Extending Layered Queueing Network with Hybrid Sub-models Representing Exceptions and Decision Making

by

Pengfei Wu B. Eng., M.A. Sc.

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

May 2013

© Copyright 2013, Pengfei Wu
NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

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

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

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

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

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

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

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

Model elements involving decision-making present a challenge to performance analysis based on queueing formalisms, if the decisions are related to performance quantities, as in some kinds of exception decisions. In a queueing model the probability of the exception must be assumed in advance. In a state-based model the decision can be represented directly and its probability determined, however state-based models often suffer from state explosion issue which makes the model unsolvable.

This thesis creates the Hybrid Performance Modeling Methodology (HPMM). This methodology extends Layered Queueing Network (LQN) by combining it with Generalized Stochastic Petri Nets (GSPN), so that it can tackle systems with decision making and particularly with exceptions. This methodology is developed through systematic behaviour partitioning and modeling formalism selection, consistent sub-model construction with several approximation techniques, and an open tool development with the iteration solution. The tool has integrated the LQN solver and the GSPN solver so far. This methodology is further generalized as Aspect-Oriented Performance Modeling (AOPM) to provide more scalable performance analysis. HPMM is demonstrated through modeling a system with a range of resource allocation exception handling cases and systems with short and long run connection cases using the network communication protocol TCP.
Acknowledgements

First and foremost, I would like to express my special appreciation to my supervisor, Dr. Murray C. Woodside, for your valuable experience and advice throughout my Ph. D journey. Your seasoned guidance, broad vision, and numerous encouragements have been integral to this thesis work.

Deepest gratitudes are also due to my committee members Dr. Ahmed E. Hassan, Dr. Amiya Nayak, Dr. Dorina C. Petriu, Dr. Gregory R. Franks, Dr. Chung-Horng, Lung, Dr. Michel Barbeau, and Dr. Voicu Groza for the valuable comments and suggestions on my research.

Many thanks go to the friends from the labs in school and the office in the company, including Dorin B. Petriu, Ming Huo, Jim Li, Jin Xu, Tao Zhen, and Xiuping Wu, et al, providing countless hours of discussion. I would like to thank the members of the RADS Lab for creating a wonderful working environment and for their support. The financial assistance from Carleton University is appreciated.

Special thanks to my family. Words can not express how grateful I am. I have got the patience, understanding and love since I was born. I would like to thank to my beloved wife, Fang Yang, for your encourage and support, and to my beloved son, Austin Huayu Wu, for being a good boy cheering me up.
Table of Contents

ABSTRACT .........................................................................................................................III

ACKNOWLEDGEMENTS ............................................................................................... IV

TABLE OF CONTENTS ..........................................................................................V

LIST OF FIGURES ................................................................................................. IX

LIST OF TABLES ................................................................................................. XV

1. INTRODUCTION ......................................................................................................1

   1.1 MOTIVATION ................................................................................................. 1
   1.2 PROPOSED METHODOLOGY ........................................................................ 3
   1.3 SOLUTIONS FOR HYBRID MODELING ........................................................ 6
   1.4 CONTRIBUTIONS ............................................................................................ 8

2. BACKGROUND ....................................................................................................... 9

   2.1 EXCEPTION HANDLING MECHANISMS ......................................................... 9
       2.1.1 Exception Handling Definition .............................................................. 9
       2.1.2 Exception Handling Models ................................................................. 11
       2.1.3 Behaviour Analysis for Exception Handling ...................................... 13
   2.2 SCENARIO MODEL NOTATIONS ................................................................. 14
   2.3 PERFORMANCE MODELING FORMALISMS ................................................. 18
       2.3.1 Queueing Network Modeling ............................................................... 19
       2.3.2 State-Based Modeling ........................................................................ 21
       2.3.3 Comparison among various methodologies ...................................... 22
   2.4 SELECTED PERFORMANCE MODEL FORMALISM NOTATIONS ............... 25
   2.5 TRANSFORMATION FROM SCENARIO MODEL TO PERFORMANCE MODEL ... 29

3. CHALLENGES AND THE APPROACH TO A SOLUTION ..................31

   3.1 THE PROBLEM ............................................................................................... 31
3.2 The State of the Art ....................................................................................................................... 33
3.3 Weakness of the State of the Art .................................................................................................... 36
3.4 Hybrid Performance Modeling Methodology Overview ................................................................. 38
3.5 Hybrid Performance Modeling Procedure ...................................................................................... 42

4. HYBRID PERFORMANCE MODELING METHODOLOGY ....44
4.1 Scope of Hybrid Performance Modeling Methodology ................................................................. 44
4.2 Behaviour Partition with Model Formalism Selection ................................................................. 47
4.3 Sub-Model Construction .................................................................................................................. 51
   4.3.1 Behaviour Sub-Scenario Construction from Behaviour Cells ................................................. 51
   4.3.2 Behaviour Sub-Scenario Construction of an Example System .............................................. 52
   4.3.3 Performance Sub-Model Construction from Behaviour Sub-Scenario ................................. 55
   4.3.4 Interaction among Performance Sub-Models ............................................................................ 56
4.4 Consistent Construction of a Hybrid Performance Model ............................................................ 57
   4.4.1 Consistency among Sub-models ............................................................................................... 58
   4.4.2 Structural Consistency ............................................................................................................. 59
   4.4.3 Behaviour Consistency ............................................................................................................ 60
4.5 Consistent Parameters for Abstract Scenario Fragments .............................................................. 61
   4.5.1 Surrogate Delays ...................................................................................................................... 62
   4.5.2 Probabilities and Average Visit Counts for the Specific Operations (Surrogate flows)........... 64
   4.5.3 Surrogate Customers ............................................................................................................... 67
   4.5.4 Parameters Notations in Data Exchange ............................................................................... 70
4.6 Solution Consistency ..................................................................................................................... 71
4.7 Hybrid Performance Model Solution and Tools .......................................................................... 74
   4.7.1 Hybrid Performance Modeling Framework ............................................................................. 74
   4.7.2 Hybrid Performance Model Solver: Combining SPNP with LQNS ......................................... 76
   4.7.3 Hybrid Performance Model Solution: an Open Hybrid Performance Modeling Tool .......... 77

5. PERFORMANCE MODELING IN RESOURCE ALLOCATION

EXCEPTION HANDLING....................................................................................................................... 83
5.1 Fixed-Probability Exceptions with Probability Propagation .......................................................... 83
   5.1.1 The Scenario Model of an Example ......................................................................................... 84
5.1.2 Performance Model Construction for this Example ............................................................ 87
5.1.3 An Approach to Parameterizing the LQN and GSPN Models of Exceptions .................. 90
5.1.4 Limitation of the Current Approach ............................................................................... 94

5.2 DIFFERENT EXCEPTION MODELS IN RESOURCE ALLOCATION EXCEPTION HANDLING .................................................................................................................. 94

5.2.1 Use Case Map Extensions ............................................................................................... 95
5.2.2 Resource Allocation Exception Handling ........................................................................ 97
5.2.3 Why is the Hybrid Modeling Solution needed ............................................................... 99

5.3 HYBRID PERFORMANCE MODELING FOR RESOURCE ALLOCATION EXCEPTION HANDLING ...................................................................................................................... 100

5.3.1 Scenario Description ....................................................................................................... 100
5.3.2 Queue Pattern: Deriving the LQN Model ...................................................................... 102
5.3.3 Queue Pattern: Hybrid Model Solution ......................................................................... 108
5.3.4 Abort Pattern: Hybrid Model Construction .................................................................... 111
5.3.5 Infinite Retry Pattern: Hybrid Model Construction ....................................................... 118
5.3.6 Finite Retry Pattern: Hybrid Model Construction ........................................................ 125

5.4 HYBRID PERFORMANCE MODEL RESULTS AND ANALYSIS ......................................... 133

5.4.1 First Case: Disk Bottleneck ............................................................................................ 134
5.4.1.1 First Case: Abort Pattern, Hybrid Model Results and Analysis ................................. 134
5.4.1.2 First Case: Infinite Retry Pattern, Hybrid Model Results and Analysis .................. 139
5.4.1.3 First Case: Finite Retry Pattern, Hybrid Model Results and Analysis .................... 143
5.4.2 Second Case: Lock Bottleneck ....................................................................................... 147
5.4.2.1 Second Case: Abort Pattern, Hybrid Model Results and Analysis ......................... 147
5.4.2.2 Second Case: Infinite Retry Pattern, Hybrid Model Results and Analysis ............. 151
5.4.2.3 Second Case: Finite Retry Pattern, Hybrid Model Results and Analysis ............... 156
5.4.2.4 Second Case: Result Discussion .............................................................................. 160

6. ASPECT-ORIENTED PERFORMANCE MODELING .......... 163

6.1 GENERALIZATIONS .............................................................................................................. 163
6.2 EXAMPLE OF COMPOSITION AND ASPECT IDENTIFICATION .................................. 165
6.3 DIFFERENCES IN SUB-MODEL CONSTRUCTION IN AOPM ......................................... 170
List of Figures

Figure 1-1 The Performance Model Building Transformation ........................................... 3
Figure 1-2 The Performance Model Solution ..................................................................... 4
Figure 1-3 Hybrid Performance Modeling Approach ....................................................... 5
Figure 2-1 Exception Handling Mechanism and Scenario Model ..................................... 10
Figure 2-2 A Possible Scenario of the Exception Mechanism ......................................... 13
Figure 2-3 Use Case Map Notation and a Simple Scenario .............................................. 16
Figure 2-4 A UCM with Fork and Join .............................................................................. 18
Figure 2-5 A Use Case Map with Two Scenarios and LQN Sub-Models ......................... 26
Figure 2-6 A GSPN Model of the Application ................................................................... 28
Figure 3-1 Hybrid Performance Modeling Idea on Extending LQN by GSPN ................. 32
Figure 4-1 The Scenario of a Use Case with Decision Making ........................................ 45
Figure 4-2 Behaviour Partitions and Behaviour Cell Composition .................................. 54
Figure 4-3 Partitioning and Approximation of a Behaviour Fragment for a Sub-Model .... 56
Figure 4-4 Surrogate Delay Definition ............................................................................. 62
Figure 4-5 Probability Definition .................................................................................... 64
Figure 4-6 Average Visit Counts Definition .................................................................... 66
Figure 4-7 Surrogate Customer Definition ..................................................................... 67
Figure 4-8 Hybrid Performance Modeling Framework with the Fixed-Point Iteration ........ 74
Figure 4-9 A Hybrid Performance Tool Combining LQN and GSPN .................................. 77
Figure 4-10 The Class Diagram of an Open Hybrid Performance Modeling Tool JAVA Implementation .................................................................................................................. 82
Figure 5-1 An Example of the Exception Handling ......................................................... 85
FIGURE 5-2 THE SCENARIO DESCRIPTION FOR THE EXAMPLE OF FIGURE 5-1 .................. 86
FIGURE 5-3 EXPANDED UCM WITH PROBABILITIES AND LABELED FLOW GRAPH OF THE
EXAMPLE ........................................................................................................................................... 87
FIGURE 5-4 A GSPN MODEL FOR THE EXAMPLE WITH PROPAGATION PROBABILITY ...... 88
FIGURE 5-5 A LQN MODEL FOR THE EXAMPLE WITH PROPAGATION PROBABILITY ...... 89
FIGURE 5-6 A CODE BLOCK AND THE ASSOCIATED FLOW GRAPH ................................. 90
FIGURE 5-7 THE EXTENDED FLOW GRAPH ............................................................................... 91
FIGURE 5-8 THE EXTENSION OF A USE CASE MAP ............................................................... 95
FIGURE 5-9 THE SCENARIO MODEL FOR QUEUE PATTERN IN EXTENDED UCM ........... 97
FIGURE 5-10 THE SCENARIO MODEL FOR ABORT PATTERN IN EXTENDED UCM ........... 98
FIGURE 5-11 THE SCENARIO MODEL FOR INFINITE RETRY PATTERN IN EXTENDED UCM 98
FIGURE 5-12 THE SCENARIO MODEL FOR FINITE RETRY PATTERN IN EXTENDED UCM 99
FIGURE 5-13 THE SCENARIO MODEL OF A WEB APPLICATION WITH THE QUEUE PATTERN 102
FIGURE 5-14 BEHAVIOUR CELLS OF THE WEB APPLICATION WITH THE QUEUE PATTERN 104
FIGURE 5-15 THE LQN MODEL OF THE QUEUE PATTERN ..................................................... 107
FIGURE 5-16 BEHAVIOUR AGGREGATION FOR THE GSPN MODEL OF THE QUEUE
PATTERN ............................................................................................................................................ 108
FIGURE 5-17 THE GSPN SCENARIO MODEL OF THE QUEUE PATTERN ............................... 109
FIGURE 5-18 THE GSPN SUB-MODEL OF THE QUEUE PATTERN ......................................... 110
FIGURE 5-19 THE SCENARIO MODEL OF THE WEB APPLICATION WITH THE ABORT PATTERN 111
FIGURE 5-20 BEHAVIOUR CELLS OF THE WEB APPLICATION WITH THE ABORT PATTERN 112
FIGURE 5-21 BEHAVIOUR PARTITION OF THE WEB APPLICATION WITH THE ABORT
PATTERN ............................................................................................................................................. 113
FIGURE 5-22 THE LQN SCENARIO MODEL OF ABORT PATTERN .......................................... 114
FIGURE 5-23 BEHAVIOUR AGGREGATION AND THE GSPN SCENARIO MODEL OF ABORT
PATTERN ............................................................................................................................................. 114
FIGURE 5-24 THE LQN SUB-MODEL OF ABORT PATTERN ..................................................... 115

x
Figure 5-25 The GSPN Sub-Model of Abort Pattern

Figure 5-26 The Scenario Model of the Web Application with the Infinite Retry Pattern

Figure 5-27 Behaviour Cells of the Web Application with the Infinite Retry Pattern

Figure 5-28 Behaviour Partition of the Web Application with the Infinite Retry Pattern

Figure 5-29 The LQN Scenario Model of the Infinite Retry Pattern

Figure 5-30 Behaviour Aggregations and GSPN Scenario Model of the Infinite Retry Pattern

Figure 5-31 The LQN Sub-Model of the Infinite Retry Pattern

Figure 5-32: The GSPN Sub-Model of the Infinite Retry Pattern

Figure 5-33 The Scenario Model of the Web Application with the Finite Retry Pattern

Figure 5-34 Behaviour Cells of the Web Application with the Finite Retry Pattern Lock

Figure 5-35 Behaviour Partition of the Web Application with Finite Retry Pattern

Figure 5-36 The LQN Scenario Model of the Finite Retry Pattern

Figure 5-37 Behaviour Aggregations and the GSPN Scenario Model of the Finite Retry Pattern

Figure 5-38 The LQN Sub-Model of the Finite Retry Pattern

Figure 5-39 The GSPN Sub-Model of the Finite Retry Pattern (Three Attempts)

Figure 5-40 Lock Acquisition Probability for the Abort Pattern (First Case)

Figure 5-41 Mean Lock Holding Time for the Abort Pattern (First Case)

Figure 5-42 Utilization of the Bottleneck (The Disk) for the Abort Pattern (First Case)

Figure 5-43 Response Time of the Web Server for the Abort Pattern (First Case)
Figure 5-63 Utilization of the Bottleneck (The Lock) for the Infinite Retry Pattern (Second Case) ................................................................. 154
Figure 5-64 Response Time of the Web Server for the Infinite Retry Pattern (Second Case) ............................................................................. 155
Figure 5-65 Useful Throughput for the Infinite Retry Pattern (Second Case) .............................................................................................. 156
Figure 5-66 Lock Acquisition Probability for the Finite Retry Pattern (Second Case) .................................................................................... 157
Figure 5-67 Mean Number of Retries for the Finite Retry Pattern (Second Case) ....................................................................................... 158
Figure 5-68 Mean Lock Holding Time for the Finite Retry Pattern (Second Case) ....................................................................................... 158
Figure 5-69 Utilization of the Bottleneck (The Lock) for the Finite Retry Pattern (Second Case) ................................................................. 159
Figure 5-70 Response Time of the Web Server for the Finite Retry Pattern (Second Case) ............................................................................. 159
Figure 5-71 Useful Throughput for the Finite Retry Pattern (Second Case) .............................................................................................. 160
Figure 6-1 The AoUCM Model of HBS .................................................................................................................................................. 167
Figure 6-2 The Composed Map of HBS .................................................................................................................................................. 168
Figure 6-3 The Behaviour Cells of HBS ............................................................................................................................................. 169
Figure 6-4 The Behaviour Partitions of HBS ........................................................................................................................................ 172
Figure 6-5 The sub-scenario $S_1$ of HBS in AOPM .................................................................................................................................. 173
Figure 6-6 The sub-scenario $S_2$ of HBS in AOPM .................................................................................................................................. 174
Figure 6-7 The sub-scenario $S_3$ of HBS in AOPM .................................................................................................................................. 175
Figure 7-1 Congestion Window Size of TCP Reno .................................................................................................................................. 180
Figure 7-2 The Network Topology ...................................................................................................................................................... 181
Figure 7-3 The System Model in the Dumbbell Topology ..................................................................................................................... 183
Figure 7-4 The TCP Reno Behaviour of a TCP Connection ................................................................................................................ 184
Figure 7-5 The Use Case Map of TCP Reno Segment Behaviour ........................................................................................................ 192
FIGURE 7-6 BEHAVIOUR PARTITION OF SEGMENT BEHAVIOUR ......................................................... 195
FIGURE 7-7 THE LQN SCENARIO MODEL OF TCP RENO .......................................................... 196
FIGURE 7-8 THE GSPN SCENARIO MODEL OF TCP RENO ....................................................... 196
FIGURE 7-9 EVOLUTION OF WINDOW SIZE OVER TIME WITH TD LOSS INDICATION......... 198
FIGURE 7-10 HIGH-LEVEL VIEW OF THE TCP HYBRID MODEL FOR THE LONG RUN
CONNECTION....................................................................................................................................... 199
FIGURE 7-11 THE GSPN SUB-MODEL FOR THE LONG RUN CONNECTION ......................... 200
FIGURE 7-12 THE LQN SUB-MODEL OF THE LONG RUN CONNECTION WITH A SPECIFIC
WINDOW SIZE........................................................................................................................................ 203
FIGURE 7-13 THE LQN SUB-MODEL FOR THE LONG RUN CONNECTION WITH THE
STRADDLED WINDOW SIZES ............................................................................................................... 205
FIGURE 7-14 HIGH-LEVEL VIEW OF THE TCP HYBRID MODEL FOR THE SHORT RUN
CONNECTION........................................................................................................................................ 210
FIGURE 7-15 THE GSPN SUB-MODEL OF THE SHORT RUN CONNECTION ......................... 210
FIGURE 7-16 THE LQN SUB-MODEL FOR THE SHORT RUN CONNECTION ........................... 213
List of Tables

TABLE 4-1 BEHAVIOUR CELLS WITH SCORES FOR SELECTED MODELING FORMALISMS......49
TABLE 4-2 SCORES AND SELECTED FORMALISMS FOR BEHAVIOUR CELLS OF THE EXAMPLE SYSTEM..........................................................................................................................50
TABLE 4-3 PARAMETER NOTATIONS IN DATA EXCHANGE..............................................................................................................................71
TABLE 4-4 KEYS, ATTRIBUTES, AND SAMPLE EXPRESSIONS IN LQN RESULT EXPRESSIONS 80
TABLE 5-1 SCORES OF BASIC BEHAVIOUR CELLS OF THE WEB APPLICATION ...............105
TABLE 5-2 SCORES OF QUEUE PATTERN BEHAVIOUR CELLS IN THE WEB APPLICATION....105
TABLE 5-3 DATA EXCHANGE BETWEEN THE LQN MODEL AND THE GSPN MODEL OF THE Queue Pattern.................................................................................................................................111
TABLE 5-4 SCORES OF ABORT PATTERN BEHAVIOUR CELLS IN THE WEB APPLICATION ....112
TABLE 5-5 DATA EXCHANGE BETWEEN THE LQN MODEL AND THE GSPN MODEL OF THE Abort Pattern ...........................................................................................................................................117
TABLE 5-6 SCORES OF THE INFINITE RETRY PATTERN BEHAVIOUR CELLS IN THE WEB Application ...........................................................................................................................................119
TABLE 5-7 DATA EXCHANGE BETWEEN THE LQN MODEL AND THE GSPN MODEL OF THE Infinite Retry Pattern...........................................................................................................................................124
TABLE 5-8 SCORES OF THE FINITE RETRY PATTERN BEHAVIOUR CELLS IN THE WEB Application ...........................................................................................................................................126
TABLE 5-9 DATA EXCHANGE BETWEEN THE LQN MODEL AND THE GSPN MODEL OF THE Finite Retry Pattern...........................................................................................................................................132
TABLE 6-1 SCORES OF BEHAVIOUR CELLS FOR THE HBS.....................................................170
TABLE 7-1 SCORES OF BEHAVIOUR CELLS OF TCP RENO SEGMENTS BEHAVIOUR.........194
TABLE 7-2 DATA EXCHANGE BETWEEN THE GSPN SUB-MODEL AND THE LQN SUB-MODEL OF THE TCP RENO LONG RUN CONNECTION...............................................................................................206
Table 7-3 Comparison of Percentages of Successful Transmissions between Hybrid Model Solution and NS-2 Simulation in the Long Run Connection......208
Table 7-4 Data Exchange between the GSPN Sub-model and the LQN Sub-model for the TCP Reno Short Run Connection ..............................................................................................................215
Table 7-5 Comparison of Percentages of Successful Transmissions and Delay between Hybrid Model Solution and NS-2 Simulation in the Short Run Connection........................................................................................................................................218
1. Introduction

This chapter provides a high level overview on the challenges of performance modeling in the software industry, especially for a system with exception handling. It also briefly introduces the hybrid performance modeling idea and summarizes the contributions of the thesis.

