-call>
<synch-call calls-mean="1" dest="dbServerGetE">
```

<result-call waiting="1.46733e-006"/>
</synch-call>
<synch-call calls-mean="1" dest="ss7IpOutIamE">
<result-call waiting="0.00000e+000"/>
</synch-call>
</activity>
<activity host-demand-mean="5e-005" name="cocoSetupE_1_ph2" phase="2">
<result-activity proc-waiting="1.70559e-005" service-time="1.68523e-004" service-time-variance="4.78451e-008" utilization="1.27212e-002"/>
<synch-call calls-mean="1" dest="dbServerSetE">
<result-call waiting="1.46733e-006"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
<entry name="cocoReleaseE_1" type="PH1PH2">
<result-entry proc-utilization="1.88716e-002" squared-coeff-variation="3.40111e+000" throughput="7.54864e+001" throughput-bound="1.71101e+002" utilization="4.80928e+000"/>
<entry-phase-activities>
<activity host-demand-mean="0.0002" name="cocoReleaseE_1_ph1" phase="1">
<result-activity proc-waiting="4.26398e-005" service-time="6.35421e-002" service-time-variance="1.38052e-002" utilization="4.79656e+000"/>
<synch-call calls-mean="2" dest="h248SubtractE">
<result-call waiting="0.00000e+000"/>
</synch-call>
<synch-call calls-mean="1" dest="dbServerGetE">
<result-call waiting="1.46733e-006"/>
</synch-call>
<synch-call calls-mean="1" dest="ss7IpOutRelE">
<result-call waiting="0.00000e+000"/>
</synch-call>
</activity>
<activity host-demand-mean="5e-005" name="cocoReleaseE_1_ph2" phase="2">
<result-activity proc-waiting="1.70559e-005" service-time="1.68523e-004" service-time-variance="4.78451e-008" utilization="1.27212e-002"/>
<synch-call calls-mean="1" dest="dbServerSetE">
<result-call waiting="1.46733e-006"/>

```
            </synch-call>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="dbServer" scheduling="fcfs">
  <result-processor utilization="1.88716e-002"/>
    <task name="dbServerT" scheduling="fcfs">
    <result-task phase1-utilization="1.88716e-002" proc-utilization="1.88716e-002"
throughput="3.01946e+002" utilization="2.26459e-002"/>
      <entry name="dbServerGetE" type="PH1PH2">
      <result-entry proc-utilization="3.77432e-003" squared-coeff-variation="1.00000e+000"
throughput="1.50973e+002" throughput-bound="4.00000e+004" utilization="3.77432e-003"/>
        <entry-phase-activities>
          <activity host-demand-mean="2.5e-005" name="dbServerGetE_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="2.50000e-005" service-time-
variance="6.25000e-010" utilization="3.77432e-003"/>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="dbServerSetE" type="PH1PH2">
      <result-entry proc-utilization="1.50973e-002" squared-coeff-variation="1.00000e+000"
throughput="1.50973e+002" throughput-bound="1.00000e+004" utilization="1.50973e-002"/>
        <entry-phase-activities>
          <activity host-demand-mean="0.0001" name="dbServerSetE_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="1.00000e-004" service-time-
variance="1.00000e-008" utilization="1.50973e-002"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="h248Ip" scheduling="fcfs">
  <result-processor utilization="1.39650e-002"/>
    <task multiplicity="20" name="h248IpT" scheduling="fcfs">
```

<result-task phase1-utilization="2.34146e+000" proc-utilization="1.39650e-002" throughput="3.77432e+002" utilization="4.99485e+000"/>
<entry name="h248IpAddE" type="PH1PH2">
<result-entry proc-utilization="7.54864e-003" squared-coeff-variation="9.94674e+000" throughput="1.50973e+002" throughput-bound="3.40136e+003" utilization="1.31624e+000"/>
<entry-phase-activities>
<activity host-demand-mean="5e-005" name="h248IpAddE_ph1" phase="1">
<result-activity proc-waiting="5.10132e-007" service-time="8.71841e-003" service-time-variance="7.56059e-004" utilization="1.31624e+000"/>
<synch-call calls-mean="0.5" dest="gwH248AddE_1">
<result-call waiting="7.55488e-007"/>
</synch-call>
<synch-call calls-mean="0.5" dest="gwH248AddE_2">
<result-call waiting="2.71051e-020"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
<entry name="h248ModifyE" type="PH1PH2">
<result-entry proc-utilization="1.88716e-003" squared-coeff-variation="3.05303e+000" throughput="7.54864e+001" throughput-bound="1.48148e+005" utilization="2.08986e-002"/>
<entry-phase-activities>
<activity host-demand-mean="2.5e-005" name="h248ModifyE_ph1" phase="1">
<result-activity proc-waiting="5.10132e-007" service-time="2.76852e-004" service-time-variance="2.34006e-007" utilization="2.08986e-002"/>
<synch-call calls-mean="1" dest="gwH248ModifyE_2">
<result-call waiting="0.00000e+000"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
<entry name="h248SubtractE" type="PH1PH2">
<result-entry proc-utilization="4.52918e-003" squared-coeff-variation="1.16072e+001" throughput="1.50973e+002" throughput-bound="4.97512e+003" utilization="1.00432e+000"/>