1.1 Motivation

Performance is a key criterion in software development and deployment because poor performance may cause customer dissatisfaction and even project failure. Performance analysis and prediction is an essential step in solving performance problems and providing performance evaluation in the life cycle of a software system. Comparing three techniques, analytical modeling, simulation, and measurement [Jain91] in performance analysis and prediction, the analytical modeling approach was found to be best, considering accuracy, cost, and scalability.

Among analytical modeling formalism, Layered Queueing Networks (LQN) [Franks00][Franks08][Rolia92][Rolia95][Woodside89][Woodside86] is deemed to be a powerful tool, which can tackle the performance of a layered system with the simultaneous resource possession. LQN can handle different message semantics like synchronous, asynchronous, and forwarding. LQN is able to provide a scalable solution due to efficient solving algorithm. However when LQN is applied to model decision making, the probability of the decision must be assumed in advance. This assumption may lead the inaccuracy in the performance result. This research is focused on extending LQN by taking advantage of a second formalism, Generalized Stochastic Petri Nets or GSPN [Marsan84][Marsan95], which is able to model decision making. Some exception handling includes the decision making based on the performance of the system. Thus this research starts with modeling exception handling.
Exception handling is common in software. Although software permeates every aspect of modern life, it is extremely difficult to produce flawless software. Fault tolerance techniques including exception handling and recovery mechanisms are used to compensate for some faults in a reliable software system. Most modern programming languages support exception definition and exception handling. With exception-handling mechanisms, more complicated behaviors like decision making are introduced in system execution scenarios. Depending on the severity of exception conditions, an exception handler can take some action such as log a trace, trigger an appropriate audit, trigger a task rollback, or trigger a reboot process. The handler actions include abort, retry, and resume. They affect the normal execution paths.

Exception handling mechanisms in a software system present a challenge in model-based performance analysis. The exception handling mechanisms may be triggered by the performance of the computer system. Different execution paths in exception handling mechanisms can be chosen based upon the logical decision from the performance. In analytical performance modeling, exception handling mechanisms can be described by the probability of occurrences. For some kinds of exceptions however the probability depends on the state behaviour over time, which is part of the model. If a completely state based model is used, the solution may be impractical due to the state explosion issue. Although a state-based model formalism is a good candidate for modeling exception handling and decision making, the corresponding state space may not be handled well in modeling a large and complicate system. The time for solving a model could be extremely long. A queue-based model could be an alternative for an efficient solution. Since a queue-based model is not able to model the states, it would model exception handling mechanisms with assumptions. The assumptions on the probabilities may affect the accuracy of performance analysis. The motivation of this research is to find a way to combine queueing-based model formalism for the contention behaviour and state-based model formalism to describe the occurrence of the exceptions. This research can provide a scalable performance solution of a system with the complicated behaviour. Therefore the hybrid performance modeling methodology proposed in this research can relieve the inadequacy of a specific model.
mechanism as well as to improve the scalability of a single-formalism model solution. It can extend the capability of any individual formalism and increase its power.

![Diagram of performance model building transformation](image)

**Figure 1-1: The Performance Model Building Transformation**

### 1.2 Proposed Methodology

This research includes performance model construction and solution. To create a performance model, a formalism is chosen according to the modeler's knowledge and experience. The structure of the performance model is derived from the system knowledge. The performance model can be parameterized by using the system knowledge and the measured data. To obtain the model structure, the behaviour scenario including operations, sequences, and the usage of resources is examined so that the elements in a performance model, the resource usage, and the interaction among elements can be derived from the scenario. For instance, Figure 1-1 shows a scenario in which users browse web pages located on the web server. When the web pages are loaded, they retrieve data from a database server and get results back to the browsers. Certainly the performance model structure can be automatically developed through a model-building transformation [Petrin02b]. Deriving from the performance model construction, the subset of parameters of a performance model may vary. The model solution [Menasce94] can be regarded as a function evaluation shown in Figure 1-2. The performance measures are function values derived through the solution tool. The input variables as parameters are the arguments of the function. In this way the
performance model parameterizing and the performance model solution are naturally integrated in the hybrid performance model solution.

![Diagram of performance model solution]

**Figure 1-2 The Performance Model Solution**

Hybrid performance modeling integrates two or more modeling formalisms to provide a hybrid performance model solution for a computer or communication application. In this hybrid performance model, multiple models are developed from the same scenario of the application using different formalisms. In this research, only two performance modeling formalisms are employed to demonstrate the hybrid performance modeling idea. The idea of hybrid performance modeling can be illustrated through a generic application in Figure 1-3. Considering the strengths of different performance modeling formalisms for modeling the behaviour, the boundary separating the different behaviour can be derived through behaviour analysis. Meanwhile, different modeling formalisms may be chosen to model the different parts of the behaviour in the same scenario, which leads a hybrid modeling solution. Thus each model is developed on some behaviour and abstracts other parts of the behaviour. The abstracted part of the behaviour turns to be the interface of this model. For instance, to model an exception handing application, one model is focused on the exception decision and the factors that control the probability of exception occurrence. Another model is focused on the rest of behaviour, using the exception probability to abstract the exception handling mechanism. To develop a performance model from the scenario with the abstracted part, the abstraction of the behaviour has to be parameterized and these
parameters can be calculated through the solution of the other model. Therefore an iterative approach is formed through solving one model in a formalism and computing the parameters of the next model. The iterative process is the major part in the solution of the hybrid performance modeling. In this iterative solution of a hybrid model, each model solution can be regarded as a function to compute a solution with the given parameters. For instance, the behaviour of an application is partitioned into Behaviour1 and Behaviour2 shown in Figure 1-3 (A). Behaviour1 and the abstraction of Behaviour2 as one sub-scenario shown in Figure 1-3(B) are modeled as Model1 in performance modeling formalism Formalism1. Behaviour2 and the abstraction of Behaviour1 as the other sub-scenario are modeled as Model2 in performance modeling formalism Formalism2. The parameters of the abstraction of Behaviour2 in Model1 have to be found and can be computed from the solution of Model2. Also the parameters of the abstraction of Behaviour1 in Model2 have to be found and can be computed from the solution of Model1. Figure 1-3 (B) shows the interface of each model and the data exchange between Model1 and Model2.

(A) Behaviour of an Application     (B) a Sub-Scenario     (C) a Hybrid Performance Model

FIGURE 1-3 HYBRID PERFORMANCE MODELING APPROACH

The idea of hybrid performance modeling can be further extended as aspect-oriented performance modeling. Like the exception handling aspect in aspect-oriented programming [Lippert00] [Filho06], the aspect model may occur in multiple places and can be woven to

the base model to derive the composed scenario. When the hybrid performance modeling behaviour analysis and partition are applied to the composed scenario with the aspect having the decision, different sub-scenarios can be identified for the aspect. If multiple performance modeling formalisms are needed for the composed scenario, a performance sub-model can be constructed for each place of the aspect. Following the same steps of hybrid performance model construction, multiple sub-models are developed in aspect-oriented performance modeling. In this way, aspect-oriented performance modeling enhances the automation of performance model construction and improving the modularity, reusability, and scalability of performance models in hybrid performance modeling.

1.3 Solutions for Hybrid Modeling

A systematic methodology has been developed and used in the hybrid performance modeling solution. This methodology is termed as Hybrid Performance Modeling Methodology (HPMM). With multiple formalisms, HPMM can be applied for modeling a system with decision making. The decision making could drive the execution of the scenario along different execution paths. This decision could base on the problem data like predicates. The predicates are counts of loops or the flag sets. The decision could also base on dynamic performance data. Performance data are like the delay along an execution path or the queue size at a point in the execution path. The latter has to be solved through an iterative model. Multiple formalisms can be deployed for more efficient solution through an iterative model. This section describes the problems that must be resolved to make hybrid modeling systematic and useful, and the approaches taken here to solve them.

In constructing a model using HPMM, it is required to partition the system description and scenario into parts correctly, construct the sub-models consistently, and develop the relationship between the sub-models. The approach taken here (in section 4.2) is to decompose the system scenario according to the modeling formalism. As the system description for a given scenario is decomposed into the behaviour fragments, the most suitable modeling formalism is chosen to focus on the behavior fragment considered the
scalability of the modeling solution. From a behaviour fragment, a performance sub-model in a chosen formalism can be developed from the focused behavior fragment and the abstraction of the remainder behaviour (in section 4.3). The connection between a behaviour fragment and the abstraction of the behavior fragment can be represented between sub-models.

Due to modeling a single overall behavior description, consistency is obtained from the beginning of behaviour decomposition, then constructing sub-models, and developing the relationship between the sub-models (in section 4.4). To derive the accurate parameters of the sub-model, the coordinated definition is required in constructing sub-models. The parameters of a sub-model are updated from the solution of other sub-models. The coordinated definition is derived from the connection between behaviour fragments and provides the accurate parameters for the interface of the sub-model. Thus the relationship between sub-models as the data exchange is established in constructing sub-model.

The hybrid performance model solver (in section 4.5) in HPMM is a tool to solve hybrid performance models. This tool requires a hybrid solution controller that invokes the solvers of different formalisms. The controller parses the solution of the sub-models and updates the parameters of the sub-models. The controller iterates to solve each sub-model and implements the data changes between sub-models. Section 4.5 describes a general architecture of the hybrid performance solution. Two tools are developed to connect LQN solver to other tools like SPNP.

HPMM is applied for modeling exception handling (in chapter 5). The taxonomy of possible sub-behaviours in exception handling behaviour models is investigated. Focusing on the resource allocation exception handling, the behaviour of a system with the resource allocation exception is decomposed. The corresponding sub-models of different exception handling patterns are developed through coordinated definition. The validation has been conducted for all hybrid models of different exception handling models.
Inspired by exception handling aspects in aspect-oriented programming [Lippert00] [Filho06], HPMM can be generalized as aspect-oriented performance modeling, AOPM (in chapter 6). The aspect in AOPM represents the decision making as a cross cutting concern in a system. Rather than using a single sub-model for the decision behaviour, the aspect behaviour can be modeled in multiple sub-models, one for each aspect insertion. The sub-model construction in AOPM follows the same approach of HPMM.

Hybrid performance modeling approaches can be also applied in modeling an important aspect, namely communication protocol behaviour (in chapter 7). The hybrid performance modeling solution does not only cover the short run TCP connection, but also include the long run TCP connection.

1.4 Contributions

The contributions of this research are summarized as follows:

1. The hybrid performance modeling methodology (HPMM) based on systematic behaviour partitioning and modeling formalism selection, to provide consistent sub-model construction (in section 3.4 and chapter 4).

2. A general approach to applying this method to model exception handling (in chapter 5).

3. Introduction and definition of the concept of aspect-oriented performance modeling, as a generalization of the hybrid modeling methodology (in chapter 6).


5. An open solution tool combining LQNS and SPNP, which integrates the LQN and GSPN modeling formalisms (in section 4.7).

6. Evaluation of HPMM using this tool on a range of exception handling cases (in section 5.4) and the short and long run connection cases (in section 7.5 and section 7.6).
2. Background

This chapter describes the background in modeling system behaviour and performance in exception handling, including the concepts of exceptions and some exception handling models. Scenario models for behaviour or related to performance model formalisms with their strengths and weaknesses are described. The notations of selected performance model formalisms are covered, which are used in the thesis. Finally the transformation from a scenario model to a performance model is discussed.

2.1 Exception Handling Mechanisms

A computer system has to cope with different exceptional conditions so that its reliability can be enhanced. Exception handling has an important place in software design and development, and in software behaviour.

2.1.1 Exception Handling Definition

The concept of exception handling has been refined in the last decade. In [Goodenough75a][Goodenough75b] Goodenough provides the definition: exception conditions are those brought to the attention of the operation's invoker and the invoker is then permitted (or required) to respond to the condition. He describes general exception-handling mechanisms in those papers. His work forms the foundation of exception handling terminology. Shaula and Daniel [Shaula85] define the exception concept as: those states not satisfying the normal case input assertion of the operation are called exceptions of that operation. Knudsen [Knudsen87] defines an exception: an exception occurrence is a computational state that requires an extraordinary computation. An exception is the computational state where something unusual has happened. The exception handler is the specification of the extraordinary computation to handle the exceptions.
Exception-handling mechanisms have been integrated into modern high-level programming languages such as C++, Java, Ada, Eiffel and Smalltalk. In these languages, an exception is viewed as an error or an event that occurs unexpectedly or infrequently. An exception occurrence can be defined as throwing an instance of an exception class in object-oriented programming languages or a system runtime error such as division by zero, or a null pointer reference. The location of an exception occurrence is defined as an escape point.

The popular object-oriented programming languages C++ and Java, are selected as examples to discuss the concept of exception handling mechanism. Both of them use a try-catch clause. To describe the behaviour of a try-catch clause, a rectangle is used to represent a code block. A graphical view of a try-catch clause and a scenario model for its behaviour are shown in Figure 2-1 (the scenario notation is described later in this chapter).

An exception handler (or a handler) is a lexical region of code that is invoked when an exception occurs. The handler's block of code is separated from the normal control path in the program. The normal control path is a regular computation process without any exceptions. An exception is handled when the handler's execution has completed. Once an exception occurs, the control flow of a program for handling the exception is determined through a specific exception-handling model [Shaula85].
Different programming languages have different rules to bind an exception occurrence to a specific exception handler. An *exception context* is the information available to the exception handler concerning the exception occurrence. The exception context may have some data that are explicitly passed by the signaler. Additionally, it may contain information implicitly passed by the language runtime. For example, the information could be the name of a method or a class that is signaled by an exception. For performance analysis, the exception context helps to define the execution path of a scenario model.

### 2.1.2 Exception Handling Models

When an exception is encountered, the handler takes actions according to different exception handling models, which may be complex. Yemini and Berry [Shaula85] identify five exception handling models: transfer of control, two kinds of terminating, retrying and resumption. Robert Miller and Anand [Miller97] develop a termination model, a resumption model, a retry model and a replacement model. Peter and Russell [Buhr00] derive non-local transfer, a termination model, a retry model, and a resumption model. Considering these sources and all bindings between exceptions and handlers, five typical exception handling models are summarized as follows:

A) **Resumption Model**

If the resumption model is chosen, the control flow transfers from the escape point to the handler to correct the improper operation. It then continues from the escape point.

B) **Propagation Model**

If the propagation model is chosen, the exception is propagated further back in the dynamic call chain, to be handled by some alternative try block. Propagation is the default for exceptions, for which no handlers are found in a try-catch block. Propagation is also the default for exception objects with a matching when-clause if no other execution control is
specified in the when-clause. Propagation implies that exception handling has only partially been concluded.

C) Retry Model

If the retry model is chosen, the execution is resumed from the beginning of the try block. There must be a clear beginning for the operation to be restarted. Usually the beginning of the guarded block is the restart point. Retry implies that the best way to continue the application is to re-execute the entire execution from which the exception occurrence arose. Usually the retry model implies that exception handling has brought the application back to a stable state.

D) Abort Model

If the abort model is chosen, the control flow transfers from an escape point to a handler, terminating the intervening block. Once the handler completes, the control flow continues as if the incomplete operation in the guarded block had terminated without encountering the exception. The abort model implies that the actions of the exception handler have replaced the try block.

E) Termination Model

If the termination model is chosen, the execution of an entire application is terminated. Choosing the termination model implies that the exception cannot be handled (i.e. the exception occurrence is indeed severe).

In the advanced languages like C++ and Java, the try-catch clause supports the abort model if the encountered exceptions are in the set of exception types that the immediate catch clause can catch. The propagation model is automatically supported if the encountered exceptions are not in the set of exception types from the immediate catch clause but in the set of exception types of the next catch clause. However if some raised exceptions are beyond the set of exception types from all catch clauses, the termination model is deployed.
As for the rest models such as a resumption model, and a retry model, they can be implemented through redirecting code or algorithm.

2.1.3 Behaviour Analysis for Exception Handling

The behaviour of a program is derived from exceptions and the exception handling models. Given a program with FunctionA, FunctionB and FunctionC, Figure 2-2 shows a possible scenario for an abort model. In the try block of FunctionA, FunctionB is invoked, and FunctionB invokes FunctionC. In Figure 2-2, there is an escape point that raises exception e4 in FunctionC. If in a guarded block a statement invokes the method that contains the escape points, this statement is also an escape point. Thus an escape point is defined in FunctionB in Figure 2-2. The different possible escape points separate the try block into pieces.

![Figure 2-2 A Possible Scenario of the Exception Mechanism](image-url)

**Figure 2-2 A Possible Scenario of the Exception Mechanism**
Each of the pieces of each guarded block that has escape points will be represented in a performance model with input parameters such as CPU demand and visit counts. The activity of a component can be abstracted into two parts: regular activity and exceptional activity, shown in Figure 2-1 and Figure 2-2 as unshaded and shaded rectangles, respectively. The regular activity implements the normal behaviour without encountering exceptions while the exceptional activity provides measures for handling the exceptions. If an exception is raised in a regular activity, the control flow moves to exceptional activity. After the exception is handled, the system returns to some regular activity.

2.2 Scenario Model Notations

The behaviour of a program will be modeled by scenarios, which describe the sequence of operations, including branching and merging of paths, forking and joining for parallelism, and repetition. A scenario is the starting point of creating a performance model [Smith90][Smith02]. As discussed in Figure 1-2 in section 1.1, a performance model is derived from analyzing the behaviour of a scenario with performance feature like workload, CPU demand, rate, state and so on. The Use Case Map notation is used to define scenarios in this thesis. However there are many other scenario notations like UML [Gomaa00] (Unified Model Language) behaviour diagrams, the performance information notation for UML defined in MARTE (Modeling and Analysis of Real-Time and Embedded Systems) [Object09], and CSM [Petriu04] (Core Scenario Model).

UML behaviour diagrams [Gomaa00] provide the important information such as behaviour and resources, which comprise the scenario model. The sequence or activity diagrams can be used to define the scenario and the state chart can express the sequence of behaviour. The deployment diagrams describe how objects are mapped to the processing resource. The OMG standard profile MARTE [Object09] brings resources into consideration and allows UML diagrams to be annotated with performance information.
CSM is proposed in paper [Petriu04], to act as the bridge from UML SPT (Schedulability, Performance and Time) profile [Petriu02b] to various performance models such as queueing networks, LQN, and Petri Net. CSM describes the behaviour in the context of resource usage, which is used in the performance model definition.

A Use Case Map (UCM) defines a high-level design model to help a user express and reason about a system's large-grained behaviour, which was contributed by R. J. A. Buhr. UCMs [Buhr96] are defined as causal scenarios, architectural entities or behavior patterns. A UCM is a collection of elements that describe one or more scenarios unfolding through a system. UCMs are a visual notation for use cases [Jacobson93] with extension of a high level of abstraction. UCMs express behaviour along the paths by sets of responsibilities. The trace of a set of responsibilities along a path can be referred to a scenario. The scenarios enable a user to grasp the behaviour of the system without getting lost in execution details.

Use Case Maps make a good basis for performance modeling because they have many features for expressing behaviour. They can be expanded with detail so that the system is easy to understand. UCMs have sufficient formality for construction of performance models, which was shown in [Petriu02b].

The basic elements [Buhr96] in a UCM are a start point, an end point, paths, responsibilities (representing processing activities) and components. There are also forks and joins and stubs, which represent a sub-map. UCMs are composed of paths with responsibilities, which may traverse components as well as scenarios or a system. A path can be interpreted in the behavioural terms as a scenario. Its visual representation is a line with a start point and an end point. A filled circle represents a start point, which indicates the stimulus and a set of preconditions to start the path. A bar ends a path and shows the results of the path (see Figure 2-3 (A)). The paths are routes along which chains of causes and effects propagate through the system. There may be named responsibility points along any path. Responsibilities shown in Figure 2-3 (A) are denoted with crosses along the path. Responsibilities represent the localized actions and functions that a system must perform at
the specified point. A rectangular box shown in Figure 2-3 (A) is a component. The components represent entities and objects that are encountered during the execution of a scenario. They can represent both hardware and software resources in a system such as objects or modules, processes or threads, and physical devices. Stubs are containers for sub-maps, and are represented as diamonds. A top-level UCM is referred to as a root map. The path segments coming in and going out of stubs are identified on the root maps. There is a very simple example in Figure 2-3 (B) that represents a scenario. The upper map is a root map with a stub Selection, with two exit points, and the lower map is the expanded scenario for the stub.

![A) Basic Elements](image)

![B) Simple Scenario with a Stub](image)

**Figure 2-3 Use Case Map Notation and a Simple Scenario**

A UCM can also model more complex scenarios including parallel paths and alternative paths. A scenario of data upgrading for authenticated users is used to illustrate fork and join concepts in UCM. If explicit concurrency appears in the same scenario, it should be expressed as parallel path segments split by the AND forks and gathered by the AND joins. Figure 2-4 shows an example of a Use Case Map for the session of data upgrading with joining and forking. The AND fork splits a single path in the scenario into two or more parallel paths. The AND fork indicates the beginning of the concurrency, and the AND join is used to end the concurrency. The scenario continues executing only after all the parallel
paths have joined. For example in Figure 2-4 only after *upgrading data* and *progressing display* have been done, can *done message display* be performed.

The OR fork and the OR join represent alternative paths. The OR fork expresses that a single path splits into two or more alternative paths. Only one of the possible branches may be traversed after the OR fork. The OR join shows the alternative paths to merge in a single path. The OR join indicates that at least one of the possible paths leading to the OR join needs to be traversed before proceeding further. In Figure 2-4 there are two alternatives after the system verifies the user: valid user and invalid user. An OR fork is used after *verifying user*. An OR join indicates that the transaction for update data is finished. In UCMs forks do not have to be matched with joins.

A token [Buhr96] representing an invocation by a user (from outside the map) is introduced at the start point and traverses the paths, following branches with the stated probabilities, and forking into multiple tokens at forks. A single path may be occupied by multiple tokens at once for concurrency. At an end point, the token or tokens leave the system. A start point can be annotated with the properties of the token source: either an open arrival model with a rate, or a closed model with a number of users and a mean think time. In the case of multiple start points the tokens have classes; this case is not considered here.
2.3 Performance Modeling Formalisms

A performance modeling formalism is an approach to creating a quantitative (mathematical or computational) model which predicts performance measures, based on parameters representing resource usage by the system operation. It includes an underlying theory, a modeling language, and methods for creating and solving the models. A performance modeling formalism has a model definition, which represents the domain-specific elements and links in its own language. It includes mathematical or computational methods for solving a model and deriving the performance results. Researchers can develop a performance model in a formalism, solve it, and then analyze the performance of a system. A performance modeling formalism turns to be an essential part of performance modeling practice.
There are many performance modeling formalisms, techniques and their extensions. They can be categorized into two major classes according to their construction and solutions. One is queueing-based modeling formalisms including queueing-networks and their extensions. They support the product-form solution and approximate solutions. The other class is state-based model formalisms. They provide the analytical solution through generating the state space and computing state probabilities. The strengths and weaknesses of model formalisms in the same category are pretty similar. The following sections survey two analytical formalisms and their solution algorithms. The comparisons are made between the queueing network based category and the state-based category.

2.3.1 Queueing Network Modeling

Queueing Networks [Kleinrock75] consist of several service stations representing resources in a computer system and a set of users representing the workload. Queueing Networks have been applied for capturing the interaction between a system workload and a system's resources in a conceptually simple manner. Deploying Queueing Networks in performance analysis has been expanded by enhancing modeling power and efficient solutions. Queueing networks with special structure are identified as product-form queueing networks [Jackson63] [Gordon67], which are the open or closed queueing networks with exponentially distributed interarrival and service times. They have an analytical solution [Baskett75] which has been extended to networks with several job classes, different queueing strategies, and general distributed service times. The first efficient algorithm was developed in [Buzen73]. Others include the convolution algorithm [Reiser76], the MVA algorithm (mean value analysis) [Reiser80], the RECAL algorithm (recursion by chain algorithm) [Conway86], and the flow-equivalent server method [Chandy75]. For very large networks, execution time of the preceding algorithms is not acceptable. Many approximation algorithms have been developed for this purpose such as Bard-Schweitzer approximation [Bard79] [Schweitzer79], and the Self-Correcting approximation [Neuse81] [Chandy82].
Most practical problems lead to non product-form networks. Based on the idea in [Bard79], the MVA algorithm is modified to approximately solve the network with priority scheduling and the network with FCFS nodes and different service rates for different job classes. Many researchers have considered some classic problems that violate product-form network conditions. In [Bryant84] author suggested an extension of the MVA for approximate analysis of priority networks. The extended queueing networks [Edward85] were introduced to take the simultaneous resource possession into account in performance modeling. Surrogate delay methods were developed in [Jacobson81][Freund83][Jacobson82] to analyze simultaneous resource possession. An iterative procedure for mutually providing the parameters between two generated models is deployed for the estimation of the delay time and the determination of the performance measures. The method of surrogate delays was also generalized by Silva and Munzt [Silva87] in non product-form networks to model simultaneous resource possession by customer chains. The delay can also be caused by the competition for software resource and that is called serialization delay. An iterative technique and a decomposition technique [Thomasian83] were proposed to model the effect of the serialization delays to the computer system performance. The fork-join networks with synchronization were investigated in [Duda87] and performance analysis was based on the decomposition and the equivalence in queueing networks for performance indices.

Layered Queueing Networks (LQN) [Franks00][Franks08] area type of the extended queueing network. An LQN combines features from Method of Layers [Rolia92][Rolia95] (MOL) and Stochastic Rendezvous Network [Woodside89][Woodside86]. LQN uses surrogate delays to solve the simultaneous resource possession problem arising from the nested calling pattern in a system. LQN solution can model synchronous messaging semantics, asynchronous messaging semantics, and forwarding messaging semantics and solve the models with the intra-task fork-join interaction. LQN solution can model multiple phases of services for serialization delay. The LQN models incorporate sequences of activities, which support sequential behaviour as well as the fork/join and branch/merge behaviour. The layered approach gives an enhanced ability to efficiently solve the structured resource models, such as client-server with layered software architectures.
2.3.2 State-Based Modeling

State-based models define a set of system states and the transitions between them, with transition rates and probabilities. Two typical state-based modeling approaches are discussed in this section. One is the process algebra based approach and the other is the Petri net based approach.

Process Algebras focus on the functional aspects of concurrent systems such as observable behaviour, control flow and synchronization. These aspects are described as collections of entities, *processes* and executing atomic *actions*, which are used to describe concurrent behaviors and synchronized communication. Several stochastic extensions of Process Algebras (e.g. [Hermanns02]) are proposed for the performance evaluation. With the time consideration and performance property investigation, different stochastic process algebras are developed: TIPP (Timed Process for Performance Evaluation) [Gotz93][Hermanns00] by Herzog et al, PEPA (Performance Evaluation Process Algebras) [Hillston96] by Hillston, and EMPA (Extended Markovian Process Algebra) [Bernardo98] by Bernardo. The performance tools like the TIPP tool, PEPA Workbench and Two Towers for EMPA, are developed to support them.

Those process algebra extensions associate exponentially distributed random time delays in actions. The derivation graph of the model is used to generate the underlying stochastic process as a continuous time Markov Chain. The performance analysis usually requires the numerical solution of the underlying Markov chain derived from the derivation graph. The performance measures such as throughput, average delay time and queue lengths can be calculated from the steady state distribution, which associates to a set of components or behaviours.

Petri Nets (PNs) are a formal modeling technique to specify the synchronization behavior of a concurrent system. A PN defines the structural components through a set of *places*, *transitions*, and *arcs*. Tokens are indistinguishable markers that reside in places, and are used
to specify PN states. The behaviour of the PN is governed by the firing rule. The PNs with stochastic timing (SPN) were proposed to link PNs to the field of performance evaluation [Molloy82]. The extension of SPN, GSPN (Generalized Stochastic Petri Net), was proposed in [Marsan84], where stochastic timing is mixed with deterministic null delays. Therefore the timing and the logical evolution of a system can be represented in a model. Stochastic Activity Networks (SANs), introduced in [Sanders00], have additional structure such as gates so that more sophisticated firing rules are introduced into nets. There are many other extensions including Colored Petri Nets in which tokens are distinguished and may carry data.

The state-based model formalisms such as Petri Nets have the similar solution with Process Algebras. A Continuous Time Markov Chain (CTMC) is derived from a GSPN model. The CTMC state space corresponds to the reachability set of a PN derived from a GSPN model. The transition rate from one state to the other is obtained as the sum of the firing rates of the transitions that are enabled. The computation of performance indices defined over a GSPN model can be performed through solving the properties associated CTMCs. This research uses GSPNs because they have been widely used for computer system modeling [Marsan95].

2.3.3 Comparison among various methodologies

Queueing network models including extended queueing models can represent many features of computer systems like parallel execution including fork-join, branching execution, synchronization, simultaneous resource possession, serialization delay. State based models can also represent every feature of computer systems, but practical problems arise when the models are solved. These problems are the state explosion issue (a limitation for computation) and the exponential state residence time assumption (a limitation for expressiveness). Limitations of queueing networks are mainly on expressiveness. Queueing network models cannot express the decision based on the performance data. Thus one can remedy the other from solution perspective.
Queueing networks have different kinds of solution methods, depending on the nature of the networks. Some networks require the calculation of state probabilities using a CTMC model, similar to the state-based model techniques. Product-form networks [Baskett75] have numerical solution algorithms like convolution [Reiser76], MVA [Reiser80], RECAL [Conway86], the flow-equivalent server method [Chandy75], and the approximate methods [Bard79][Schweitzer79] based on MVA. Thus those algorithms are of much lower computational complexity. However the necessary assumptions are too strong for this research. Extended queueing networks [Edward85] are developed for non product-form network problems with iterative algorithm, surrogate delay [Jacobson81][Freund83][Jacobson82], and decomposition techniques in those systems. These solvers can solve the classic problems such as simultaneous resource possession, serialization delay, and parallel processing including asynchronous task and fork-joint systems, all of which are beyond the scope of product-form queueing network.

The LQN solver [Franks00] employs MVA [Reiser80] and surrogate delays [Jacobson81][Freund83][Jacobson82] to provide generally scalable solutions for complex systems. The expressiveness of LQN is sufficient to capture features like concurrency, thread/process pool limits, simultaneous resource possession, remote procedure call, thread/process blocking delays, branching activity flows, non-blocking RPC, fork/join and parallel consensus, multiprocessors/cores, priority and fair-share scheduling [Li09]. LQN models the system through the nested remote procedure calls or asynchronous messages crossing multiple layers LQN is used to determine throughputs and delays, and the contention for software and hardware resources, and to identify bottlenecks [Neilson95]. Using approximate MVA, the LQN can usually estimate performance within 5% errors for a system with tens to hundreds of customer classes and of host processors.

Although LQN is powerful enough to address many features, it has limited expressiveness for certain behaviours. For example, LQN can not estimate cache miss frequency or
timeouts on RPC. In these cases LQN lacks the capacity to model the decisions in a scenario; it is noted that the decision depends upon performance itself.

In state-based modeling, Petri nets and process algebras have often been used for performance analysis. Great efforts have been made to find efficient solutions. In Petri-net based approach, there are Timed Petri Nets (PNs), Stochastic Petri Nets (SPNs) and then Generalized Stochastic Petri Net (GSPNs). In process-algebra based solution, there are Process Algebra (PA), Stochastic Process Algebra (SPA), TIPP [Gotz93][Hermanns90], PEPA [Hillston96], and EMPA [Bernardo98]. Donatelli et al. [Donatelli95] compared PEPA with GSPN from different aspects such as model construction and the size of the state space for one application at the same complexity level, quantitative analysis and qualitative analysis and so on. GSPN is a more mature formalism than PEPA in the efficiency of the analysis algorithm. GSPN is very expressive. A GSPN model has the explicit notation of the states and expresses the rewards and performance measures. It can capture features like concurrency, synchronization and conflicts. Researchers also put effort into reducing the state space size through state aggregation, state space decomposition, and surrogate delay for state-based modeling approach (e.g. [Ribaudo95]).

However, the solution approaches for GSPN and PEPA are both based upon the numerical solution of the underlying CTMC. The state space size and the solution time tend to grow exponentially with system size. This is the state explosion problem. The model computation time may be unacceptable or there is a huge memory requirement. Although there is much progress in handling a large state space, it is still a problem for many industrial systems. For instance a GSPN solver, SPNP [Trivedi98] could not solve a simple model if the number of initial tokens in a certain place was increased to over 1000 in a GSPN model. Also there are some limits to expressiveness like most of the transition firing times must exponentially distribute in a GSPN model.

Vernon [Vernon87] made a comparison of Queueing Networks and Performance Petri Nets (PPN), which agrees in broad outline with the foregoing. Queueing Networks provide
efficient evaluation and are suited to exploration of performance. PPN could express synchronization events which determine the system performance, but due to the expense of computational resource, it is limited to small to moderate size systems. Although most of Vernon's judgments still apply, there are changes. Model formalisms and solutions have been evolving in the past decades, and have overcome some of the limitations. For instance, extended queueing networks like LQN incorporate synchronization primitives and Petri nets are extended to include colored tokens. Bause [Bause93] even combined them as Queueing Petri Nets, however they must be considered as state-based modeling because the solution is based on steady state probabilities including the states of the queue [Bause94] [Bause95].

Some comparisons of the categories are discussed in section 2.3.1 and 2.3.2. Queueing-based modeling formalisms in general have efficient algorithms as well as the direct feedback on component level. Representing the influence of the complex logic especially decision making on states of a system instead of predefining some variables like the mean outcome of the logics as probability in a queueing-based model remains to be studied and solved. State-based modeling formalisms are closer to the behavioral description and easy to capture the states and decision upon states. However it suffers from the state explosion problem. Based on [Tribastone10] (and many other analyses), we conclude that LQN is "scalable" (with limits, naturally) and GSPN is "less scalable".

2.4 Selected Performance Model Formalism Notations

This section presents the notations of LQN model formalism and GSPN model formalism since they are used to illustrate the performance modeling methodology advocated in the thesis. Certainly HPMM is not limited to these two formalisms. LQN and GSPN are selected because each one is a typical model formalism in the corresponding category. Each of them has characteristics of its category as discussed in section 2.3.
**LQN notations**

The semantics of LQNs will be explained using UCMs to define their behaviour. A UCM is shown in Figure 2-5 (A), with two scenarios. Responsibilities \( a, b, e, \) and \( p \) are along the path of scenario \( S1 \). Responsibility \( p \) indicates the pure delay like the sleeping time of a thread or a process. Responsibilities \( d, e, f, \) and \( g \) are along the paths of scenario \( S2 \). Components can be nested and they can be referred as sub-components. In Figure 2-5 (A), the inner components with solid lines are called "tasks" \( T1 \) to \( T5 \) and they are referred as sub-components. In this case, these inner components represent processes, while the outer components \( C-A, C-B, C-C \) are just called "components". Each of these outer components is loaded on a different processor \( P-A, P-B, P-C \) respectively.

Layered queueing arises when a software server task has to wait for service at its processor, or at another task. Figure 2-5 (B) shows the tasks and the requests for service that are implied by the UCM in Figure 2-5(A), with one sub-model for each scenario. Tasks are

---

**Figure 2-5 A USE CASE MAP WITH TWO SCENARIOS AND LQN SUB-MODELS**

Layered queueing arises when a software server task has to wait for service at its processor, or at another task. Figure 2-5 (B) shows the tasks and the requests for service that are implied by the UCM in Figure 2-5(A), with one sub-model for each scenario. Tasks are
indicated by rectangles, divided into a field for the task and its properties, and fields for entries which provide classes of service. The entries have been named for the first responsibility they perform. In S1, entry a makes a request to entry b and waits for the reply; this kind of "synchronous" request is indicated by an arrow style with a filled head. Entry b then makes a similar request to entry c, and entry c does to entry p. In scenario S2, entry d makes a synchronous request to e, which forwards it for processing to f and g (the forwarding task does not wait for a reply; the reply goes to the originator). Forwarding request is indicated by a dashed arrow. After the reply from T4, entry g does a further operation for responsibility h (this kind of delayed operation is called a "second phase"), and then a second phase request to entry j. This final request is asynchronous, with no reply, indicated by an arrow with an open arrowhead.

Tasks and processors are both resources which act as servers. Each resource has a queue and a discipline, and may be multiple (that is, a multi-server such as a multithreaded task or a multiprocessor). Each task has a host processor, which identifies the physical device that carries out its operations. The processors for each of the dashed components in the UCM are indicated by ovals attached to each task. The pure delay like responsibility p does not attach to any component. Thus a task and a processor should be created with multiplicity as infinity. As shown in Figure 2-5(B), there are tasks T1, T2, T3 and TP for S1, and T1, T2, T3, T4, and T5 for S2. T1 and T2 reside on processor P-A; T3 and T4 reside on processor P-B; T5 resides on processor P-C; and TP resides on processor P-P.

GSPN notations

An example taken from [Marsan95] is used to explain GSPN as well as their notations. GSPN are a performance analysis tool based on a graphical representation. GSPN are an extension of Petri Nets. In GSPN, some transitions are timed and the others are immediate. Elements of GSPN models include places, transitions, arcs and tokens. Places, graphically represented by circles, stand for possible local system states or conditions. Tokens, indicated by black dots, reside in places and together represent the Petri Net state. Transitions
represent events that change the state of the system. The event can be the completion of an activity (timed transition) or the verification of some logical condition (immediate transition). A transition is enabled if all of the places attached to its input arcs have a token present. A transition which is enabled may fire, causing one token to be removed from each of its input places, and one token to be added to each of its output places. In GSPN, enabled immediate transitions fire at once, while enabled timed transitions have a constant rate of possible firing as long as they are enabled, giving an exponential delay if they are not disabled. Arcs are used to connect between places and transitions in both directions to indicate the sequence of the event occurrences and the local state transformations.