<entry-phase-activities>
<activity host-demand-mean="3e-005" name="h248SubtractE_ph1" phase="1">

<result-activity proc-waiting="5.10132e-007" service-time="6.65231e-003" service-time-variance="5.13655e-004" utilization="1.00432e+000"/>
<synch-call calls-mean="0.5" dest="gwH248SubtractE_1">
<result-call waiting="7.55488e-007"/>
</synch-call>
<synch-call calls-mean="0.5" dest="gwH248SubtractE_2">
<result-call waiting="0.00000e+000"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
</task>
</processor>
<processor name="ss7Ip" scheduling="fcfs">
<result-processor utilization="9.71603e-003"/>
<task multiplicity="150" name="ss7IpInT" scheduling="fcfs">
<result-task phase1-utilization="1.71781e+002" proc-utilization="5.18685e-003" throughput="1.72895e+002" utilization="2.57888e+002"/>
<entry name="ss7IpInIamE" type="PH1PH2">
<result-entry proc-utilization="2.59342e-003" squared-coeff-variation="2.88159e+000" throughput="8.64475e+001" throughput-bound="2.38930e+003" utilization="8.61072e+001"/>
<entry-phase-activities>
<activity host-demand-mean="3e-005" name="ss7IpInIamE_ph1" phase="1">
<result-activity proc-waiting="2.71407e-007" service-time="9.96063e-001" service-time-variance="2.85895e+000" utilization="8.61072e+001"/>
<synch-call calls-mean="1" dest="cocoSetupE_1">
<result-call waiting="9.27478e-001"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
<entry name="ss7IpInRelE" type="PH1PH2">
<result-entry proc-utilization="2.59342e-003" squared-coeff-variation="2.88982e+000" throughput="8.64475e+001" throughput-bound="2.57180e+003" utilization="8.56738e+001"/>
<entry-phase-activities>
<activity host-demand-mean="3e-005" name="ss7IpInRelE_ph1" phase="1">

<result-activity proc-waiting="2.71407e-007" service-time="9.91050e-001" service-time-variance="2.83832e+000" utilization="8.56738e+001"/>
<synch-call calls-mean="1" dest="cocoReleaseE_1">
<result-call waiting="9.27478e-001"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
</task>
<task multiplicity="150" name="ss7IpOutT" scheduling="fcfs">
<result-task phase1-utilization="7.55320e+000" proc-utilization="4.52918e-003" throughput="1.50973e+002" utilization="1.13298e+001"/>
<entry name="ss7IpOutIamE" type="PH1PH2">
<result-entry proc-utilization="2.26459e-003" squared-coeff-variation="2.99758e+000" throughput="7.54864e+001" throughput-bound="2.99820e+003" utilization="3.77660e+000"/>
<entry-phase-activities>
<activity host-demand-mean="3e-005" name="ss7IpOutIamE_ph1" phase="1">
<result-activity proc-waiting="2.58594e-007" service-time="5.00302e-002" service-time-variance="7.50302e-003" utilization="3.77660e+000"/>
<synch-call calls-mean="1" dest="class5SwitchIAM_2">
<result-call waiting="0.00000e+000"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
<entry name="ss7IpOutRelE" type="PH1PH2">
<result-entry proc-utilization="2.26459e-003" squared-coeff-variation="2.99758e+000" throughput="7.54864e+001" throughput-bound="2.99820e+003" utilization="3.77660e+000"/>
<entry-phase-activities>
<activity host-demand-mean="3e-005" name="ss7IpOutRelE_ph1" phase="1">
<result-activity proc-waiting="2.58594e-007" service-time="5.00302e-002" service-time-variance="7.50302e-003" utilization="3.77660e+000"/>
<synch-call calls-mean="1" dest="class5SwitchREL_2">
<result-call waiting="0.00000e+000"/>
</synch-call>
</activity>
</entry-phase-activities>

```
      </entry>
    </task>
  </processor>
  <processor multiplicity="0" name="class5Switch_2" scheduling="fcfs">
  <result-processor utilization="7.54864e+000"/>
    <task multiplicity="500" name="class5SwitchT_2" scheduling="fcfs">
    <result-task phase1-utilization="7.54864e+000" proc-utilization="7.54864e+000"
throughput="1.50973e+002" utilization="1.13230e+001"/>
      <entry name="class5SwitchIAM_2" type="PH1PH2">
      <result-entry proc-utilization="3.77432e+000" squared-coeff-variation="1.00000e+000"
throughput="7.54864e+001" throughput-bound="1.00000e+004" utilization="3.77432e+000"/>
        <entry-phase-activities>
          <activity host-demand-mean="0.05" name="class5SwitchIAM_2_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="5.00000e-002" service-time-
variance="2.50000e-003" utilization="3.77432e+000"/>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="class5SwitchREL_2" type="PH1PH2">
      <result-entry proc-utilization="3.77432e+000" squared-coeff-variation="1.00000e+000"
throughput="7.54864e+001" throughput-bound="1.00000e+004" utilization="3.77432e+000"/>
        <entry-phase-activities>
          <activity host-demand-mean="0.05" name="class5SwitchREL_2_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="5.00000e-002" service-time-
variance="2.50000e-003" utilization="3.77432e+000"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="gwH248_1" scheduling="fcfs">
  <result-processor utilization="8.22824e-002"/>
    <task multiplicity="10" name="gwH248T_1" scheduling="fcfs">
    <result-task phase1-utilization="2.21592e+000" proc-utilization="3.01946e-003"
throughput="1.50973e+002" utilization="5.66783e+000"/>
      <entry name="gwH248AddE_1" type="PH1PH2">
```

<result-entry proc-utilization="1.50973e-003" squared-coeff-variation="4.12145e+000" throughput="7.54864e+001" throughput-bound="9.22509e+002" utilization="1.23599e+000"/>
<entry-phase-activities>
<activity host-demand-mean="2e-005" name="gwH248AddE_1_ph1" phase="1">
<result-activity proc-waiting="4.90860e-005" service-time="1.63737e-002" service-time-variance="1.10495e-003" utilization="1.23599e+000"/>
<synch-call calls-mean="1" dest="gwCacoAddE_1">
<result-call waiting="2.49483e-003"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
<entry name="gwH248SubtractE_1" type="PH1PH2">
<result-entry proc-utilization="1.50973e-003" squared-coeff-variation="4.59112e+000" throughput="7.54864e+001" throughput-bound="1.27226e+003" utilization="9.79927e-001"/>
<entry-phase-activities>
<activity host-demand-mean="2e-005" name="gwH248SubtractE_1_ph1" phase="1">
<result-activity proc-waiting="4.90860e-005" service-time="1.29815e-002" service-time-variance="7.73694e-004" utilization="9.79927e-001"/>
<synch-call calls-mean="1" dest="gwCacoSubtractE_1">
<result-call waiting="2.49483e-003"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
<entry name="gwH248ModifyE_1" type="PH1PH2">
<result-entry proc-utilization="0.00000e+000" squared-coeff-variation="0.00000e+000" throughput="0.00000e+000" throughput-bound="9.09091e+004" utilization="0.00000e+000"/>
<entry-phase-activities>
<activity host-demand-mean="1e-005" name="gwH248ModifyE_1_ph1" phase="1">
<result-activity proc-waiting="0.00000e+000" service-time="0.00000e+000" service-time-variance="1.00500e-008" utilization="0.00000e+000"/>
<synch-call calls-mean="1" dest="gwCacoModifyE_1">
<result-call waiting="0.00000e+000"/>
</synch-call>
</activity>
</entry-phase-activities>

</entry>
</task>
<task multiplicity="4" name="gwCacoT_1" scheduling="fcfs">
<result-task phase1-utilization="1.82861e+000" phase2-utilization="1.40554e-002" proc-utilization="7.92630e-002" throughput="1.50977e+002" utilization="4.73658e+000"/>
<entry name="gwCacoAddE_1" type="PH1PH2">
<result-entry proc-utilization="6.79397e-002" squared-coeff-variation="2.91298e+000" throughput="7.54885e+001" throughput-bound="3.66300e+002" utilization="1.05125e+000"/>
<entry-phase-activities>
<activity host-demand-mean="0.0008" name="gwCacoAddE_1_ph1" phase="1">
<result-activity proc-waiting="3.61928e-005" service-time="1.38079e-002" service-time-variance="5.64915e-004" utilization="1.04234e+000"/>
<synch-call calls-mean="1" dest="gwSvcSetupE_1">
<result-call waiting="4.75653e-007"/>
</synch-call>
</activity>
<activity host-demand-mean="0.0001" name="gwCacoAddE_1_ph2" phase="2">
<result-activity proc-waiting="1.80964e-005" service-time="1.18096e-004" service-time-variance="1.03275e-008" utilization="8.91492e-003"/>
</activity>
</entry-phase-activities>
</entry>
<entry name="gwCacoSubtractE_1" type="PH1PH2">
<result-entry proc-utilization="1.13233e-002" squared-coeff-variation="3.93340e+000" throughput="7.54885e+001" throughput-bound="5.06971e+002" utilization="7.91407e-001"/>
<entry-phase-activities>
<activity host-demand-mean="0.0001" name="gwCacoSubtractE_1_ph1" phase="1">
<result-activity proc-waiting="3.61928e-005" service-time="1.04157e-002" service-time-variance="4.32318e-004" utilization="7.86267e-001"/>
<synch-call calls-mean="1" dest="gwSvcReleaseE_1">
<result-call waiting="4.75653e-007"/>
</synch-call>
</activity>
<activity host-demand-mean="5e-005" name="gwCacoSubtractE_1_ph2" phase="2">
<result-activity proc-waiting="1.80964e-005" service-time="6.80964e-005" service-time-variance="2.82748e-009" utilization="5.14050e-003"/>
</activity>

```
        </entry-phase-activities>
      </entry>
      <entry name="gwCacoModifyE_1" type="PH1PH2">
      <result-entry proc-utilization="0.00000e+000" squared-coeff-variation="0.00000e+000"
throughput="0.00000e+000" throughput-bound="2.66667e+004" utilization="0.00000e+000"/>
        <entry-phase-activities>
          <activity host-demand-mean="0.0001" name="gwCacoModifyE_1_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="0.00000e+000" service-time-
variance="1.00000e-008" utilization="0.00000e+000"/>
          </activity>
          <activity host-demand-mean="5e-005" name="gwCacoModifyE_1_ph2" phase="2">
          <result-activity proc-waiting="0.00000e+000" service-time="0.00000e+000" service-time-