The GSPN model in Figure 2-6 describes a readers and writers system, in which a set of processes may access a common database for either reading or writing. The GSPN model has seven places \( P = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7\} \) and seven transitions \( T = \{T_1, T_2, T_3, t_4, t_5, t_6, t_7\} \). \( T_1, T_2 \) and \( T_3 \) are timed transitions. The rest is immediate transitions. \( T_1 \) models the process activity not related to the database access. \( T_2 \) and \( T_3 \) model the database read activity and the write activity respectively. Two places are initially marked: \( P_1 \) and \( P_4 \). The former indicates how many jobs there are in the net, while the latter represents whether a job is permitted to execute a write activity or a read activity. There is an inhibitor arc, represented by a circle-headed arc from the place \( P_6 \) to the transition \( t_7 \). It means any number of readers may access the database currently, but a writer requires exclusive access to the resource.
2.5 Transformation from Scenario Model to Performance Model

Various papers have described the transformation from the scenario model to the queueing-based model. In paper [Cortellessa00][Grassi02], the authors present a methodology that produces a queueing network model from UML diagrams like Use Case Diagrams, Sequence Diagrams and Deployment Diagrams. Especially in paper [Grassi02], the extended UML with the adoption of mobility in paradigms in software architectures was deployed. Petriu and her students [Petriu99][Petriu02a][78] proposed systematic methods of building LQN models from software architecture. The structure is specified through UML collaboration diagrams and the behaviour is described through Sequence or Activity diagrams in the software architecture. Woodside and his students [Petriu05] present an approach through which LQN performance model can be derived from the system scenario models described in Use Case Map (UCM). The UCM2LQN tool [Petriu03] is developed to automate these methods and it has been integrated into the UCMNavigator.

Various papers have also described the methods deriving state-based model from scenario model. In [Pooley99], the author described some ideas on the derivation of Stochastic Process Algebra models from UML diagrams. In the paper [King00], the UML specifications such as Collaboration and State-chart diagrams can be transformed into GSPN. The paper uses the protocol of the communication, the alternating bit protocol as example to approve it. In the paper [Bernardi02], the authors proposed an automatic transformation of State-chart and Sequence diagrams into GSPN. The transformations are based on the UML Meta model. In the paper [Lopez-Grao04], the authors were focused on activity diagrams and a tool was developed. The tool translates each diagram element into a GSPN model and the composition of these nets results in a complete GSPN model.

It has been seen that there is an intuitive connection between a scenario model and a performance model from section 2.4. Both of them describe the behaviour of a computer system from different perspectives. An activity in a scenario model is mapped to an element
with a time delay in a performance model. Resources and components in scenario model are translated as constraints in a performance model. The sequence, logics, etc can be converted the links among the elements and constraints in a performance model.
3. Challenges and the Approach to a Solution

This chapter starts with the challenges in performance modeling of complex systems using different modeling formalisms. It also examines the combination of LQN and GSPN formalisms in a hybrid modeling method, what questions such a combination poses, and how they can be resolved.

3.1 The Problem

Decision making to control the execution path is introduced in systems to help them adapt to the execution environment, to enhance the reliability or robustness, or to introduce flexibility in the use of new features. The exception handling mechanism discussed chapter 2 is a good example. An example is used to illustrate the logic part of an application and explain the decision making of this logic part. Assuming the queue is deployed in this application, each request is enqueued to be processed if the queue is not full. If the queue is full, an exception is raised, using one of the exception handling models described in section 2.1. The decision on whether the queue is full, which drives different work flows in a scenario, is made by a part of this application, which is called the “logical part”. Around the logical part, different work flows execute, determined by decision making. If the probability of an exception were constant, the decision could be modeled by a random choice, which is always straightforward. However the probabilities of an exception depend on the different workloads, and must be derived as a part of the performance model.

To tackle performance prediction and performance evaluation of this kind of system, different performance modeling formalisms are deployed to develop analytical performance models. Generally researchers can select formalisms among queueing network modeling category or among the state-based modeling category considering the popularity of formalisms and their knowledge and expertise on them. LQN can be considered to model this application due to its scalability and effective solution as discussed in Section 2.3.1.
However LQN could not represent a logic condition especially decision making upon the performance of the application like delay, queue length, overflow and etc. Although the numbers of requests or request probabilities can be used with the assumption, it will impact the accuracy and the generality of modeling approach in LQN model formalism. GSPN is also considered due to its capacity to cover the gap by modeling logic conditions through the tokens in places and representing decision making in an analytical model. However the state space size and the solution time of GSPN discussed in section 2.3.2 prevent itself to be a feasible solution for a complex application. Neither LQN nor GSPN is ideally suited to modeling this kind of application.

This research considers combining LQN and GSPN in a kind of hybrid performance modeling, to take advantage of the strengths of both formalisms as shown in figure 3-1. The power of each formalism can be extended by the interaction with the other one under hybrid performance modeling framework. Both formalisms are complementary in modeling definition and solution computation. To combine LQN and GSPN, a hybrid performance modeling framework is developed, including model definition, sub-model and data exchange, and iterative solution among sub-models within a unified framework.

![Diagram](image)

**Figure 3-1 Hybrid Performance Modeling Idea on Extending LQN by GSPN**
3.2 The State of the Art

The concepts underlying the hybrid performance modeling framework like model definition, sub-model and data exchange, iterative solution among sub-models are not new. In the past decades, a lot of research has been conducted on combining different performance model formalisms or sub-models, and iterative solution idea and concepts for performance analysis and evaluation. In this section, the state of the art on these concepts is explored.

Several projects have developed an integrated model definition and solution environment that supports multiple formalisms and tools. HIT is an integrated tool [Beilner89], which supports modeling of computer system in a hierarchical manner based on a layered view. The HIT performance evaluation of the descriptive model includes the simulative and analytical techniques such as stochastic discrete-event simulation, exact result evaluation of product-form solution, approximate evaluation techniques for non-product-form solution, numerical evaluation for Markov chains, and sub-model analysis and aggregation [Beilner95]. IMSE (Integrated Modeling Support Environment), a support environment described by Pooley [Pooley92], supports several phases such as modeling study, experiments, model construction, and model analysis and reports during the performance analysis. It combines queueing and Petri-Net tools with others in a single system, with the goal of using models for different parts of a study, as appropriate, sharing the measurement data and giving composite reports. IDEAS (Integrated Design Environment for Assessment of Computer Systems and Communication Networks) [Fricks96], supports integrating diverse modeling engines through the common interpreter and provides a uniform and easy access to many existing performance modeling tools such as SPNP, SHARP, Distributed SPNP (D-SPNP) and Fluid Stochastic Petri Net (F-SPN). The FREUD project [Moorsel98] allows a user to select from registered formalisms, solvers and GUI tools and implements reusability for performance modeling tools. The FREUD project has the same purpose as IMSE and IDEAS but exploits web-based configurable tools. All approaches in this direction provide the common interface and environment for performance modeling solution. Although
computer systems can be modeled in different model formalisms, the interaction among different formalisms is limited and only the results from different formalism are composed together.

A more restricted kind of multi-formalism modeling supports only multiple state-based formalisms (such as stochastic petri nets and stochastic process algebras, and possibly queueing networks modeled at the Markov-chain level). Bause [Bause93] proposed Queueing Petri Nets, a formalism that combines Queueing Networks and Petri Nets. While it integrates the formalisms in the way this work also seeks to do, it models queues at the state level. Thus it does not improve the scalability of the solution and the state space explosion problem still exists. Sanders and his students [Sanders99][Daly00][Deavours02] worked on Mobius framework supporting multiple state-based formalisms and multiple solutions. Models defined in different formalisms are translated into equivalent models using frame components so that the solver may operate on frame component independent of model formalisms. The Mobius tool allows compatible formalisms to be translated into framework components through abstract function interface (AFI) [Derisavi02]. The DEDS toolbox [Bause98] presented by Bause and his colleagues integrates multiple formalisms for combining functional and quantitative analysis of discrete event dynamic system. It converts the expressed models into a common model interchange format, the Abstract Petri Net Notation (APNN). It allows combining different modeling formalism such as Queueing networks, Generalized Stochastic Petri Nets, Colored Petri Nets. The MODEST [Bohnenkamp03] tool, shown by Bohnenkamp and his colleagues, is a modeling language specifying process algebra. The model specified with MODEST provides multiple views ranging from finite-state automata, to timed automata, and discrete event stochastic processes such as stochastic automata and Markov Decision Processes. It implements a single formalism multiple solution. The MODEST tool allows different external tools for performance solution through the proper adaptor modules called satellite modules. Each module can generate the state-space implementation of a MODEST specification. Vittorini and her colleague propose OsMoSys (Object-based multi-formaliSm MOdeling of SYStems)/DrawNET framework [Vittorini04] [Marco05], which use the modeling approach to build multi-formalism models and workflow management to implement multi-solution.
Combined solution with strong multi-formalism has been achieved either by a hierarchical combination of results, or by iteration. In a hierarchical combination the solution of one model provides parameters to another, higher in the hierarchy, and there is no iteration. Balbo, Bruell et. al. [Balbo88] combined queueing network and GSPNs for complex models through exploiting the best features such as flow-equivalent server and the efficient solution. The QN models are developed for the input parameters of a GSPN model and vice versa. These are hierarchical solutions in which the QN sub-model is solved first as input to the GSPN sub-model, or vice versa. Heidelberger and Trivedi [Heidelberger83] modeled the possible states of activity of parallel programs by a high-level Markov model, whose transition rates were determined by a QN sub-model for each of its states. HIT supported hierarchical combinations of results. SHARPE [Sahner96] for reliability and performability analysis allows combining different formalisms such as acyclic directed graphs, Markov and semi-Markov models, product-form queueing networks, and generalized stochastic Petri net. Models can be developed by using hierarchical composition and they can be solved through exchanging results within the framework. The report [Olivia04] described a hybrid model which included LQN and Markov Chain sub-models and some auxiliary computations. The Mobius tool described above also has facilities for combining any model, defined separately in any other formalism, with these in an iterative solution strategy much like that described here. In fact Mobius provides a candidate solver for this work, as described in chapter 4.

Unrelated to hybrid modeling but also exploiting iteration of sub-models, there are approximation methods which break a single model into sub-models in the same formalism and solve them by iteration. Ciardo and Trivedi [Ciardo91] proposed "nearly independent decomposition" for stochastic Petri Net models. In their work, each sub-model exports some measures to the other sub-models and can be solved through a fixed-point iteration and all sub-models are Petri Net model. Campos and his colleagues [Campos93] employed a similar iterative technique for the approximate computation of stochastic connected marked graphs. Ciardo and Miner proposed and implemented SMART [Ciardo96][Ciardo01] for performance model analysis. SMART integrates state-based models, Discrete Time and Continuous Time Markov Chains as well as high-level formalisms including Stochastic Petri
Nets and Queueing Network. The state-space generation techniques are deployed in the analysis of the logical behaviours and the Kronecker numerical solution is used for stochastic and timing behaviour. The sub-models are solved through the numerical methods and simulation. Results of sub-models are exchanged through fixed-point iterations. The iterative technique is based on the net decomposition with a single cut. These solutions have purposes similar to this research, such as state-space reduction, computing time reduction and scalability. Similar strategies and algorithms are employed in the present hybrid performance modeling method.

3.3 Weakness of the State of the Art

It is obvious that researchers’ efforts discussed in section 3.2 enhance the power of analytical performance modeling through hybrid modeling idea, sub-model conception, and iterative solution. However there are weaknesses in these efforts. These weaknesses are summarized as: a continuing non-scalable solution, a lack of a methodology for hybrid model construction, and having limited extendibility.

Some efforts retain non-scalable solution methods. SHARPE[Sahner96], SMART[Ciardo96][Ciardo01], the Mobius framework [Sanders99][Daly00][Deavours02], DEDS [Bause98], MODEST [Bohnenkamp03] and OsMoSys [Vittorini04][Marco05] integrate multiple formalisms and solutions within a framework and provide a hybrid model through data exchange. However all formalisms in these frameworks are limited to the state-based modeling approach. The state explosion is still a big challenge in the hybrid solution. Queueing Petri Nets [Bause93] merges queueing network into Petri Nets as one formalism and it still faces the same challenge as the state explosion issue. A goal of the present research is to avoid this problem by using only quite small state-based models (for instance, with less than 1 million states).

No previous research has described a methodology for constructing models in different formalisms that are consistent and demonstrably cover the system behaviour. HIT
[Beilner89], IMSE [Pooley92], IDEAS [Fricks96], and FREUD [Moorsel98] project provide the integrated tools including multiple formalisms and solvers. However, none of them describe how to construct models and compose the performance model results. The interaction of sub-models is very limited in the generality of model construction. Correct sub-model construction is a major goal of the current research.

The efforts like [Ciardo91][Campos93] have the limited extendibility, because they are focused on a particular class of models. The approaches as independent decomposition in stochastic Petri net model and iterative technique for the approximate computation of stochastic connected marked graph are focused on homogeneous state-based sub-models. The common state explosion problem and scalability issue make these approaches have the limited adoption.

The issues above will be addressed by a mature hybrid methodology through the following aspects:

- **Scope:** This aspect will address what kind of applications the proposed HPMM can cover so that the hybrid performance modeling solution can be applied for performance evaluation and prediction.

- **System Partitioning Method:** Within the defined scope, HPMM will systematically address how to partition a given system from the performance modeling perspective. Each partition will correspond to a chosen formalism and may be called an "aspect" of the performance model.

- **Sub-Model Construction:** Based on the system partition, HPMM will provide an approach to constructing a behavior sub-model which can cover one aspect of the system and the abstraction or the approximation of the remaining aspects of the system.

- **Consistent data relationships:** To feed the parameters of the abstraction or the approximation, data exchange between sub-models will be defined.
• Solution: HPMM should include a tool which can iterate solving each sub-model and feeding the proper results back so that other sub-models can use them.

3.4 Hybrid Performance Modeling Methodology Overview

Hybrid performance modeling methodology (HPMM) combines multiple performance models in different formalisms and solves these models iteratively. Each of these groups of issues discussed in section 3.3 is covered by HPMM. HPMM will be examined in the light of the existing hybrid modeling methods surveyed in section 3.2.

Scope: HPMM is proposed to handle the software system with stochastic behaviour, in which there is queueing for the service systems (web services, service oriented architectures, enterprise systems generally) and decision making based on performance data, such as resource allocation exceptions. The target domain of HPMM is the same as that of layered queueing networks. They may be layered systems, but non-layered systems are included as a special simple case. Then this research extends the domain of LQN by including systems whose performance depends critically on a certain class of decisions. LQN can already model stochastic decisions with a known probability, but needs extensions for decisions which depend on performance, such as timeouts, retries due to buffer overflows, and similar exceptions. This class of decisions is to be included. Petri nets have a general capability to express system logic, so all decisions in this class are in scope. Thus HPMM combining LQN and GSPN is employed to model the system with decision making upon the performance.

Both selecting the most suitable performance model formalism and maximizing the advantages of the formalism motivate us to combine different formalisms together to form HPMM idea. HPMM is developed as a systematic methodology involving behaviour decomposition, performance model construction, and iterative solution.
**Partitioning:** To identify the decision points and their influence, without being biased by the system structure, the partitioning is done within a behaviour description (scenario specification), rather than within a structural description. This turns out to have critical advantages for determining the boundaries of the sub-models and for enforcing consistency across these boundaries. For both of these issues it is essential to begin from the same description for all sub-models. This does not prevent the use of a pre-built LQN fragment that represents a part of the system that has no internal decision behaviour, however. What is necessary is that the part of the system being partitioned should be described by a single behaviour model. Thus HPMM applies the formalism-oriented behaviour partition. Formalism-oriented behaviour partition means that users analyze the different aspects of the system behaviour and partition it with a performance model formalism selection consideration.

The formalism-oriented behaviour partition is different from the time scale decomposition [Olivia04] [Hillston95] based on the nearly complete decomposability. The aggregates in [Olivia04][Hillston95] are formed by grouping states without first constructing the overall state space. High-level formalisms, such as LQNs in [Olivia04] and SPAs [Hillston95] provide a means to model a computer system systematically and the results of high-level are aggregated for the overall performance evaluation. The formalism-oriented behaviour partition is also different from structural decomposition with near independence that is exploited in [Ciardo91][Bohnenkamp03]. Although the iterative approximate solution is used for the homogeneous model formalism (see [Ciardo91][Bohnenkamp03]), the structure decomposition is only used for the weakly connected nets and the nets can be cut separately. The delay in one sub-model is the parameter of a substitution in the other sub-model. The formalism-oriented behaviour partition is also different from the hierarchical decomposition in [Balbo88][Bause93]. In the hierarchical decomposition, one sub-model is represented as an element in the other sub-model. It can be only applied to a computer system having the similar hierarchical structure. If different aspects of the behaviour may not be separated physically through the time scale decomposition, the near independent, or the hierarchical approach, the behaviour can be referred to as “entangled”. In the entangled behaviour, different aspects of the behaviour are woven together. Ideally the entangled behaviour can
be separated from the performance modeling solution perspective. After analyzing entangled behaviour, the suitable performance modeling formalism is chosen to model each aspect of the entangled behaviour. Therefore the entangled behaviour can be handled by hybrid performance model methodology. Different suitable performance modeling formalisms lead the partition in the behaviour of a computer system as behaviour fragments. It forms the formalism-oriented behaviour decomposition.

**Sub-model construction:** Well-established techniques can be used to make a sub-model from a behaviour fragment that is designated for one formalism. However the sub-model must be *completed*, in order to represent that fragment within the context of the entire system. In this work the completion is derived from an abstracted version of the remaining behaviour. The completion approximates the rest of the system, and the degree of approximation is determined in the behaviour domain. This provides a simplified model-building process with traceability from the entire system (behaviour) to the set of sub-models.

For simplicity of presentation, a single sub-model is created for each formalism. In some cases, it is also possible for one formalism to have several separate sub-models (for example, to represent different non-interacting exceptions). In such cases our single sub-model of that formalism has several non-interacting parts. In the models shown in this thesis, that means there is always one LQN sub-model and one PN sub-model. This is not a strong restriction, and can be relaxed to encompass other formalisms and multiple sub-models in each one.

HPMM includes how to develop sub-models in different model formalisms, identify the data exchange among sub-models, and solve sub-models in a hybrid performance model through the existing tools. The process is designed to guarantee the consistency of sub-models. The parameters of the approximated elements in one sub-model are derived from the other sub-models through data exchange. This sub-model definition process in HPMM is referred to as Coordinated Definition. HPMM provides an environment, on which users can develop and solve a hybrid performance model.
Compared to previous work of hybrid performance modeling surveyed in section 2.3, HPMM has several improvements. Compared to HIT [Beilner89], IMSE [Beilner95], IDEAS [Fricks96], and FREUD [Moorsel98], HPMM not only provides an environment for facilitating model definition and solution, but also identifies the essential connection among sub-models in different model formalisms. The method improves the capacity of a model formalism through data exchange between sub-models. Fundamentally the approach neither creates a new formalism compared to [Bause93], nor develops sub-models in a hierarchical way compared to [Balbo88]. The formalism-oriented behaviour partition and the coordinated definition of sub-model construction give the preparation for data exchange among sub-models so that the strengths of different formalisms are exposed in HPMM. The limitation of a model formalism can be compensated for by other model formalisms in the method. The objective of HPMM is to cope with a system with entangled behaviour for performance evaluation, for which a single formalism cannot handle the system elegantly due to limitations in its modeling power. SHARPE [Sahner96], SMART [Ciardo96][Ciardo01], Mobius [Sanders99][Daly00][Deavours02], DEDS [Bause98], MODEST [Bohnenkamp03], and OsMoSys [Vittorini04][Marco05] develop unified frameworks and multiple formalism models can be defined in the frameworks. However most of the formalisms are state-based model formalisms and the rest of the formalisms are product-form queueing networks, fault tree, and etc. HPMM is more general and it advocates selecting the “best” model formalism in each category. Therefore the hybrid method benefits from the strengths of each performance model formalism category.

HPMM provides systematic principles for behaviour analysis, sub-model construction and hybrid model solution, for modelers to follow. It uses the formalism-oriented partition, the coordinated definition of the sub-model construction, and an iterative solution tool.

**Solution:** In order to accommodate different solvers and potentially additional formalisms, the solution is found based on steady-state solutions of the sub-models. The influence of one sub-model to another is expressed by steady-state results which are used to determine
parameters (such as equivalent delays) within the completion of the other sub-model. Then a solution is sought by iteration, beginning from a default setting for all such parameters.