variance="2.50000e-009" utilization="0.00000e+000"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="gwH248_2" scheduling="fcfs">
  <result-processor utilization="9.05837e-002"/>
    <task multiplicity="10" name="gwH248T_2" scheduling="fcfs">
    <result-task phase1-utilization="1.11269e-001" proc-utilization="3.77432e-003"
throughput="2.26459e+002" utilization="2.75378e-001"/>
      <entry name="gwH248AddE_2" type="PH1PH2">
      <result-entry proc-utilization="1.50973e-003" squared-coeff-variation="2.46209e+000"
throughput="7.54864e+001" throughput-bound="1.21951e+004" utilization="7.25683e-002"/>
        <entry-phase-activities>
          <activity host-demand-mean="2e-005" name="gwH248AddE_2_ph1" phase="1">
          <result-activity proc-waiting="9.76420e-005" service-time="9.61342e-004" service-time-
variance="2.27541e-006" utilization="7.25683e-002"/>
            <synch-call calls-mean="1" dest="gwCacoAddE_2">
            <result-call waiting="7.74875e-006"/>
            </synch-call>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="gwH248ModifyE_2" type="PH1PH2">
```

<result-entry proc-utilization="7.54864e-004" squared-coeff-variation="1.49113e+000" throughput="7.54864e+001" throughput-bound="9.09091e+004" utilization="1.89729e-002"/>
<entry-phase-activities>
<activity host-demand-mean="1e-005" name="gwH248ModifyE_2_ph1" phase="1">
<result-activity proc-waiting="9.76420e-005" service-time="2.51342e-004" service-time-variance="9.41989e-008" utilization="1.89729e-002"/>
<synch-call calls-mean="1" dest="gwCacoModifyE_2">
<result-call waiting="7.74875e-006"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
<entry name="gwH248SubtractE_2" type="PH1PH2">
<result-entry proc-utilization="1.50973e-003" squared-coeff-variation="1.43996e+000" throughput="7.54864e+001" throughput-bound="8.33333e+004" utilization="1.97278e-002"/>
<entry-phase-activities>
<activity host-demand-mean="2e-005" name="gwH248SubtractE_2_ph1" phase="1">
<result-activity proc-waiting="9.76420e-005" service-time="2.61342e-004" service-time-variance="9.83493e-008" utilization="1.97278e-002"/>
<synch-call calls-mean="1" dest="gwCacoSubtractE_2">
<result-call waiting="7.74875e-006"/>
</synch-call>
</activity>
</entry-phase-activities>
</entry>
</task>
<task multiplicity="4" name="gwCacoT_2" scheduling="fcfs">
<result-task phase1-utilization="8.36280e-002" phase2-utilization="1.94645e-002" proc-utilization="8.68094e-002" throughput="2.26459e+002" utilization="2.59025e-001"/>
<entry name="gwCacoAddE_2" type="PH1PH2">
<result-entry proc-utilization="6.41634e-002" squared-coeff-variation="7.59008e-001" throughput="7.54864e+001" throughput-bound="4.70588e+003" utilization="6.95911e-002"/>
<entry-phase-activities>
<activity host-demand-mean="0.0008" name="gwCacoAddE_2_ph1" phase="1">
<result-activity proc-waiting="3.59515e-005" service-time="8.35952e-004" service-time-variance="6.41293e-007" utilization="6.31030e-002"/>
</activity>

```
        <activity host-demand-mean="5e-005" name="gwCacoAddE_2_ph2" phase="2">
        <result-activity proc-waiting="3.59515e-005" service-time="8.59515e-005" service-time-
variance="3.79251e-009" utilization="6.48817e-003"/>
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="gwCacoModifyE_2" type="PH1PH2">
    <result-entry proc-utilization="1.13230e-002" squared-coeff-variation="3.06351e-001"
throughput="7.54864e+001" throughput-bound="2.66667e+004" utilization="1.67507e-002"/>
      <entry-phase-activities>
        <activity host-demand-mean="0.0001" name="gwCacoModifyE_2_ph1" phase="1">
        <result-activity proc-waiting="3.59515e-005" service-time="1.35952e-004" service-time-
variance="1.12925e-008" utilization="1.02625e-002"/>
        </activity>
        <activity host-demand-mean="5e-005" name="gwCacoModifyE_2_ph2" phase="2">
        <result-activity proc-waiting="3.59515e-005" service-time="8.59515e-005" service-time-
variance="3.79251e-009" utilization="6.48817e-003"/>
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="gwCacoSubtractE_2" type="PH1PH2">
    <result-entry proc-utilization="1.13230e-002" squared-coeff-variation="3.06351e-001"
throughput="7.54864e+001" throughput-bound="2.66667e+004" utilization="1.67507e-002"/>
      <entry-phase-activities>
        <activity host-demand-mean="0.0001" name="gwCacoSubtractE_2_ph1" phase="1">
        <result-activity proc-waiting="3.59515e-005" service-time="1.35952e-004" service-time-
variance="1.12925e-008" utilization="1.02625e-002"/>
        </activity>
        <activity host-demand-mean="5e-005" name="gwCacoSubtractE_2_ph2" phase="2">
        <result-activity proc-waiting="3.59515e-005" service-time="8.59515e-005" service-time-
variance="3.79251e-009" utilization="6.48817e-003"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwSvc_1" scheduling="fcfs">
```

<result-processor utilization="1.05654e-001"/>
<task multiplicity="20" name="gwSvcT_1" scheduling="fcfs">
<result-task phase1-utilization="1.75463e+000" phase2-utilization="9.31395e-003" proc-utilization="1.05654e-001" throughput="1.50934e+002" utilization="4.51145e+000"/>
<entry name="gwSvcSetupE_1" type="PH1PH2">
<result-entry proc-utilization="6.41470e-002" squared-coeff-variation="1.21291e+000" throughput="7.54671e+001" throughput-bound="1.98610e+003" utilization="9.83559e-001"/>
<entry-phase-activities>
<activity host-demand-mean="0.0008" name="gwSvcSetupE_1_ph1" phase="1">
<result-activity proc-waiting="7.02521e-005" service-time="1.29712e-002" service-time-variance="2.06020e-004" utilization="9.78902e-001"/>
<synch-call calls-mean="1" dest="gwControlRouteRequestE_1">
<result-call waiting="3.55308e-004"/>
</synch-call>
<synch-call calls-mean="1" dest="coreSvcSetupE">
<result-call waiting="7.67693e-008"/>
</synch-call>
<synch-call calls-mean="1" dest="gwNlcPt2PtSetE_1">
<result-call waiting="1.05465e-004"/>
</synch-call>
<synch-call calls-mean="1" dest="gwIlcPt2PtSetE_1">
<result-call waiting="1.05465e-004"/>
</synch-call>
<synch-call calls-mean="1" dest="coreSvcConnectE">
<result-call waiting="7.67693e-008"/>
</synch-call>
</activity>
<activity host-demand-mean="5e-005" name="gwSvcSetupE_1_ph2" phase="2">
<result-activity proc-waiting="1.17087e-005" service-time="6.17087e-005" service-time-variance="2.63709e-009" utilization="4.65697e-003"/>
</activity>
</entry-phase-activities>
</entry>
<entry name="gwSvcReleaseE_1" type="PH1PH2">
<result-entry proc-utilization="4.15069e-002" squared-coeff-variation="2.04027e+000" throughput="7.54671e+001" throughput-bound="2.56739e+003" utilization="7.80386e-001"/>
<entry-phase-activities>

```
          <activity host-demand-mean="0.0005" name="gwSvcReleaseE_1_ph1" phase="1">
          <result-activity proc-waiting="3.51260e-005" service-time="1.02790e-002" service-time-
variance="2.18166e-004" utilization="7.75729e-001"/>
            <synch-call calls-mean="1" dest="gwControlDisconnectE_1">
            <result-call waiting="3.55308e-004"/>
            </synch-call>
            <synch-call calls-mean="1" dest="coreSvcReleaseE">
            <result-call waiting="7.67693e-008"/>
            </synch-call>
          </activity>
          <activity host-demand-mean="5e-005" name="gwSvcReleaseE_1_ph2" phase="2">
          <result-activity proc-waiting="1.17087e-005" service-time="6.17087e-005" service-time-
variance="2.63709e-009" utilization="4.65697e-003"/>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="gwSvcConnectE_1" type="PH1PH2">
      <result-entry proc-utilization="0.00000e+000" squared-coeff-variation="0.00000e+000"
throughput="0.00000e+000" throughput-bound="2.50000e+004" utilization="0.00000e+000"/>
        <entry-phase-activities>
          <activity host-demand-mean="0.00075" name="gwSvcConnectE_1_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="0.00000e+000" service-time-
variance="5.62500e-007" utilization="0.00000e+000"/>
          </activity>
          <activity host-demand-mean="5e-005" name="gwSvcConnectE_1_ph2" phase="2">
          <result-activity proc-waiting="0.00000e+000" service-time="0.00000e+000" service-time-
variance="2.50000e-009" utilization="0.00000e+000"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="coreSvc" scheduling="fcfs">
  <result-processor utilization="1.66028e-001"/>
    <task multiplicity="125" name="coreSvcT" scheduling="fcfs">
    <result-task phase1-utilization="1.25915e+000" phase2-utilization="1.83736e-002" proc-
utilization="1.66028e-001" throughput="2.26401e+002" utilization="2.13437e+000"/>
```

```
<entry name="coreSvcConnectE" type="PH1PH2">
<result-entry proc-utilization="6.03737e-002" squared-coeff-variation="1.31214e+000"
throughput="7.54671e+001" throughput-bound="8.06452e+004" utilization="1.28075e-001"/>
<entry-phase-activities>
<activity host-demand-mean="0.00075" name="coreSvcConnectE_ph1" phase="1">
<result-activity proc-waiting="6.23102e-005" service-time="1.61594e-003" service-time-
variance="3.77568e-006" utilization="1.21950e-001"/>
<synch-call calls-mean="1" dest="gwSvcConnectE_2">
<result-call waiting="5.11097e-007"/>
</synch-call>
</activity>
<activity host-demand-mean="5e-005" name="coreSvcConnectE_ph2" phase="2">
<result-activity proc-waiting="3.11551e-005" service-time="8.11551e-005" service-time-
variance="3.47064e-009" utilization="6.12454e-003"/>
</activity>
</entry-phase-activities>
</entry>
<entry name="coreSvcSetupE" type="PH1PH2">
<result-entry proc-utilization="6.41470e-002" squared-coeff-variation="9.18364e-001"