3.5 Hybrid Performance Modeling Procedure

To summarize the procedure of HPMM, four important issues need to be addressed: how to partition the behaviour of a system; how to build the sub-models in different formalisms for aspects of this system; what kinds of quantities and relationships need to communicate or exchange performance indices from one sub-model to another sub-model; the convergence behaviour of the fixed point iterative algorithm should be deployed in the solution tool. The basic steps in developing a hybrid performance model for a system are briefly summarized as follows:

1. from the description, design or implementation of the system, analyze the behaviour and identify the in-focus behaviour and out of focus behaviour for the intended formalism (section 4.2),

2. derive the formalism-oriented partition of the behaviour and obtain the separate aspect sub-scenarios \( S_1, \ldots, S_N \) from the scenario \( S \) for the intended formalisms \( F_1, \ldots, F_N \) (section 4.3.1),

3. construct separate performance sub-models \( M_1(p_1) \ldots M_N(p_N) \) through the user definition or the model transformation to a performance model from each sub-scenario with the Coordinated Definition. In general each sub-model \( M_i \) has a parameter vector \( p_r \) which is derived either from aspect description or from the other sub-models. For instance, the probability of the event occurrence cannot be calculated within the sub-model due to limitations of its intended formalism. These values are to be calculated from the results of the other sub-models (section 4.3.3 and section 4.3.4).

4. analyze the relationships among sub-models to:

   a. establish how each parameter vector \( p_i \) is determined from the result vector \( R = (R_1(M_1), \ldots, R_N(M_N)) \) given by all the sub-models. Here \( R_i(M_i) \) represents the vector of
results derived from sub-model $M_i$. This relationship can be defined in a set of equations as $p_i = f_i(V)$ for each sub-model (section 4.4 and section 4.5).

b. establish a consistent measure over sub-model solutions. A function $\Delta$ is defined such that $\Delta = 0$ if the results from all the sub-models are consistent, or $\Delta > 0$ otherwise. For example there is one result vector result for sub-model $M_i$ represented by $R_i$. The index is specified as $(R_{ij})$ to represent the $j$th result for sub-model $M_i$. $\Delta_j(R_j) = |(R_{ij} - \langle R_{ij} \rangle_{1:n})|$ is derived and there is $\Delta_j = \max_{i} (\Delta_j(R_j))$ (section 4.6).

5. solve sub-models iteratively (section 4.7):
   a. provide an initial value for each parameter in $p_i$ for sub-model $M_i$ to make it solvable;
   b. at each iteration, solve the sub-models in order $1, ..., N$ to obtain a new value for $R$.
   c. from the second iteration, after each iteration and test $\Delta < \varepsilon$ ($\varepsilon$ is the predefined convergence criteria)
      i. if testing fails, a) this iteration number does not exceed the predefined maximum iteration number, for each sub-model set $p_i = f_i(V)$ using the most recent results, and repeat from step 5b; b) this iteration number is equal to the predefined maximum iteration number, stop and report that the solver does not converge.
      ii. if testing succeeds, stop and report that the solution is converged and the results.

The most important three steps are analyzing the behaviour with formalism selection and formalism-oriented partition, constructing the consistent sub-models with the coordinated definition and solving the hybrid performance model with the existing solver tools. These three steps are emphasized in the next chapter. The rest is the same as common performance model analysis, construction and solution.
4. Hybrid Performance Modeling Methodology

This chapter describes the Hybrid Performance Modeling Methodology in detail, for the combination of LQN and Generalized Stochastic Petri Nets (GSPN).

4.1 Scope of Hybrid Performance Modeling Methodology

Hybrid Performance Modeling Methodology (HPMM) is proposed to extend LQN by GSPN to form a hybrid solution. The initial experience in HPMM with consistent model construction is described in [Wu05]. HPMM has the same target domain as that for LQN like stochastic behaviour. Thus HPMM can model the sequence of the operations as different processes on processors. HPMM is able to cover the parallel and alternative composition of such sequences, recursive to any depth of layers. HPMM has the capacity to support the parallel and alternative sequences that terminate without rejoining. HPMM is able to model multiclass systems with a scenario for each class like LQN. Furthermore, HPMM extends the target domain of LQN to support the decision making governing alternative sequences by GSPN. The decision making could depend on performance measures as well as on problem data or on attributes of the environment. The decision making upon performance measures is not well described in the queueing networks. This research is focused on general classes of systems dominated by stochastic behaviour and queueing, which also contain decision making. These classes of systems are illustrated as a UCM follows. Figure 4-1 shows a simple example of the system behaviour with a decision, as a UCM.
The example has the responsibilities R0, R1, R2, R3, ..., R8, and R9 along paths P0, P1, P2, and P3 in Figure 4-1. Responsibilities R0, R1, R2, and R3 are located in component C1 and R4, R5, R6, R7, R8, and R9 are in component C2. The notation branching decision element D1 is emphasized by a dashed circle. In the scenario of this example, the execution proceeds along the path P0 and finishes responsibilities R0, R1, R2, and R3. There is a decision made at the decision cell D1. The outcome of the decision leads the execution path branch along one of the paths P1, P2, or P3.

The salient features of the above example system are summarized. The system has one start point. The scenario of the system is dominated by stochastic behaviour, queueing (implicit, for components and processors), and service delay. The scenario has a decision (in some way as yet not stated) based on the performance of the system, which leads to different execution paths.

There are varieties of decisions that could be part of the above system. They can be categorized as follows. The hybrid or non-hybrid performance modeling solution is also briefly discussed for each category.

- Purely random decisions with fixed probabilities. To model this class of systems, the hybrid performance solution is not needed. Either LQN or GSPN is chosen to model based the nature of the problem.
• Decisions based on a problem data. So-called "scenario data" can be included in scenarios to model such decisions. Thus the tokens in places of a GSPN model can be used to model the predicates of the problem data.

• Decisions based on dynamic system data. The dynamic system data state could be a certain piece of data in the cache. Scenarios do not model this within its scenario semantics. This is outside of LQN and GSPN modeling scope. It is also out of the hybrid modeling solution scope.

• Decisions based on dynamic performance data. The performance data could be a delay along an execution path, queue length at a point in the path of scenarios like shortest queue first, and buffer overflows and similar exceptions. This is the primary target tackled by the hybrid modeling solution. Thus in the example in Figure 4-1, it is assumed that the decision in decision cell \( D1 \) is made upon the dynamic performance data.

Hybrid Performance Modeling Methodology in its most general form is employed to model a system with stochastic behaviour and queueing. The system includes the decision making and the decisions are made based on dynamic performance. For simplicity most of this thesis is restricted to two formalisms: LQN is used to model the whole system with the hybrid modeling framework. Due to lack of capacity to model the queue size, buffer and exceptions, LQN is extended by GSPN. GSPN is used to model the dynamic performance model to remedy the limitation of LQN. Thus extending LQN by GSPN becomes hybrid performance modeling solution in this research. And further, the application of HPMM is focused on the many kinds of exception handling, as a class of decisions.

“Extension” of LQN in this work is not an extension of the LQN tools themselves, but an extension of their reach and range of applicability, through combining them with GSPN tools in a kind of collaborative framework.
4.2 Behaviour Partition with Model Formalism Selection

The key first step in constructing a model is to decide what parts are modeled in what formalism. This is done by partitioning the behaviour model as defined by a UCM. Attention is restricted to the LQN and GSPN formalisms.

The algorithm of behaviour partition in HPMM is summarized as follows:

1. Identify behaviour cells in the UCM through behaviour analysis;

2. For every cell in the UCM, score it for modeling by LQN and GSPN. A table including common elements of a UCM with scores is provided;

3. From the scores, determine a performance modeling formalism for each behaviour cell.

A behaviour cell is defined for each UCM element (like responsibility, decision branch, merge, fork, join), augmented by cells for potential queueing where the UCM path crosses a component boundary. The behaviour cell is assumed not to be divided any more. For instance, the system with the behaviour model described in Figure 4-1 can be divided into behaviour cells as responsibility cells $R_0, R_1, R_2, R_3, \ldots, R_8$, and $R_9$, decision cells $D_1$, and queue cells $Q_1$ and $Q_2$. $Q_1$ and $Q_2$ are at the points of entry into components $C_1$ and $C_2$ respectively.

Every behaviour cell in a UCM can be scored as low, medium and high for a given performance modeling formalism. The consideration includes the capacity of the selected formalism to cover the cells, the time to solve the model, and scalability, as judged by modelers. The formalism with the highest score is assigned to the cell. The modeling formalism candidates are LQN and GSPN, being representatives of the queueing network modeling category and state-based modeling category.
The common behaviour cells discussed above are examined and scored. In fact, they are often observed in performance modeling practice. Responsibility cell is one basic element. Both GSPN and LQN have the capability to model it. This cell can be represented as a timed transition in a GSPN model and an entry in a task or an activity for a task in a LQN model. The score of responsibility cell is rated as medium/high for both GSPN and LQN.

Queueing cell is another basic element in the performance modeling excise. It can be modeled by both GSPN and LQN. However considering the computation time of solvers of GSPN and LQN and the scalability, the queue cell is rated medium/high for LQN and low for GSPN. Generally there is one more Responsibility cell following a Queueing cell.

Fork, join, merge cells are also common cells in UCM of systems and both GSPN and LQN are able to model them. From the computation time of GSPN and LQN solvers and scalability, both GSPN and LQN are recommended as medium/high for these cells.

Decision cell with fixed probability can be modeled by both GSPN and LQN and it is rated as medium considering the computation time and the scalability of the model. Decision cells are modeled by places and transitions that implement the decision logic in GSPN. LQN is not able to model the decision explicitly. Thus these cells are ranked as high for GSPN and as low for LQN.

Thus there is possibility that the scores for both formalisms could be low or high. If both scores of the behaviour cell for LQN and GSPN are high, it is equivalent that they are set to medium for both. If both score of the behaviour cell for LQN and GSPN are low, it means the behaviour cell has the abstract representation in both formalisms. For instance, the decisions cell with the dynamic system data has to be represented as probability in both LQN and GSPN. The score of this decision cell is low for both formalisms. This is not solvable in hybrid modeling idea discussed here and it is out of the scope of the present thesis. Therefore the scores of common behaviour cells for GSPN and LQN are tabulated in Table 4-1.
Table 4-1 Behaviour Cells with Scores for Selected Modeling Formalisms

<table>
<thead>
<tr>
<th>Behaviour Cells</th>
<th>GSPN</th>
<th>LQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Fork</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Join</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Merge</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Decision Cell with fixed probability (branching)</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Decision Cell with static problem data (branching)</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Decision Cell with dynamic performance data (branching)</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Decision Cell with dynamic system data (branching)</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

A performance model formalism is selected to model behaviour cells according to Table 4-1. It is very straightforward that the formalism with the higher score is selected for the behavior cell. For instance LQN is selected to model queue cell and GSPN is selected to model decision cell with problem data and one with performance data. Thus for a given system described in Figure 4-1, LQN is selected for behaviour cells as queue cells $Q_1$ and $Q_2$ and GSPN is selected for decision cell $D_1$. If both of GSPN and LQN have the medium scores, both of them are optional. If this turns to be case, the selection of the formalism for the neighbor behaviour cell can impact the selection of formalism for this behaviour cell. The selected formalism as LQN for queue cells $Q_1$ and $Q_2$ impacts the selection of formalism for neighbor behaviour cells like responsibility cells $R_0, R_1, R_2, R_3, \ldots, R_8,$ and $R_9$. Thus LQN is also selected for responsibility cells $R_0, R_1, R_2, R_3, \ldots, R_8,$ and $R_9$. Thus
the scores of each behaviour and the selected formalisms for the system in Figure 4-1 are listed in Table 4-2.

Based on the selected modeling formalism, a given behaviour cell is in focus by the selected formalism and out of focus by the other formalisms. For instance the behaviour cell, decision cell $D1$, is in focus by GSPN formalism and GSPN sub-model and out of focus by LQN formalism and LQN sub-model. The behaviour cell is modeled as the explicit representation in a sub-model in the selected formalism. It is also modeled as an abstract representation in other sub-models in the unselected formalisms. The abstract representation could be the surrogate delay for a set of behaviour cells, the average value of the visit counts for a behaviour cell, or the probability of the logical decision.

<table>
<thead>
<tr>
<th>Behaviour Cells</th>
<th>GSPN</th>
<th>LQN</th>
<th>Selected Formalism</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R0$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$R1$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$R2$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$R3$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$R4$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$R5$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$R6$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$R7$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$R8$</td>
<td>Medium</td>
<td>Medium</td>
<td>LQN</td>
</tr>
<tr>
<td>$Q1$</td>
<td>Low</td>
<td>High</td>
<td>LQN</td>
</tr>
<tr>
<td>$Q2$</td>
<td>Low</td>
<td>High</td>
<td>LQN</td>
</tr>
<tr>
<td>$D1$</td>
<td>High</td>
<td>Low</td>
<td>GSPN</td>
</tr>
</tbody>
</table>
4.3 Sub-Model Construction

Based on the behaviour cells and the selected model formalisms from the behaviour partitioning, the sub-scenario construction from the behaviour cells is described, and then the performance sub-model construction from each sub-scenario.

4.3.1 Behaviour Sub-Scenario Construction from Behaviour Cells

Given a scenario $S$ for a system to be modeled, the behaviour $B$ of this system can be partitioned into $M$ behaviour cells as described in section 4.2. $BC_1, BC_2, \ldots, BC_M$ are used to represent all these behaviour cells. $BC_i (i = 1, \ldots, M)$ will be treated as a behaviour set with one element. Thus, we can write the overall behaviour as the set $B$: $\bigcup_i BC_i = B$, $(i = 1, \ldots, M)$. One formalism is assigned to each behaviour cell according to the ranks of the modeling formalisms in Table 4-1. In total, there are $N$ model formalisms in behaviour analysis with a model formalism selection, denoted as $F_1, F_2, \ldots, F_N$.

The behaviour cells are merged and abstracted to derive sub-scenarios for all selected model formalisms. There is a set of $L$ behaviour cells $\{BC_{j1}, BC_{j2}, \ldots, BC_{jL}\}$ for a given model formalism $F_j$, $(j = 1, \ldots, N)$. The remaining $M - L$ behaviour cells are the complement of $\{BC_{j1}, BC_{j2}, \ldots, BC_{jL}\}$ in the behaviour $B$. The $L$ behaviour cells are in focus by the model formalism $F_j$. The remaining $M - L$ behaviour cells are out of focus by the model formalism $F_j$. The $L$ behaviour cells are aggregated as a behaviour fragment $BF_j$. Thus the remaining $M - L$ behaviour cells are aggregated as another behaviour fragment $\overline{BF_j}$. Both $BF_j$ and $\overline{BF_j}$ should be modeled in the formalism $F_j$. $\overline{BF_j}$ is abstracted as $\overline{BF_j}^A$. $\overline{BF_j}^A$ could be single behaviour cell and be modeled in formalism $F_j$. A sub-scenario $S_j$ is
derived from the union behaviour \( B_j = BF_j \cup BF_j^A \). The process of merging and abstracting for a model formalism is repeated to each formalism. Thus a set of sub-scenarios \( S_j, (j = 1, ..., N) \) can be obtained and one sub-scenario \( S_j \) is for each formalism.

Consistency between the sub-scenarios is a result of the sub-scenario construction. Any given behavior cell \( BC_i \) appears in \( S_j \) for formalism \( F_j \) either as part of behaviour fragment \( BF_j \) or is abstracted as part of \( BF_j^A \). It also appears in \( S_k \) for formalism \( F_k \), either in \( BF_k \) or \( BF_k^A \). The quantitative parameters and performance attributes of the behavior cell \( BC_i \) representation in both derived sub-scenarios \( S_j \) and \( S_k \) also must agree as to units and dimension. These aspects of consistency are maintained during performance sub-model construction from the scenario. More detailed discussion will be included in section 4.4.

4.3.2 Behaviour Sub-Scenario Construction of an Example System

The scenario of an example system is described, shown in a UCM in Figure 4-2 (A). This system involving clients, a web server and a database server is used to illustrate the steps from behaviour partition to sub-scenario construction. In this system, each customer sends a request to the web server for browsing web pages. The request keeps along the main path \( MP \) and conducts \( WebRead \) and \( WebWrite \) operations in sequence on the web server indicated as \( WebComponent \). The web server sends requests to the database server. If the request can acquire the write lock, it continues along \( MP \) and conducts operation \( DbWrite \). Otherwise the request goes along the branching path \( BP1 \) and operation \( DbFail \) is conducted. Both \( DbWrite \) and \( DbFail \) operations are executed on the database server indicated as \( DBComponent \).

The system is partitioned and behaviour cells are identified in HPMM. Each element in the UCM is identified as a behaviour cells. Thus there are behavior cells, \( BC_1, BC_3, BC_4, BC_7, \) and \( BC_8 \), and they are shown in Figure 4-2 (B). These behaviour cells are for
responsibilities *Thinking, WebRead, WebWrite, DbWrite* and *DbFail*. The queueing behaviour cells are identified as *BC*₂, and *BC*₅ for the queue of *WebComponent* and the queue of *DbComponent* respectively. On the database server, there is a decision, *D*, representing the performance data like the availability of the writing lock. This decision, *D*, leads to different paths for the requests based on whether the lock is available or not, which is the state of the system. The behaviour of the decision *D* is identified as a behaviour cell *BC*₆. Therefore the behaviour of the system is divided into ten behaviour cells. Figure 4-2 (A) shows all ten behaviour cells. There is \( B = \bigcup_{i} BC_i \). According to the scores of each behaviour cell defined in Table 4-1, GSPN is used to model the decision cells with dynamic performance data. LQN is deployed to model the queue cells. Thus the behaviour cell, the decision with performance data like *BC*₆, is in focus by the GSPN. The behaviour cells, the queue cells like *BC*₂, and *BC*₅, are in focus by LQN. Considering the scalability of the sub-model solution, LQN is selected for the responsibilities like *BC*₁, *BC*₃, *BC*₄, *BC*₇, and *BC*₈.
A behaviour fragment is constructed for each selected performance modeling formalism. Thus $BC_1$, $BC_2$, $BC_3$, $BC_4$, $BC_5$, $BC_7$, and $BC_8$ are aggregated as the behaviour fragment $BF_1$. $BC_6$ is the behaviour cell for $BF_2$. There are $BF_1 = BC_1 \cup BC_2 \cup BC_3 \cup BC_4 \cup BC_5 \cup BC_7 \cup BC_8$ and $BF_2 = BC_6$. The aggregation is illustrated in Figure 4-2 (C). $BC_6$ is highlighted for $BF_2$ and the remaining behaviour cells are for $BF_1$. 

FIGURE 4-2 BEHAVIOUR PARTITIONS AND BEHAVIOUR CELL COMPOSITION
A sub-scenario is constructed for each formalism from its in-focus behaviour fragment and the approximation of the other behaviour fragment. Sub-scenario $S_1$ and $S_2$ are derived from behaviour fragments $BF_1$ and $BF_2$ and their abstractions. In this system, there is $B = BF_1 \cup BF_2$. Thus, $BF_1 = BF_2$ and $BF_i = BF_2$ are derived. Based on the decision results of $D$, the behaviour cells $BC_6$ can be abstracted as a probability like $p$, which is used as the abstraction of the behaviour fragment $BF_2$, represented as $BF_2^A$. Sub-scenario $S_1$ is derived from the union behaviour $B_1 = BF_1 \cup BF_2^A$, which is also represented as $B_1 = BF_1 \cup BF_2^A$. The sub-scenario $S_1$ derived from the union behaviour $B_1$ is shown in Figure 4-2 (D). The sub-scenario $S_2$ can be also derived from the union behaviour $B_2 = BF_2 \cup BF_1^A$ following the same sub-scenario construction.

4.3.3 Performance Sub-Model Construction from Behaviour Sub-Scenario

For each sub-scenario $S_j$ with associated performance model formalism $F_j$, a sub-model $M_j$ is developed in formalism $F_j$. As discussed in section 4.3.1, the sub-scenario $S_j$ is derived from the union behaviour $B_j = BF_j \cup BF_j^A$. $BF_j^A$ could be presented as elements like an aggregated delay, a probability, or a load in the sub-scenario $S_j$. Following the regular process of performance model construction in formalism $F_j$ like [Petriu02b], the sub-model $M_j$ can be constructed from the sub-scenario $S_j$. Section 2.5 has discussed the transformation from a scenario model to an LQN model and a GSPN model, which can be used as sub-model construction here.

The process to obtain a sub-model from the initial scenario is summarized in Figure 4-3. Figure 4-3 shows the process from the behaviour partition to sub-model construction involving out of focus behaviour approximation and behaviour fragments union to sub-scenario.
For the given sample discussed in section 4.3.2, sub-models are constructed from sub-scenarios. Thus sub-model $M_1$ is developed from scenario $S_1$ in LQN and $M_2$ is from $S_2$ in GSPN.

4.3.4 Interaction among Performance Sub-Models

It is mentioned in section 4.3.1 that consistency is provided by the sub-scenario construction of HPMM, because each behaviour cell occurs in both sub-scenarios, either as it was defined or in an abstraction. It is essential that where cells are abstracted, the abstractions should have performance attributes which are consistent in units etc. with the original cells. When the sub-model $M_j$ is developed from sub-scenario $S_j$ in formalism $F_j$, the consistency feature is inherited. It may be that the abstracted cells have parameters that depend on the system performance; these parameters will be initialized to arbitrary (but reasonable values) and then calculated iteratively from the model solution. This introduces another feature of model construction, the identification of mutual dependency between sub-models.
Mutual dependency is a feature in HPMM. Given the behaviour fragment $BF_j$ abstracted in sub-model $M_j$, the parameters of abstract elements for $BF_j^A$ can be calculated from other sub-models as $\bigcup_{i \neq j} M_i$. Thus sub-model $M_j$ depends on other sub-models $\bigcup_{i \neq j} M_i$. Thus the mutual dependency can be established if multiple sub-models are involved in HPMM and the calculation of abstraction is derived from other models. This idea can be also extended to support the parameter of a sub-model depends on a solution of its own sub-model in this process.

Dependency among sub-models affects the solution of the hybrid performance model. If only one way dependency exists, the sub-models that export the data are solved first, then the rest of the sub-models that import the data are solved. This is the hierarchical solution. If there is the mutual dependency, the iteration algorithm has to be deployed in the solving tools. This research is focused on mutual dependency in HPMM.

4.4 Consistent Construction of a Hybrid Performance Model

The sub-models in HPMM are made consistent by construction.

In ad hoc hybrid modeling it is easy to construct inconsistent sub-models and it may be difficult to check the consistency. Consistency in construction is better, and is provided in HPMM, as described above. Part of it is the consistency of quantitative attributes of sub-models, to support data exchange between them during the solution. Formalizing the consistency is the most important missing piece in hybrid modeling approach. It is especially important in models where the sub-model boundary do not correspond the boundaries of well-defined modules.

Consider the parameters of sub-models. The parameters should agree with the real world as to units and dimensions. Typical parameters are times for the delay or execution demand,
rates or throughputs, and dimensionless for probability and for the number of tokens (like populations, states, and availabilities). For model construction from in-focus behaviour cells the derivation of correct parameter values is part of modeling in general and will not be considered here. However the properties and attributes of abstracted out-of-focus behaviour fragments are more of a problem. The performance quantities of the abstracted elements should be consistent. The quantities of these elements in a model are to be derived from other sub-model result. This can be solved through coordinated definition and it is specifically for the hybrid modeling solution. This section is focused on the modeling abstraction of behaviour fragments for the consistency.

4.4.1 Consistency among Sub-models

During the sub-model construction, the common origin of the in-focus behaviour fragment in one formalism and the corresponding out of focus fragment in another formalism are the basis for creating consistent sub-models. The consistent data definition of out of focus part is identified from the data exchange between sub-models and it must be correctly presented in each sub-model. This data definition approaches to being more accurate as the sub-models are solved iteratively. The consistency is considered under the following three headings.

- **Structural consistency.** Each sub-model covers all aspects of the computer system behaviour. The behaviour fragment has the explicit representation or the abstract representation in sub-models. The boundary between the explicit representation and the abstract representation in a sub-model is consistent with the partition between in focus behaviour fragment and out of focus behaviour fragment in a scenario model. The interface points on the path where they cross from one behaviour fragment to the other one are consistent with the performance quantities defined in the corresponding sub-models.

- **Behaviour consistency.** Behaviour attributes such as operations, paths, executions, and resource occupation, represented in different sub-models should be consistent with the behaviour. It means the behaviour attributes such as the execution demand of
operations, the average counts of the operations per execution, and the probabilities for the different paths should be consistently defined in different sub-models.

- **Solution consistency.** In the fixed-point algorithm, HPMM provides the parameters for the next iteration, which are calculated from the results of all sub-models in this iteration. The consistency is kept from the iteration to the next one in hybrid performance modeling solution.

### 4.4.2 Structural Consistency

The boundary between the in focus behaviour fragment and the out of focus behaviour fragments is automatically consistent with sub-model construction developed from behaviour fragments. After the scenario is partitioned, $BF_i$ and $\overline{BF}_i$ are presented in the explicit representation and the abstract representation respectively in sub-scenario $S_i$. The boundary between $BF_i$ and $\overline{BF}_i$ is kept between $BF_i$ and $\overline{BF}_i^A$ as well as the explicit representation and the abstract representation in $S_i$. After a performance sub-model $M_i$ is constructed or transformed from the sub-scenario $S_i$, the boundary is maintained to separate elements of in focus behaviour from elements of out of focus behaviour in the performance sub-model.

The interface points of out of focus behaviour fragments are consistent with the performance quantities defined for sub-models. In a UCM, the paths connect behaviour cells and indicate the sequence from the start point to the end point, and every path in the overall scenario $S$ has a corresponding path in sub-scenario $S_i$. The point on a path where it crosses from one behaviour fragment to another will be called an interface point. Consider a sub-scenario $S_i$, with behaviour fragments $BF_i$ and $\overline{BF}_i$ (abstracted as $\overline{BF}_i^A$) and an interface point between behaviour fragments $BF_i$ and $\overline{BF}_i$ in the original scenario. The semantics of passing an interface point of $\overline{BF}_i$ is the same as for passing the corresponding
point of $BF_i^A$. For example, the probability and throughput in $BF_i^A$ have the same
definition of the events of entry or exit as in $BF_i$. Also the delay between entering and
leaving $BF_i^A$ can be identified with the residence time delay in $BF_i$. The mean populations
of execution tokens and busy resources are the same also. If there are multiple entry and
exit points, then the exit probability, delays, resource uses and populations correspond for
each entry/exit pair.

4.4.3 Behaviour Consistency

Behaviour consistency is automatic also in each performance model definition. The entry
and exit pairs in $BF_i^A$ are consistent with the pairs in $BF_i$ in each sub-scenario $S_i$. Thus
the calculations of the performance quantities for $BF_i^A$ in a performance sub-model are
consistent abstractions for the calculation of the same quantities for $BF_i$. This is termed as
behaviour consistency.

Behaviour consistency is applied in hybrid performance model construction through
performance quantity definition. In general, performance quantities are delays, throughputs,
populations in a part of the system or behaviour, resource usages, visit counts, and
probabilities. When a performance sub-model $M_i$ is developed from the sub-scenario $S_i$,
some parameters of $BF_i^A$ that define parameters in $M_i$ are to be calculated from other sub-
models $M_j (j \neq i)$. Ideally the performance quantity parameter of an element in $M_i$ like
delay and visit counts includes the distribution that the element corresponds. But the mean
value of the element is used in hybrid performance model construction.
4.5 Consistent Parameters for Abstract Scenario Fragments

To provide behavioral consistency the parameters of out-of-focus abstract fragments must be derived systematically from those of its constituent cells, or from performance attributes of the system. The common parameters of out of focus behaviour in a performance sub-model of the hybrid model solution are:

1. Surrogate Delays: Surrogate delays are used to represent the mean response time of a set of the behaviour cells.
2. Probabilities: The probabilities for different paths are deployed to represent the abstract results of the complicate logical decision.
3. Average Visit Counts: The average number of visits is counted during one execution. It represents an approximation of accessing to a behaviour cell in a performance model.
4. Surrogate Customers: Surrogate customers are estimated as the number of the active instances of requests (or jobs). They indicate the load of the contention to resources like processes, threads, requests and etc..

The following notation will be used throughout this remaining section:

\[ N \] = the population of customers in the system

\[ C(N) \] = the cycle time, the time for a customer token to cycle through one response and one delay (think time) outside the system.

\[ f_p(N) \] = the throughput of the path \( p \)

\[ P_p(N) \] = the probability of going through the path \( p \)

\[ V_R(N) \] = the mean number of visits of the operation(s) per each execution (frequency)

\[ S_R \] = the service time of the operation \( R \)

\[ W_R(N) \] = the waiting time (queueing time and service time) of the operation \( R \)
4.5.1 Surrogate Delays

A surrogate delay is a single delay which represents the sum of time spent in some subsystem or state. Jacobson and Lazowska developed the method of surrogate delays for approximate solution of queueing models, in [Jacobson81]. The two sub-nets are constructed from a system and the solver iterates between them. Each sub-model includes an explicit representation of part of the behaviour and a delay server representing the rest of the behaviour. The method of surrogates was generalized by Silva and Muntz [Silva87] for non product-form QNs to handle resources by customer chains. Surrogate delays are also employed in HPMM. The surrogate delay may be a service time or a waiting time computed in different sub-models.

Suppose there is a scenario with $N$ customers in a closed network. The scenario is modeled through a Use Case Map in Figure 4-4 (A) with the path $p$. Each customer performs the
operations (responsibilities) \( R_1, R_2, R_3 \) and \( R_4 \) sequentially along the path \( \rho \). The behaviour in the scenario is partitioned by the dashed line in Figure 4-4 (A). The performance model formalism \( F_1 \) is used to model behaviour cell \( R_1 \) and \( R_4 \) in the explicit representation with behaviour cell \( R_2 \) and \( R_3 \) in the abstract representation in sub-model \( M_1 \). The formalism \( F_2 \) models behaviour cell \( R_2 \) and \( R_3 \) through the explicit representation and behaviour cell \( R_1 \) and \( R_4 \) in the abstract representation in sub-model \( M_2 \). Thus \( R_1 \) and \( R_4 \) are in focus by \( M_1 \) but \( R_2 \) and \( R_3 \) are out of focus by \( M_1 \). The notation of a surrogate delay is introduced in Figure 4-4 and is presented as a cross with a square. In Figure 4-4 (B) behaviour cells \( R_2 \) and \( R_3 \) are abstracted as a surrogate delay \( SD_{23} \) defined in sub-model \( M_1 \). The same modeling approach is also applied for \( M_2 \) but \( R_2 \) and \( R_3 \) are in focus and \( R_1 \) and \( R_4 \) are out of focus by sub-model \( M_2 \) (see Figure 4-4 (C)).

In the sub-model \( M_2 \), either the service time or the waiting time for both operations \( R_2 \) and \( R_3 \) are computed so that results can be used in sub-model \( M_1 \). If only the service time of both operations \( R_2 \) and \( R_3 \) is derived from \( M_2 \), the resource contention for the surrogate delay \( SD_{23} \) should be modeled in sub-model \( M_1 \). If the waiting time is obtained, the resource contention is ignored and the infinite processors with the pure delay are defined.

The data exported by \( M_2 \) to \( M_1 \) is expressed in the direct computation through Equ. 4-1 or Equ. 4-2:

\[
S_{SD_{23}} = \text{the service time of both operations } R_2 \text{ and } R_3 \text{ from calculation} \quad \text{Eqn. 4-1}
\]

or

\[
W_{SD_{23}}(N) = \text{the waiting time of both operations } R_2 \text{ and } R_3 \text{ from calculation} \quad \text{Eqn. 4-2}
\]

If the operations are not adjacent and are scattered in the scenario, there is a shortcut way to calculate the waiting time or the service time of them together. The complementary computation is used to calculate the service time or the waiting time for each one separately. The complementary computation needs the cycle time. For instance, in figure 4-4 (A),
operations $R1$ and $R4$ are physically separated. It is assumed that there are a number active customers $N$ along the path $p$ and the throughput of the path is $f_p(N)$. Thus the cycle time is defined as Equ. 4-3:

$$C(N) = \frac{N}{f_p(N)}$$  \hspace{1cm} \text{Equ. 4-3}$$

The data imported by $M_2$ from $M_1$ is calculated through the following Equ. 4-4:

$$W_{SD14}(N) = C(N) - W_{SD23}(N)$$  \hspace{1cm} \text{Equ. 4-4}$$

Therefore the abstraction, surrogate delays $SD23$ or $SD14$, can be calculated through the direct computation or the complementary computation. The data exchange in hybrid model solution can be set up between sub-models $M_1$ and $M_2$.

4.5.2 Probabilities and Average Visit Counts for the Specific Operations (Surrogate flows)

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4-5.png}
\caption{Probability Definition}
\end{figure}

Considering a scenario, the decision behaviour cell governs the behaviour of the abort model among exception handling models. As shown in Figure 4-5 (A), the scenario model contains...
a stub, which represents the decision behaviour cell. There is a path \( p \) in the scenario model and the stub leads to two paths: \( p1 \) and \( p2 \). There is a set of operations/responsibility: \( R1, R2, R3, R4, R5 \) and \( R6 \). The behaviour cells can be partitioned into two behaviour fragments: one is for the stub with the decision behaviour cell and the other is for the rest of the behaviour. Sub-model \( M_1 \), which models Figure 4-5 (B), focuses on the decision behaviour cell in the stub. The rest of the behaviour cells are out of focus by the sub-model \( M_1 \) and are abstracted as the surrogate delays: \( SD12, SD34, \) and \( SD56 \) for the abstract representation. Sub-model \( M_2 \) modeling Figure 4-5 (C) covers the rest of the behaviour through an explicit representation but models the decision behaviour cell by the probabilities for each exit path of the decision selection: \( p1 \) and \( p2 \) as the abstract representation. Thus the sub-model \( M_2 \) imports the data from the sub-model \( M_1 \) for completing the model definition. In the sub-model \( M_1 \) there are the throughputs \( f_p(N), f_{p1}(N) \) and \( f_{p2}(N) \), which represent the throughputs for the paths \( p, p1 \) and \( p2 \) respectively. The data imported by \( M_2 \) from \( M_1 \) is defined as the following equations:

\[
P_{p1}(N) = \frac{f_{p1}(N)}{f_p(N)} \quad \text{Equ. 4-5}
\]

\[
P_{p2}(N) = \frac{f_{p2}(N)}{f_p(N)} \quad \text{Equ. 4-6}
\]

The data imported by \( M_1 \) from \( M_2 \) is defined as surrogate delays, which are already discussed, see Equ. 4-1, 4-2, and 4-4.
Considering another scenario, there is a decision behaviour cell that governs the behaviour of the infinite retry model among exception handling models. The scenario model is described in Figure 4-6 (A) and a stub also represents the decision behaviour cell but leads the retry behaviour. There is a path $p$ and the stub leads to the retry path, $p1$. There is a set of operations/responsibilities: $R1$, $R2$, $R3$, $R4$, $R5$ and $R6$. The similar partition is applied: one is the behaviour fragment for the decision behaviour stub and the other is the behaviour fragment for the rest of the behaviour. The sub-model $M_1$ that models Figure 4-6 (B) focuses on the decision behaviour fragment. Other behaviours are out of focus by the sub-model $M_1$ and can be modeled as the surrogate delays: $SD12$, $SD34$, and $SD56$ with the approximation in the sub-model $M_1$. The sub-model $M_2$ for Figure 4-6 (C) describes the rest behaviour through the explicit representation but model the logical behaviour as the average number of visit counts for the operations through the abstract representation: $R5$ and $R6$. Thus the sub-model $M_2$ imports the data from the sub-model $M_1$ for the model definition. The throughputs $f_p(N)$ and $f_{p1}(N)$ for the paths $p$ and $p1$ can be obtained in sub-model $M_1$. The data exported by $M_1$ to $M_2$ as the following equation:

$$V_R(N) = f_{p1}(N) / f_p(N)$$

Equ. 4-7
The data imported by $M_1$ from $M_2$ is defined as surrogate delays, which have been already discussed. Surrogate delays are omitted here and they can be referred to Equ. 4-1, Equ. 4-2, or Equ. 4-4.

4.5.3 Surrogate Customers

The number of customer token in a sub-model is important and should present the maximum feasible level of contention, which may be limited by parameters. For example, in models in Figure 4-5 (B) and Figure 4-6 (B) all customers in the computer system are treated as active instances to compete for the logical cell. In MVA sub-model construction for LQNS (section 2.2.3 in [Franks00]), different customer derivation policies such as a single server, a multi-server and an infinite server are applied for estimating the number of customers in each sub-model derived from a LQN model. Following this idea, a similar approach is proposed for as surrogate customers to represent the resource contention accurately.

![Figure 4-7 Surrogate Customer Definition](image-url)
Considering a scenario for a three-tier web application, there are clients, a web server and a database server. The clients send requests to the web server after thinking time represented in Figure 4-7 as \( R0 \). There are ten threads on the web server and execute the responsibility \( R1 \) and then forward requests to the database server. There are five threads on the database server and they perform the database operations. The database operations include the query operation shown as \( R2 \) and the update operation shown as \( R3 \) and \( R4 \) in Figure 4-7. However, there is only one thread, which is allowed to execute the update operation. Therefore, there is a lock, which governs the update operation. If the thread on the database server can get the lock, the request goes along the path \( p1 \). Otherwise, the request goes along the path \( p2 \). The abort model among exception handling models is applied to the lock and the logical behaviour cell for the lock governs the behaviour of the system.

As shown in Figure 4-7 (A), the scenario model contains a stub, which represents the logical behaviour cell. There is a path \( p \) and the stub leads to the retry path, \( p1 \). There is a set of operations: \( R0, R1, R2, R3, R4, R5 \) and \( R6 \). The behaviour can also be partitioned into two behaviour fragments: one is for the stub with the logical behaviour and the other is for the rest of the behaviour. As section 4.4.3.2, sub-model \( M1 \) focusing on the logical behaviour set and sub-model \( M2 \) focusing on the remaining behaviour can be developed. Therefore, sub-model \( M1 \) is developed based on Figure 4-7 (B). The scenario fragment for sub-model \( M2 \) is omitted since it is similar to Figure 4-5 (C) and the same abstraction, probability, is also applied. It is easy to collect the delay for the aggregation of responsibilities \( R3 \) and \( R4 \) and the delay for the aggregation of responsibilities \( R5 \) and \( R6 \) in sub-model \( M2 \). It is assumed that the mean delay for the update operation can be calculated from the sub-model \( M2 \) and is represented as \( \text{DEL}_{\text{update}} \) so that the delay for the aggregation of responsibilities \( R0, R1, \) and \( R2 \) can be derived. As discussed in section 4.4.3.1, the cycle time of each request is used for the calculation of the rate for surrogate delay \( SD012 \) and is defined as Equ. 4-8:

\[
C(N) = \frac{N}{f_p(N)} \quad \text{Equ. 4-8}
\]

The data imported by \( M1 \) from \( M2 \) is calculated through the following Equ. 4-9:
Following Table 4-1, GSPN is applied to sub-model $M_1$. In sub-model $M_1$, there are $N$ customers competing for the lock resource after surrogate delay $SD012$ in the modeling perspective. However there are only five threads on the database server to compete for the lock resource in the real system. If $N$ (number of clients) is greater than five (number of threads), the contention for the lock resource is overestimated. Also if $N$ is a large number, the solver for sub-model $M_1$ takes long time to solve it and occupy a lot of memory space.