throughput="7.54671e+001" throughput-bound="2.18150e+004" utilization="6.00704e-001"/>
<entry-phase-activities>
<activity host-demand-mean="0.0008" name="coreSvcSetupE_ph1" phase="1">
<result-activity proc-waiting="1.55776e-004" service-time="7.87866e-003" service-time-
variance="5.81828e-005" utilization="5.94579e-001"/>
<synch-call calls-mean="1" dest="coreControlRouteRequestE">
<result-call waiting="6.13882e-004"/>
</synch-call>
<synch-call calls-mean="2" dest="coreNlcPt2PtSetE">
<result-call waiting="0.00000e+000"/>
</synch-call>
<synch-call calls-mean="1" dest="gwSvcSetupE_2">
<result-call waiting="5.11097e-007"/>
</synch-call>
</activity>
<activity host-demand-mean="5e-005" name="coreSvcSetupE_ph2" phase="2">
<result-activity proc-waiting="3.11551e-005" service-time="8.11551e-005" service-time-
variance="3.47064e-009" utilization="6.12454e-003"/>
```

```
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="coreSvcReleaseE" type="PH1PH2">
    <result-entry proc-utilization="4.15069e-002" squared-coeff-variation="1.65901e+000"
throughput="7.54671e+001" throughput-bound="2.39923e+004" utilization="5.48741e-001"/>
      <entry-phase-activities>
        <activity host-demand-mean="0.0005" name="coreSvcReleaseE_ph1" phase="1">
        <result-activity proc-waiting="9.34653e-005" service-time="7.19010e-003" service-time-
variance="8.77103e-005" utilization="5.42616e-001"/>
          <synch-call calls-mean="1" dest="coreControlDisconnectE">
          <result-call waiting="6.13882e-004"/>
          </synch-call>
          <synch-call calls-mean="1" dest="gwSvcReleaseE_2">
          <result-call waiting="5.11097e-007"/>
          </synch-call>
        </activity>
        <activity host-demand-mean="5e-005" name="coreSvcReleaseE_ph2" phase="2">
        <result-activity proc-waiting="3.11551e-005" service-time="8.11551e-005" service-time-
variance="3.47064e-009" utilization="6.12454e-003"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
 </processor>
 <processor name="gwControl_1" scheduling="fcfs">
 <result-processor utilization="9.05605e-003"/>
  <task name="gwControlT_1" scheduling="fcfs">
  <result-task phase1-utilization="1.68928e-001" proc-utilization="9.05605e-003"
throughput="1.50934e+002" utilization="1.71946e-001"/>
    <entry name="gwControlRouteRequestE_1" type="PH1PH2">
    <result-entry proc-utilization="3.01868e-003" squared-coeff-variation="1.00000e+000"
throughput="7.54671e+001" throughput-bound="2.50000e+004" utilization="3.01868e-003"/>
      <entry-phase-activities>
        <activity host-demand-mean="4e-005" name="gwControlRouteRequestE_1_ph1" phase="1">
        <result-activity proc-waiting="0.00000e+000" service-time="4.00000e-005" service-time-
variance="1.60000e-009" utilization="3.01868e-003"/>
```

```
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="gwControlDisconnectE_1" type="PH1PH2">
    <result-entry proc-utilization="6.03737e-003" squared-coeff-variation="1.39159e+000"
throughput="7.54671e+001" throughput-bound="4.80769e+002" utilization="1.65909e-001"/>
      <entry-phase-activities>
        <activity host-demand-mean="8e-005" name="gwControlDisconnectE_1_ph1" phase="1">
        <result-activity proc-waiting="0.00000e+000" service-time="2.19843e-003" service-time-
variance="6.72569e-006" utilization="1.65909e-001"/>
          <synch-call calls-mean="1" dest="gwNlcPt2PtSetE_1">
          <result-call waiting="5.92136e-005"/>
          </synch-call>
          <synch-call calls-mean="1" dest="gwIlcPt2PtSetE_1">
          <result-call waiting="5.92136e-005"/>
          </synch-call>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
</processor>
<processor name="gwSvc_2" scheduling="fcfs">
<result-processor utilization="1.66028e-001"/>
  <task multiplicity="20" name="gwSvcT_2" scheduling="fcfs">
  <result-task phase1-utilization="5.88948e-001" phase2-utilization="2.33466e-002" proc-
utilization="1.66028e-001" throughput="2.26401e+002" utilization="1.03485e+000"/>
    <entry name="gwSvcConnectE_2" type="PH1PH2">
    <result-entry proc-utilization="6.03737e-002" squared-coeff-variation="6.94829e-001"
throughput="7.54671e+001" throughput-bound="2.50000e+004" utilization="6.83913e-002"/>
      <entry-phase-activities>
        <activity host-demand-mean="0.00075" name="gwSvcConnectE_2_ph1" phase="1">
        <result-activity proc-waiting="5.31203e-005" service-time="8.03120e-004" service-time-
variance="5.65322e-007" utilization="6.06091e-002"/>
        </activity>
        <activity host-demand-mean="5e-005" name="gwSvcConnectE_2_ph2" phase="2">
        <result-activity proc-waiting="5.31203e-005" service-time="1.03120e-004" service-time-
variance="5.32176e-009" utilization="7.78219e-003"/>