To accurately describe the competition, the five threads on the database server can be treated as surrogate customers and they are the active instances for the resource contention. The sub-model $M_1'$ with five surrogate customers will replace sub-model $M_1$. Therefore the sub-model $M_1'$ can avoid the overestimation problem. On the other hand, the solving time and the state space for sub-model $M_1'$ can be decreased compared to the sub-model $M_1$ since the sub-model $M_1'$ only models five threads as surrogate customers. The sub-model $M_1'$ should import the throughput of the threads on the database server $f_p'(N)$ and the mean delay of the update operation $DEL_{update}$ from sub-model $M_2$. The number, $N_{sc}$, is used to represent the number of surrogate customers (here is the number of threads). The cycle time for each thread (surrogate customer) on the database server is defined as Equ. 4-10:

\[
C_{sc}(N) = \frac{N_{sc}}{f_p'(N)} \quad \text{Equ. 4-10}
\]

The data imported by $M_1'$ from $M_2$ is calculated through the following Equ. 4-11:

\[
W_{SD012}(N) = C_{sc}(N) - DEL_{update} \quad \text{Equ. 4-11}
\]
4.5.4 Parameters Notations in Data Exchange

In the above sections, all calculations are based on customers, throughputs, delays, and derived probabilities. Those calculations are used to present the data exchange when a hybrid performance model is defined. To facilitate the data exchange definition, some parameter notations are discussed here.

Parameter notations here are limited on LQN and GSPN formalisms and focused on the elements like loads, entries in LQN, and timed transitions, immediate transactions, and places in GSPN. There are two types of parameter notations in the data exchange. One is the parameter and the other is the result. The value of a given parameter notation must be specified for the selected formalism if this notation is parameter type. For instance, the population of customers in a closed model should be defined in both the LQN model and the GSPN model. If the notation is result type, it means the value of the notation is derived from the model result. The service time of an entry is obtained from the LQN model results. Therefore, the parameter notations used in this thesis are listed in Table 4-3.
### Table 4-3 Parameter Notations in Data Exchange

<table>
<thead>
<tr>
<th>Notations</th>
<th>Formalisms</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>LQN/GSPN</td>
<td>Parameter</td>
<td>Population of customers in a closed model</td>
</tr>
<tr>
<td>f</td>
<td>LQN/GSPN</td>
<td>Parameter</td>
<td>Load throughput of a system (Requests/TimeUnit)</td>
</tr>
<tr>
<td>ServiceTime(entry)</td>
<td>LQN</td>
<td>Result</td>
<td>Service time collected through an entry (TimeUnit)</td>
</tr>
<tr>
<td>Throughput(task)</td>
<td>LQN</td>
<td>Result</td>
<td>Throughput of a task</td>
</tr>
<tr>
<td>Throughput(entry)</td>
<td>LQN</td>
<td>Result</td>
<td>Throughput of an entry</td>
</tr>
<tr>
<td>Demand(entry)</td>
<td>LQN</td>
<td>Parameter</td>
<td>Service demand of an entry</td>
</tr>
<tr>
<td>Call(entry1, entry2)</td>
<td>LQN</td>
<td>Parameter</td>
<td>Number of calls from entry1 to entry2</td>
</tr>
<tr>
<td>$\mu$(timedtransition)</td>
<td>GSPN</td>
<td>Parameter</td>
<td>Rate for timed transition (Per TimeUnit)</td>
</tr>
<tr>
<td>Mark(place)</td>
<td>GSPN</td>
<td>Parameter</td>
<td>Marking number of a place</td>
</tr>
<tr>
<td>Prob(immediatetransition)</td>
<td>GSPN</td>
<td>Parameter</td>
<td>Probability to fire an immediate transition</td>
</tr>
<tr>
<td>f(immediatetransition)</td>
<td>GSPN</td>
<td>Result</td>
<td>Rate for an immediate Transition (Request/TimeUnit)</td>
</tr>
</tbody>
</table>

#### 4.6 Solution Consistency

The third aspect of consistency is forcing the solutions of the two sub-models to be consistent with each other and with the performance of the system when the iterative...
solution is converged. Thus the consistency is enforced by constraining the parameters of the abstract out-of-focus fragments.

For the sub-model $M_i$, the parameters of the abstract out-of-focus fragment that are not determined directly from its constituent cells form a parameter vector $X_i$ and the performance results can be gathered into an output vector $Y_i$, which approximates the system performance results vector $Y$. Hybrid modeling exploits relationships between $X_i$ and $Y$, some of which are defined in the previous section. That is, it uses relationships between a parameter in $X_i$ such as a set of branch probabilities, and the resulting flow rates in the branches, which are part of $Y$, as defined in section 4.5.2. This is a kind of consistency condition for $X_i$. In practice we must use elements of $Y_i$ or $Y_j$ in place of $Y$ which is unknown. If all the elements of $X_i$ satisfy the necessary relationships with $Y_i$, and the elements of $X_j$ satisfy the necessary relationships with $Y_j$, then we will say that $M_i$ and $M_j$ have "solution consistency".

The necessary relationships are imposed through a fixed point iteration. A functional notation is introduced for sub-models in the iterative solution. As before, $i$ is used as an index of sub-models and there are $N$ sub-models. $j$ is used as an index of iterations and after $M$ iterations, the solution is converged. $i$ is in the range from 1 to $N$ and $j$ is in the range from 1 to $M$. For a given iteration $j$ and sub-model $M_i$, here is the functional notation listed here.

$X_{ji}$: the parameter vector of the sub-model $M_i$ including the parameter of $BF_i$ in the iteration $j$.

$Y_{ji}$: the performance measures of the sub-model $M_i$ in the iteration $j$ and performance measures are used in other sub-models $M_k$, $(k \neq i)$. 

72
$Y_{ji} = SM\{X_{ji}\}$: the performance solution of $M_i$ in the iteration $j$. Here solving a sub-model is treated as a function. Parameter vector is independent variable and performance measures are the dependent variable.

$X_{ji} = F_i\{U\{Y_{(j-1)i}\}, (k \neq i)\}$: the calculation of the parameter vector for sub-model $M_i$ in the iteration $j$, given the solution of all the other sub-models $M_k, (k \neq i)$.

Following the discussion in behaviour consistency, the construction of the function set $F_i$ for the sub-model $M_i$ is defined for data exchange. Generally, the function set $F_i$ includes a set of functions to derive the performance quantities used parameters of sub-model $M_i$. There are four performance quantities involved in the hybrid solution: surrogate delay, throughput/load, probability and mean population. These performance quantities can be calculated through the function from the result of the other sub-models $M_k, (k \neq i)$. The result should include delay, throughput, and population.

Little's theorem simply indicating the relation between throughput, population and delay is deployed in function development. The Little's theorem can be used to calculate throughput, population and delay according to different scenarios.

The development of functions for data exchange is summarized through four kinds of performance quantities.

- **Surrogate Delay** could be calculated as an identity function on the cross-path delay in a sub-model $M_k, (k \neq i)$. This identity function could add up the individual delay with weight. Surrogate delay can be also calculated through Little's formula with the solution population and throughput.

- **Throughput/load** can be obtained directly from the derived throughput of the other sub-model $M_k, (k \neq i)$. It can be also derived through the Little's formula with the solution population and derived delays.
- Probability can be computed through the ratio of the solution throughputs in the other sub-model.
- Mean population is given by the Little's formula with solution delay and throughput.

4.7 Hybrid Performance Model Solution and Tools

A hybrid performance model consists of multiple separate models in different model formalisms, which are solved iteratively. The solvers are coordinated through the hybrid solution controller, which retrieves performance measures from each solver and provides the input parameters for other models. The hybrid solution controller implements the iteration algorithm and monitors the convergence. This chapter introduces the generic structure of the hybrid performance model solver. Two hybrid performance model solvers are also described: the first tool combining the LQN solver and the SPNP Petri Net Solver, and the second one, an open hybrid performance model solution supporting LQNS and SPNP.

4.7.1 Hybrid Performance Modeling Framework

![Figure 4-8 Hybrid Performance Modeling Framework with the Fixed-Point Iteration](image)
The framework supports both constructing and solving a hybrid performance model. The hybrid performance modeling framework enables users to define the sub-models for a computer system in different model formalisms. The abstractions are represented in each sub-model. Normally the variables are defined for the abstract elements in a sub-model so that the values of them are specified. The hybrid performance modeling framework allows users to develop the equations that indicate data exchange among sub-models. Each variable in a sub-model is calculated through equations from performance measures of other sub-models.

The hybrid performance modeling framework takes advantage of existing solving tools to solve sub-models. Once performance measures are derived after solving a sub-model, the framework retrieves the interesting performance measures and provides them for data-exchange equations. The framework provides the initial guess for the value of variables in a sub-model that is computed first in the first iteration, specifies the solving sequence among sub-models, and defines the convergence criteria for the iteration algorithm.

Therefore a general hybrid performance modeling framework is defined in Figure 4-11. It comprises of a hybrid solution controller, a set of performance model solvers and a set of performance models in different model formalisms. The modelers specify the initial guess, the solving sequence, the equations for data exchange among sub-models, and the convergence criteria through the hybrid solution controller. The hybrid solution controller invokes each solver to solve the sub-model by assigning the values to the variables defined in the sub-model. The controller also retrieves the performance measures from the results of sub-models. Each sub-model is defined with variables by modelers and is solved in a sequence defined in the controller.
4.7.2 Hybrid Performance Model Solver: Combining SPNP with LQNS

The first tool combined SPNP [Trivedi98] and LQNS [Franks00][Franks95], which are well known solving tools for the model formalisms: Petri Net and LQN respectively.

As discussed in section 4.7.1, this hybrid performance model solver includes the hybrid solution controller, LQNS to solve a LQN sub-model, and SPNP to solve a Stochastic Petri Net. Figure 4-11 illustrates the sub-model solving steps in the hybrid performance model solution including the interaction among the hybrid solution controller, a LQN sub-model, a GSPN sub-model and their solvers: LQNS and SPNP. In the hybrid performance model solution, a LQN sub-model or a GSPN is defined with its parameters. The model with parameters is defined as a node. In Figure 4-12, the LQN node has input parameters $X_{LQN}$ and the GSPN node has input parameters $X_{GSPN}$. A model is instantiated through the hybrid solution controller when the parameters in a node are assigned with values. After the instantiation of a model, the model can be solved through the corresponding solver. The result of this model can be used to calculate the parameters of the other model. The algorithm can be summarized as follows:

1. Given the initial guess value for $X_{LQN}$;
2. Use $X_{LQN}$ and solve the LQN sub-model through LQNS (denoted as a solver function $f_{LQN}$);
3. Derive the result $Y_{LQN}$ ($Y_{LQN} = f_{LQN}(X_{LQN})$);
4. Use $Y_{LQN}$ to calculate the value of $X_{GSPN}$;
5. Use $X_{GSPN}$ and solve the GSPN sub-model through SPNP (denoted as a solver function $f_{GSPN}$);
6. Derive the result $Y_{GSPN}$ ($Y_{GSPN} = f_{GSPN}(X_{GSPN})$);
7. Use $Y_{GSPN}$ to calculate the value of $X_{LQN}$;
8. Calculate the delta of the value of $X_{LQN}$ between step 7 and step 2,
   
   a. if delta is less than the predefined convergence criteria, the solution is derived from $Y_{LQN}$ and $Y_{GSPN}$
   
   b. if delta is greater than the predefined convergence criteria and the number of iterations is less than Max_Iteration, move to step 2;
   
   c. if delta is greater than the predefined convergence criteria and the number of iterations is equal to Max_Iteration, stop and report the non-convergence error;

---

**Figure 4-9 A Hybrid Performance Tool Combining LQN and GSPN**

4.7.3 Hybrid Performance Model Solution: an Open Hybrid Performance Modeling Tool

To make the hybrid performance model solver be more flexible and maintainable, an open hybrid performance modeling tool was developed. In this tool, it is assumed that a performance model is defined in a file (or files) and the solver can be invoked through an
executable command. The model solving process is invoking the solver command with the parameter, a file or a set of files. In fact most performance solvers such as LQNS, SPNP, and etc. follow this assumption. Even some performance solutions may not follow this pattern and they may require multiple executables in solving models and retrieve results. These executables can be integrated through the batch file. Therefore this tool preserves independence of each performance formalism and solution and makes the integration easier.

The open hybrid performance modeling tool is designed and developed through an object-oriented programming language, JAVA. From the hybrid performance modeling procedure defined in section 3.4, it is derived that this procedure can be treated as the interaction between performance sub-models and functions. Each sub-model is instantiated with an input for variables and can be solved to generate an output. Functions take the output as input and are calculated output, which can be taken in as the input for performance sub-models. Thus performance sub-models and functions are abstracted as a HybridEntity class with input and output. An attribute, label, is added to this class to indicate which type of entity such as LQN, GSPN, or FUNCTIONS an entity is.

An abstract subclass Functions class is developed inheriting the super class HybridEntity. An abstract method calculate() is defined in Functions class. Therefore the calculations on a function set indicating data exchange from one sub-model to the other sub-model implement this method.

Another subclass, SubModel is also created to inherit from the super class HybridEntity. Generally speaking, it is required that a performance model is defined in an input file with a fixed extension in the name in a specific path. Once it is solved, the result is generated into output file with a fixed extension in the name. Thus three attributes, pathFileName, inputExt, and outExt, are added to SubModel class, which represent the path and filename of the model, the extension of the model file name, and the extension of the output file name.
For each performance modeling formalism, a subclass is created to extend \textit{SubModel} class. So far two subclasses, \textit{GspnModel}, \textit{LqnModel}, are developed to inherit from this class. GSPN formalism is integrated as \textit{GspnModel} class is created. The GSPN model solving process can be summarized as follows: A GSPN model is defined in a file with extension "\texttt{.c}" as the file type. The model can be solved through firing the command "\texttt{sp ***.c}" and the file with extension "\texttt{***.out}" is generated as an output file. Therefore in \textit{GspnModel} class, method \textit{execute()} has implemented this solving process and method \textit{retrieveResults()} retrieves the performance result for a specific iteration through parsing the output file.

LQN formalism is also integrated and \textit{LqnModel} class is added to this tool. LQN model solving process begins with a LQN model in a file with file type extension "\texttt{.lqn}" , which is converted into the LQN XML format with file type extension "\texttt{.xml}" and solved through invoking the command "\texttt{lqns ***.xml}". Finally the performance results of each element are put into the same file in XML format. So in \textit{LqnModel} class, this solving process is implemented through method \textit{execute()}. 
Table 4-4 Keys, Attributes, and Sample Expressions in LQN Result Expressions

<table>
<thead>
<tr>
<th>Keys</th>
<th>Attributes</th>
<th>Sample Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>utilization</td>
<td>processor(p_1)-&gt;utilization</td>
</tr>
<tr>
<td>task</td>
<td>phase1-utilization</td>
<td>task(t_1)-&gt;phase1-utilization</td>
</tr>
<tr>
<td></td>
<td>proc-utilization</td>
<td>task(t_2)-&gt;proc-utilization</td>
</tr>
<tr>
<td></td>
<td>throughput</td>
<td>task(t_3)-&gt;throughput</td>
</tr>
<tr>
<td></td>
<td>utilization</td>
<td>task(t_4)-&gt;utilization</td>
</tr>
<tr>
<td>entry</td>
<td>proc-utilization</td>
<td>entry(e_1)-&gt;proc-utilization</td>
</tr>
<tr>
<td></td>
<td>squared-coeff-variation</td>
<td>entry(e_2)-&gt;squared-coeff-variation</td>
</tr>
<tr>
<td></td>
<td>throughput</td>
<td>entry(e_3)-&gt;throughput</td>
</tr>
<tr>
<td></td>
<td>throughput-bound</td>
<td>entry(e_4)-&gt;throughput-bound</td>
</tr>
<tr>
<td></td>
<td>utilization</td>
<td>entry(e_5)-&gt;utilization</td>
</tr>
<tr>
<td>activity</td>
<td>proc-waiting</td>
<td>activity(a_1)-&gt;proc-waiting</td>
</tr>
<tr>
<td></td>
<td>service-time</td>
<td>activity(a_2)-&gt;service-time</td>
</tr>
<tr>
<td></td>
<td>service-time-variance</td>
<td>activity(a_3)-&gt;service-time-variance</td>
</tr>
<tr>
<td></td>
<td>utilization</td>
<td>activity(a_4)-&gt;utilization</td>
</tr>
</tbody>
</table>

To obtain the result of a LQN model, LQN result expression is invented in this tool. An LQN result expression is defined as the pair of key and attribute and it is written as `key->attribute`. Keys represent elements in an LQN model such as processor, task, entry, call, and activity with names. Different keys have different attributes. Both of them are defined in Table 4-2. Table 4-2 includes the sample expressions so that the entity name such as \( p_1 \), \( t_1 \), etc. is replaced in LQN result expression usage. Hence the class attribute, `expressions`, is
defined in \textit{LqnModel} class. The method \textit{retrieveResults()} first parses LQN result expressions so that it can figure out the key and attribute pair. Then this method retrieves the performance results for a specific iteration through LQN model in XML format based upon the key and attribute pair.

The \textit{HybridModel} class implements the hybrid performance modeling procedure defined in section 3.5. A convergence criterion is specified through attribute \textit{epsilon}. The maximum number of iterations is defined as attribute \textit{MAX\_ITERATION}. Method \textit{iteration()} iterates each element in attribute \textit{models}, invoke method \textit{execute()} if this element is \textit{SubModel} object, or method \textit{calculate()} if this element is \textit{Functions} object. If the difference of the value of input variables between two iterations in a row is less that the value of attribute \textit{epsilon}, or the number of iterations reaches the value of \textit{MAX\_ITERATION}, the iteration is terminated.

The class diagram of the open hybrid performance modeling tool Java implementation is shown in Figure 4-15. This hybrid performance model tool is defined as "open" because more similar subclasses can be added as extensions of \textit{SubModel} class so that this tool can support more formalisms. The more formalisms are integrated, the more powerful this tool is.
Figure 4-10 The Class Diagram of an Open Hybrid Performance Modeling Tool
JAVA Implementation
5. Performance Modeling In Resource Allocation Exception Handling

Inspired by modeling exception handling mechanisms discussed in section 2.1, the general problem of modeling exceptions was addressed. This chapter focuses on resource allocation exceptions. It starts with performance model construction for generic exceptions with probability propagation. If the probability is unknown, a hybrid approach may provide performance estimates. For the various exception handling mechanisms, hybrid performance modeling is applied following the steps identified in chapter 3 and chapter 4. Performance results from the hybrid model are validated against simulation results.

Resource allocation exceptions are common and include failure to allocate a node, CPU, memory, buffer, cache, process, or thread, rejection of a service request, and lock or critical section request failures. Four typical resource allocation patterns described in section 2.1 are considered: queueing pattern, aborting pattern, infinite retry pattern and finite retry pattern.

5.1 Fixed-Probability Exceptions with Probability Propagation

Before considering resource allocation, it will be helpful to consider generic exception handling with known probabilities of exception. This will describe the exception probability propagation in the calling hierarchy, which is also applicable when the exception originates in the resource allocation.

To construct the performance model for exception handling, the control flow of a computer system is analyzed. For each method, the caller, which calls this method, is named as a higher-level method. The callee, who is called by this method, is named as a lower-level method. For a recursive method, the higher-level or the lower level method can be identified as different instances of a method. In the performance analysis perspective, the propagation model, the abort model, the resumption model, and the retry model are implemented...
through different exception handler binding strategies, and different instances of the regular activity and the exceptional activity. This section focuses on the abort model, but the approach can be extended to the propagation model, the resumption model, and the retry model. The system to be modeled will be a software component which contains the exception handling mechanism, and which is assumed to be called by a user and to give a response to that user. The exception handler binding strategy of Java or C++ is applied in this system. The behaviour fragments for the GSPN and LQN model formalisms are to be constructed, and an approach to obtain the parameters of the two performance models is developed.

5.1.1 The Scenario Model of an Example

The scenario model uses sub-maps to represent the behaviour of called methods, including alternative exit points, and wherever an exception is raised it has a branch to the appropriate handler behaviour. Figure 5-1 shows an example through a conventional display of sections of code with arrows for transfer of control. Execution begins in method MethodA. The method MethodA is under study and invokes lower level methods, MethodB and MethodC. MethodB and MethodC can raise the exception sets \{e1, e2\} and \{e2, e3\} respectively. After invoking method MethodB and MethodC, MethodA may raise exception e3.