```

```
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="gwSvcSetupE_2" type="PH1PH2">
    <result-entry proc-utilization="6.41470e-002" squared-coeff-variation="7.55690e-001"
throughput="7.54671e+001" throughput-bound="6.92042e+003" utilization="2.85771e-001"/>
      <entry-phase-activities>
        <activity host-demand-mean="0.0008" name="gwSvcSetupE_2_ph1" phase="1">
        <result-activity proc-waiting="2.12481e-004" service-time="3.68358e-003" service-time-
variance="1.08306e-005" utilization="2.77989e-001"/>
          <synch-call calls-mean="1" dest="gwControlRouteRequestE_2">
          <result-call waiting="3.92573e-004"/>
          </synch-call>
          <synch-call calls-mean="1" dest="gwNlcPt2PtSetE_2">
          <result-call waiting="3.27266e-008"/>
          </synch-call>
          <synch-call calls-mean="1" dest="gwIlcPt2PtSetE_2">
          <result-call waiting="5.21739e-013"/>
          </synch-call>
        </activity>
        <activity host-demand-mean="5e-005" name="gwSvcSetupE_2_ph2" phase="2">
        <result-activity proc-waiting="5.31203e-005" service-time="1.03120e-004" service-time-
variance="5.32176e-009" utilization="7.78219e-003"/>
        </activity>
      </entry-phase-activities>
    </entry>
    <entry name="gwSvcReleaseE_2" type="PH1PH2">
    <result-entry proc-utilization="4.15069e-002" squared-coeff-variation="2.20054e+000"
throughput="7.54671e+001" throughput-bound="7.60456e+003" utilization="2.58132e-001"/>
      <entry-phase-activities>
        <activity host-demand-mean="0.0005" name="gwSvcReleaseE_2_ph1" phase="1">
        <result-activity proc-waiting="1.06241e-004" service-time="3.31734e-003" service-time-
variance="2.57400e-005" utilization="2.50350e-001"/>
          <synch-call calls-mean="1" dest="gwControlDisconnectE_2">
          <result-call waiting="3.92573e-004"/>
          </synch-call>
        </activity>
```

```
          <activity host-demand-mean="5e-005" name="gwSvcReleaseE_2_ph2" phase="2">
          <result-activity proc-waiting="5.31203e-005" service-time="1.03120e-004" service-time-
variance="5.32176e-009" utilization="7.78219e-003"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="coreControl" scheduling="fcfs">
  <result-processor utilization="9.05605e-003"/>
    <task name="coreControlT" scheduling="fcfs">
    <result-task phase1-utilization="2.04131e-001" proc-utilization="9.05605e-003"
throughput="1.50934e+002" utilization="2.07150e-001"/>
      <entry name="coreControlRouteRequestE" type="PH1PH2">
      <result-entry proc-utilization="3.01868e-003" squared-coeff-variation="1.00000e+000"
throughput="7.54671e+001" throughput-bound="2.50000e+004" utilization="3.01868e-003"/>
        <entry-phase-activities>
          <activity host-demand-mean="4e-005" name="coreControlRouteRequestE_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="4.00000e-005" service-time-
variance="1.60000e-009" utilization="3.01868e-003"/>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="coreControlDisconnectE" type="PH1PH2">
      <result-entry proc-utilization="6.03737e-003" squared-coeff-variation="1.77614e+000"
throughput="7.54671e+001" throughput-bound="4.80769e+002" utilization="2.01113e-001"/>
        <entry-phase-activities>
          <activity host-demand-mean="8e-005" name="coreControlDisconnectE_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="2.66491e-003" service-time-
variance="1.26137e-005" utilization="2.01113e-001"/>
            <synch-call calls-mean="2" dest="coreNlcPt2PtSetE">
            <result-call waiting="0.00000e+000"/>
            </synch-call>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
```

```
  </processor>
  <processor name="gwIlc_1" scheduling="fcfs">
  <result-processor utilization="1.50934e-001"/>
    <task name="gwIlcT_1" scheduling="fcfs">
    <result-task phase1-utilization="1.50934e-001" proc-utilization="1.50934e-001"
throughput="1.50934e+002" utilization="1.50934e-001"/>
      <entry name="gwIlcPt2PtSetE_1" type="PH1PH2">
      <result-entry proc-utilization="1.50934e-001" squared-coeff-variation="1.00000e+000"
throughput="1.50934e+002" throughput-bound="1.00000e+003" utilization="1.50934e-001"/>
        <entry-phase-activities>
          <activity host-demand-mean="0.001" name="gwIlcPt2PtSetE_1_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="1.00000e-003" service-time-
variance="1.00000e-006" utilization="1.50934e-001"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="gwNlc_1" scheduling="fcfs">
  <result-processor utilization="1.50934e-001"/>
    <task name="gwNlcT_1" scheduling="fcfs">
    <result-task phase1-utilization="1.50934e-001" proc-utilization="1.50934e-001"
throughput="1.50934e+002" utilization="1.50934e-001"/>
      <entry name="gwNlcPt2PtSetE_1" type="PH1PH2">
      <result-entry proc-utilization="1.50934e-001" squared-coeff-variation="1.00000e+000"
throughput="1.50934e+002" throughput-bound="1.00000e+003" utilization="1.50934e-001"/>
        <entry-phase-activities>
          <activity host-demand-mean="0.001" name="gwNlcPt2PtSetE_1_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="1.00000e-003" service-time-
variance="1.00000e-006" utilization="1.50934e-001"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="gwControl_2" scheduling="fcfs">
  <result-processor utilization="9.05605e-003"/>
```

```
    <task name="gwControlT_2" scheduling="fcfs">
    <result-task phase1-utilization="1.77991e-001" proc-utilization="9.05605e-003"
throughput="1.50934e+002" utilization="1.81010e-001"/>
      <entry name="gwControlRouteRequestE_2" type="PH1PH2">
      <result-entry proc-utilization="3.01868e-003" squared-coeff-variation="1.00000e+000"
throughput="7.54671e+001" throughput-bound="2.50000e+004" utilization="3.01868e-003"/>
        <entry-phase-activities>
          <activity host-demand-mean="4e-005" name="gwControlRouteRequestE_2_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="4.00000e-005" service-time-
variance="1.60000e-009" utilization="3.01868e-003"/>
          </activity>
        </entry-phase-activities>
      </entry>
      <entry name="gwControlDisconnectE_2" type="PH1PH2">
      <result-entry proc-utilization="6.03737e-003" squared-coeff-variation="1.35507e+000"
throughput="7.54671e+001" throughput-bound="4.80769e+002" utilization="1.74972e-001"/>
        <entry-phase-activities>
          <activity host-demand-mean="8e-005" name="gwControlDisconnectE_2_ph1" phase="1">
          <result-activity proc-waiting="0.00000e+000" service-time="2.31852e-003" service-time-
variance="7.28423e-006" utilization="1.74972e-001"/>
            <synch-call calls-mean="1" dest="gwNlcPt2PtSetE_2">
            <result-call waiting="2.95406e-008"/>
            </synch-call>
            <synch-call calls-mean="1" dest="gwIlcPt2PtSetE_2">
            <result-call waiting="4.14181e-013"/>
            </synch-call>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="coreNlc" scheduling="fcfs">
  <result-processor utilization="3.01868e-001"/>
    <task multiplicity="12" name="coreNlcT" scheduling="fcfs">
    <result-task phase1-utilization="3.90151e-001" proc-utilization="3.01868e-001"
throughput="3.01868e+002" utilization="3.90151e-001"/>
      <entry name="coreNlcPt2PtSetE" type="PH1PH2">
```

<result-entry proc-utilization="3.01868e-001" squared-coeff-variation="6.49848e-001" throughput="3.01868e+002" throughput-bound="1.20000e+004" utilization="3.90151e-001"/>
<entry-phase-activities>
<activity host-demand-mean="0.001" name="coreNlcPt2PtSetE_ph1" phase="1">
<result-activity proc-waiting="2.92453e-004" service-time="1.29245e-003" service-time-variance="1.08553e-006" utilization="3.90151e-001"/>
</activity>
</entry-phase-activities>
</entry>
</task>
</processor>
<processor name="gwIlc_2" scheduling="fcfs">
<result-processor utilization="1.50934e-001"/>
<task multiplicity="6" name="gwIlcT_2" scheduling="fcfs">
<result-task phase1-utilization="1.70934e-001" proc-utilization="1.50934e-001" throughput="1.50934e+002" utilization="1.70934e-001"/>
<entry name="gwIlcPt2PtSetE_2" type="PH1PH2">
<result-entry proc-utilization="1.50934e-001" squared-coeff-variation="7.93371e-001" throughput="1.50934e+002" throughput-bound="6.00000e+003" utilization="1.70934e-001"/>
<entry-phase-activities>
<activity host-demand-mean="0.001" name="gwIlcPt2PtSetE_2_ph1" phase="1">
<result-activity proc-waiting="1.32509e-004" service-time="1.13251e-003" service-time-variance="1.01756e-006" utilization="1.70934e-001"/>
</activity>
</entry-phase-activities>
</entry>
</task>
</processor>
<processor name="gwNlc_2" scheduling="fcfs">
<result-processor utilization="1.50934e-001"/>
<task multiplicity="3" name="gwNlcT_2" scheduling="fcfs">
<result-task phase1-utilization="1.66931e-001" proc-utilization="1.50934e-001" throughput="1.50934e+002" utilization="1.66931e-001"/>
<entry name="gwNlcPt2PtSetE_2" type="PH1PH2">
<result-entry proc-utilization="1.50934e-001" squared-coeff-variation="8.26709e-001" throughput="1.50934e+002" throughput-bound="3.00000e+003" utilization="1.66931e-001"/>
<entry-phase-activities>

```
        <activity host-demand-mean="0.001" name="gwNlcPt2PtSetE_2_ph1" phase="1">
        <result-activity proc-waiting="1.05985e-004" service-time="1.10598e-003" service-time-
variance="1.01123e-006" utilization="1.66931e-001"/>
        </activity>
      </entry-phase-activities>
    </entry>
  </task>
 </processor>
        </lqn-model>
```