The control flow of a program moves to the exception handler once there is an exception encountered at the escape point. Figure 5-1 shows the sequence when exception e3 is raised at the second escape point in MethodC, transferring control to Handler(e3). The assumption in this section is that the probabilities of occurrence of exceptions are known.
A scenario model is defined in Figure 5-2 for all exceptions raising at all possible escape points. The escape points are numbered from 1 to N for each function, according to the sequence of the exception occurrence and provided there is more than one. If there are N escape points in a function, the function is divided into (N+1) code blocks. Each block is mapped to a responsibility in a UCM and blocks are numbered from 1 to (N+1). MethodA, MethodB and MethodC have 4, 3, 3 blocks respectively. Each responsibility has a name including method name and block index, for example Arep1, Arep2 ... for MethodA. The escape points: ep1, ep2, are in MethodB and the escape points, ep3 and ep4 are in MethodC. The escape point, ep5 is in MethodA. All escape points are shown in Figure 5-2. The method invocation is represented as a stub in a scenario model and a stub can be expanded as a sub-scenario for the lower level method execution. All the exception handlers are defined in the scenario for MethodA and responsibilities for each handler are also specified in the scenario model in Figure 5-2. In a scenario model, the OR fork is used to represent the exception handling.
handling which leads to two different paths, one for the regular activity and the other for the exceptional activity.

The expanded UCM scenario model is drawn in Figure 5-3 (A) with the stubs expanded. The path for the guarded scope with the regular activity is defined as the main path. Thus the path of the guarded scope of MethodA is the main path. The stubs along the main path in MethodA are expanded and the sub-scenarios of the lower level are plugged in the main path (see Figure 5-3 (A)). The path that leads to the exceptional activity as the exception handler is defined as the branch path. There are five branch paths along the main path.
5.1.2 Performance Model Construction for this Example

This section explains how to model exception handling with the probability propagation. For simplicity, other constraints such as resources, multiple instances, and the interaction with other components are ignored. There are five escape points: ep1, ep2, ep3, ep4, and ep5 in Figure 5-3 (A). The probabilities of raising an exception at the escape points are assumed to be known and they are $p_1, p_2, ..., p_5$ respectively. The function $\text{prob}_{rs}(ep_i) = p_i, i = 1, ..., 5$ represents the probability of raising a specific exception at an escape point. There is an assumption that only one exception can be raised at each escape point. If necessary, multiple
escape points can be created in the same location. The function $\text{except}_rs(ep) = e$ returns the exception instance $e$ at the escape point $ep$. In the example, there are $\text{except}_rs(ep_1) = e_1$, $\text{except}_rs(ep_2) = e_2$, $\text{except}_rs(ep_3) = e_2$, $\text{except}_rs(ep_4) = e_3$, and $\text{except}_rs(ep_5) = e_3$. In Figure 5-3 (A) the probability is also labeled on each branch path for each raised exception instance.

![Figure 5-4 A GSPN Model for the Example with Propagation Probability](image-url)
To model this component in a GSPN model, each responsibility is mapped to a timed transition. A branching element is transformed into one place with an immediate transition for each branch. The weight for each immediate transition is equal to the probability for each branch in the scenario model. Figure 5-4 shows a GSPN model. Each responsibility in the UCM in Figure 5-3 (A) is mapped to a timed transition with the same label in the GSPN model.

Following the procedure discussed in section 4.3.3, the LQN model for the example is obtained in Figure 5-5. MethodA has an entry for each of its responsibilities and handlers, with the same name (entry Arep1 for the responsibility Arep1 in MethodA, Ahd1 for Handler1 in MethodA, and etc.), and MethodB and MethodC likewise. A customer task is introduced with the entry HLEn which calls a set of entries representing activities for MethodA, with a call count equally to mean number of visits to the activity in one call to MethodA. This approach is used because LQN cannot represent an entry such as Arep1 calling another entry such as Arep2 (there is an activity graph capability in LQN which could be used for this purpose, giving an alternative model structure). Similarly calls from MethodA to entries of MethodB and MethodC have call counts according to the frequency of their invocation. The call counts are derived by traversing the UCM, as discussed next.

![Figure 5-5 A LQN Model for the Example with Propagation Probability](image-url)
5.1.3 An Approach to Parameterizing the LQN and GSPN Models of Exceptions

The network of branches in a scenario with multiple exceptions translates readily into a corresponding GSPN, but requires attention to obtain the call parameters in the LQN created from the scenario. For completeness, the calculation of the call parameters is described using a flow graph, and its application is shown for the UCM in Figure 5-3 which gives the LQN structure in Figure 5-5.

```java
public void exampleMethod(String filename)
{
    int lines = read(filename);
    if (lines==0)
    {
        report();
    }
    else
    {
        while (lines>0)
        {
            parse(lines);
            lines--;
        }
        complete();
    }
}
```

**Figure 5-6 A Code Block and the Associated Flow Graph**

A flow graph \( F = (G, a, \zeta) \) is a triple consisting of a directed digraph \( G \) together with distinguished nodes \( a, \zeta \) of \( G \) satisfying: (i) the node \( a \) (the start node) has directed paths to all other nodes of \( G \); (ii) every node has a directed path to \( \zeta \) (the stop node). The example of a program (written in Java) and a corresponding flow graph is given in Figure 5-6. Paper [Fenton85] shows how a given program including the select statements and the loop statements is mapped to a flow graph. The papers [Fenton85][Fenton86] define the composition of flow graphs if there is the invocation between two flow graphs and illustrate the sub-flow graph conception. The conception of path in flow graphs is the same as the path in UCMs. Also the node in flow graphs is like the responsibility in UCMs. The flow
graph notation is extended to support the exception handling like the abort model. The exception handling mechanism introduces an exception handler path and the nodes representing actions in an exception handler along the exception handler path. An escape node is graphically represented through a circle in the extended flow graphs. The same example presented in Figure 5-6 is presented on the left of Figure 5-7. But the condition selection in Figure 5-7 is replaced as a try catch. An extended flow graph is developed on the right of Figure 5-7. Thus how to derive the extended flow graph from the code of a program with exception handling is illustrated.

```java
public void examplemethod_1(string filename)
{
    try
    {
        int lines = read(filename);
        while (lines>0)
        {
            parse(lines);
            lines --;
        }
        complete();
    }
    catch (lineZeroException)
    {
        report();
    }
}
```

**Figure 5-7 The Extended Flow Graph**

A flow graph is constructed from the UCM using the approach described for software in [Fenton85]. A weight is given for each node representing its visit count, with a visit count of 1 for the starting node (and the remainder being unknown), and a label on each edge representing the probability of taking the successor path along that edge.

A flow graph used here is a directed graph with a unique start node, and a path from the start node to every other node. From the UCM, the flow graph is constructed using the approach described for software in [Fenton85]. This gives a graph with a node for each
responsibility and control point (branch/merge/fork/join). The nodes are numbered (with node 1 being the starting node) and node \( i \) is given a weight \( W_i \) representing the number of visits to node \( i \) for each execution of the graph, starting at node 1. Node 1 has a weight of 1, and the others are to be found. The edge from node \( i \) to node \( j \) has label \( p_{ij} \) representing the probability of taking the path to node \( j \) when leaving node \( i \). Then the weights \( W \) are found from the equations:

\[
W_i = \sum_j W_j p_{ji}, i > 1
\]

taking \( p_{ij} = 0 \) for arcs which are missing.

In constructing a LQN performance model, the weights are used to find the labels for LQN call arcs. Where a UCM arc \((i,j)\) represents a transfer from one component to another, the LQN has a call with a label \( W_i / W_j \) from an entry to another.

Following the calculation of the weights for the flow graph, the weights are derived from Figure 5-3 (B) as follows:

\[
W(Arep1) = W(Brep1) = 1, \quad W(Brep2) = 1 - p_1, \\
W(Brep3) = W(Arep2) = W(Crep1) = \prod_{i=1}^{2} (1 - p_i), \quad W(Crep2) = \prod_{i=1}^{3} (1 - p_i), \\
W(Crep3) = W(Arep3) = \prod_{i=1}^{4} (1 - p_i), \quad W(Arep4) = \prod_{i=1}^{5} (1 - p_i).
\]

For each responsibility as an exception handler, there are:

\[
W(Ahdl_k) = \sum_{j=1}^{5} (\lambda_j p_j \prod_{i=1}^{j-1} (1 - p_i)) \quad \text{for } k = 1, 2, 3,
\]

in which: \( \lambda_j = 1 \) if \( excep_{-rs}(ep_j) = e_k \)

\[
\lambda_j = 0 \quad \text{if } excep_{-rs}(ep_j) \neq e_k \quad \text{for } j = 1 \ldots 5.
\]
There is $W(HLEn) = 1$ for the starting node. The parameters of entries for the normal execution in method are specified as follows:

\[ Call(HLEn, Arepl) = \frac{W(Arepl)}{W(HLEn)} = 1, \]

\[ Call(HLEn, Arep2) = \frac{W(Arep2)}{W(HLEn)} = \prod_{i=1}^{2} (1 - p_i), \]

\[ Call(HLEn, Arep3) = \frac{W(Arep3)}{W(HLEn)} = \prod_{i=1}^{4} (1 - p_i), \]

\[ Call(HLEn, Arep4) = \frac{W(Arep4)}{W(HLEn)} = \prod_{i=1}^{5} (1 - p_i). \]

For the entries as exception handlers, there are:

\[ Call(HLEn, Ahdl_k) = \frac{W(Ahdl_k)}{W(HLEn)} = \sum_{j=1}^{k} (\lambda_j p_j \prod_{i=1}^{j-1} (1 - p_i)) \text{ for } k = 1, 2, 3, \]

in which $\lambda_j = 1$ if $\text{excep}_{-rs}(ep_j) = e_k$

\[ \lambda_j = 0 \text{ if } \text{excep}_{-rs}(ep_j) \neq e_k \text{ for } j \text{ from 1 to 5}. \]

The parameters for the rest of the calls are derived:

\[ Call(Arepl, Brepl) = \frac{W(Brepl)}{W(Arepl)} = 1, \]

\[ Call(Arepl, Brep2) = \frac{W(Brep2)}{W(Arepl)} = 1 - p_1, \]

\[ Call(Arepl, Brep3) = \frac{W(Brep3)}{W(Arepl)} = \prod_{i=1}^{2} (1 - p_i), \]

\[ Call(Arep2, Crepl) = \frac{W(Crepl)}{W(Arep2)} = 1, \]

\[ Call(Arep2, Crep2) = \frac{W(Crep2)}{W(Arep2)} = \prod_{i=3}^{3} (1 - p_i), \]

\[ Call(Arep2, Crep3) = \frac{W(Crep3)}{W(Arep2)} = \prod_{i=3}^{4} (1 - p_i). \]
5.1.4 Limitation of the Current Approach

In section 5.1, a scenario is employed to analyze how a performance model is developed. Performance model construction especially for LQN and GSPN is described. Following the nature of exception handling and the propagation of the occurrence of an exception, an approach to parameterizing the LQN and GSPN was also illustrated. This approach is applicable when the probability is known.

However if the exception occurs upon the state of a system, the approach to performance model construction with probability propagation is out of the realm where it is applicable. Assumption of the state of a system in term of probability seems the only way to approach this but may lead the inaccuracy performance analysis. HPMM is originated from this to provide the more accurate performance evaluation. In the remaining of this chapter, the exception handling mechanisms applied in resource allocation are used to illustrate the hybrid performance modeling practice.

5.2 Different Exception Models in Resource Allocation Exception Handling

Resource allocation is very important in a real system and in a performance model. In a real system, one cannot talk about the execution demand for the operation without considering resource allocation. This section discusses the scenarios that occur when different exception handling models are applied to resource allocation exceptions. Thus an interesting system that deploys different exception handling models in resource allocation is presented as a case study to demonstrate HPMM. This case study is interesting because it is very common in a distributed system for a request to take different actions according to the availability of resources. All the most popular exception handling models such as abort, infinite retry, and finite retry, will be investigated for resource allocation exception handling. Then, more disciplines are added to resource allocation and they may not be modeled through the queueing-based modeling strategy. Finally, resource allocation exception handling presents a
question how to construct and solve the performance model if the probability of a raised exception in resource allocation is unknown.

In the following sections, Use Case Map notation is extended to describe scenarios for resource allocation. Several typical exception handling models combined with resource allocation disciplines are illustrated through extended Use Case Maps.

5.2.1 Use Case Map Extensions

To facilitate describing the scenario for different resource allocation disciplines, some extensions of UCM for more intuitive graphic description are made. A UCM is extended through two elements: the resource together with the resource acquiring action and the resource releasing action, and a special resource allocation stub (different from the regular stub) in Figure 5-8.

![Figure 5-8 The Extension of a Use Case Map](image)

The element of resources is one extension of a UCM. Although by default the resource is just a component in a UCM, a special notation is introduced so that the behaviour during the resource allocation can be modeled. A resource is represented as a rounded rectangle and this work uses identifying names beginning with “R”. The upward triangle and the downward triangle are used to represent the actions of acquiring the resource and releasing the resource. They are located on the intersection of the path and the resource. Two
resources, R1 and R2, and their acquiring and releasing actions are shown in Figure 5-8. The necessary information about resources, such as the amount of a resource, should be attached to a UCM.

There are two kinds of resources that can be presented in a UCM: execution resource (hardware), and logical resource or software resource (software) [Woodside01]. If a UCM is used to develop a performance model, all resources that affect system performance should be defined in a UCM and all responsibilities defined within the resource rectangle should be executed through the resource. The brief introduction of resources is introduced as follows:

- **Execution resources.** Execution resources host the execution of the operations. Usually they are system devices such as processors, buses, and peripheral equipment such as disks, printers, etc. Execution resources are associated with components. When a new component is entered, its execution resource must implicitly be acquired.

- **Logical resources or software resources.** These resources are controllers that govern the behaviour of requests. The functionalities of these controllers are diverse. A controller conducts the admission control for a specific resource. For instance, a controller can instantiate a request as a thread in a multi-thread task, or allocate the space or buffer to the requests for moving further.

The resource allocation stub introduced here hides the fine-grained resource allocation detail. The resource allocation stub is graphically represented by a shaded diamond immediately preceding the allocation element on the path (see Figure 5-8). The plug-in for a resource allocation stub may have multiple start points representing different requests for the same resource, with labels that identify the points where it is bound. Multiple end points can be defined in a resource allocation stub so that they lead to different paths, with labels that identify the paths. The resource allocation stub may include operations, such as waiting for a fixed period, and decisions with branching paths. For instance in Figure 5-8, there is an extra path, `branchpath`, introduced by the resource allocation for a resource R2.
5.2.2 Resource Allocation Exception Handling

Based on the exception handling mechanisms discussed in section 2.1.3, four resource allocation patterns are defined: Queue pattern, Abort pattern, Infinite Retry pattern, and Finite Retry pattern. The basic characteristics of these patterns are discussed as follows:

- **Queue pattern**: All requests for the resource are put in a queue. Once the resource is available, a request from the queue can be selected to deploy the resource and conduct further operations. The decision as to which request should be chosen depends on the resource allocation disciplines such as FIFO, HOL, PPP, PS or RAND. The resource allocation stub in the scenario is omitted for the queue pattern since there are no extra paths or extra operations to be represented in a UCM. The queuing discipline is associated with the resource acquisition element instead. Also the queue pattern is set as a default pattern for the resource allocation discipline on an extended use case map. Figure 5-9 shows the scenario with resource R1 and resource R2 and the resource allocation for them is the queue pattern. To simplify the discussion for the rest of the patterns presented in this section, the queue pattern is always selected for the resource R1 allocation. Different resource allocation patterns are assigned for resource R2.

![Figure 5-9 The Scenario Model for Queue Pattern in Extended UCM](image)

- **Abort pattern**: In this pattern, the request tries to acquire the resource and if it cannot, it gives up and goes through the abort operation. The abort pattern includes the abort path for this request since a customer believes that the resource will not be available anymore or the request can't acquire the resource any more. A scenario
model in a UCM with resources R1 and R2 is illustrated in Figure 5-10. The resource allocation for R1 is the queue pattern and the resource allocation for R2 is the abort pattern. In the resource allocation stub, there are two exits for this resource allocation stub. One exit is for the requests that acquire the resource R2. These requests can move further along the *Acquire* path and perform the operation while deploying resource R2. The other exit is for the requests that cannot obtain the resource. These requests conduct the abort operation along the *Abort* path, and then terminate execution of the UCM.

![Diagram of resource allocation stub](image)

**Figure 5-10 THE SCENARIO MODEL FOR ABORT PATTERN IN EXTENDED UCM**

- *Infinite Retry pattern*: If the resource is occupied by other operations, the request retries to acquire the resource after a fixed timeout period. The retry behavior continues until the request obtains the resource. The resource allocation stub is shown in Figure 5-11. Within the stub a timer is used to represent a fixed time period. There is one exit that leads to the *Acquire* path.

![Diagram of infinite retry pattern](image)

**Figure 5-11 THE SCENARIO MODEL FOR INFINITE RETRY PATTERN IN EXTENDED UCM**
• **Finite Retry pattern**: In the finite retry pattern, the customer believes that the resource may have disappeared and won't come back, after a number of failed attempts. The request aborts the remaining operations along the abort path. The behaviour in the finite retry pattern is a combination between waiting for a fixed time period as in the infinite retry pattern multiple times and then abort as in the abort pattern. A scenario model in a UCM is shown in Figure 5-12. In the resource allocation stub, a timer shows the fixed time. Two exits are defined in the stub: one exit labeled as *Abort* is for the requests that cannot acquire the resource after numbers of tries, and the other one labeled as *Acquire* is for the requests that acquire the resource within finite times then perform further operations.

![Figure 5-12 The Scenario Model for Finite Retry Pattern in Extended UCM](image)

**5.2.3 Why is the Hybrid Modeling Solution needed**

Section 5.2.2 presents different disciplines in resource allocation when different exception handling models are deployed. During the performance analysis of resource allocation, some comparisons on behaviour analysis among resource allocation exception handling strategy are summarized here. In the queue pattern, there is no extra operation or extra path for the requests but rather, waiting in the queue if there is no resource available. This scenario is identical with the one that the queue-based performance modeling formalism can model. Compared to the queue pattern, the rest of the patterns include either the extra operations like waiting for the fixed time period then retrying, or the extra paths as the abort path. The scenario model of the abort pattern includes the abort path, which is also in the finite retry
pattern. The waiting for the fixed time period operation is also defined in the infinite retry pattern and the finite retry pattern. In the patterns other than the queue pattern, there are logical conditions that govern the resource allocation behaviour. The logical conditions are decided by the dynamic performance data like the availability of the resource or (and) the static problem data like the number of times to attempt to acquire the resource. The logical conditions can be identified as the decision cells. Following chapter 4, hybrid performance modeling is appropriate for this system.

5.3 Hybrid Performance Modeling for Resource Allocation Exception Handling

This section shows how HPMM can be applied to systems with resource exceptions, using an example of a web application with the four different resource allocation patterns described in Section 5.2 (queue, abort, finite, and infinite retry).

5.3.1 Scenario Description

As the discussion in section 5.2, resource allocation is common in software systems. Combining exception handling models with resource allocation enriches the resource allocation disciplines and provides the flexibility in the resource allocation context. We can consider in particular a database lock resource, which provides exclusive access to some record in a concurrent environment. Thus only one customer can access the record when the lock is applied. This is common in the database system. When the lock is not available, it is treated as an exception. The exception handling models are bound to the database lock. The behaviour of the exception handling models is considered in a layered application like a web application including a web layer and a database layer. The exception handling models impact the interaction between the web layer and the database layer, which is typical behaviour in the resource allocation exception handling system.
Thus an example of a web application is used to address the hybrid performance model construction of different exception handling models. A three tier web application is introduced. The exception handling models are employed in the resource allocation of this application. The system involves clients, one web server and one backend database server. The clients interact with the web site on the web server, alternating periods of thinking and sending requests to the web site. Clients always wait for a response from the previous request and keep the cycle of thinking, sending requests, and waiting for a response. This system is a typical closed system. The web server supports ten threads concurrently running and implements the page request operations. The requests are sent to the database server from the web server. The database server uses a couple of database operations such as query and update to implement the requests from the web server. The query and update operations both require several storage operations. There is the write operation in the update of the database. To guarantee the data consistently processing, there is a lock resource in the database server for the update operation. The multiple threads running on the database server content the lock resource. The different resource allocation exception handling models, named as the patterns described in section 5.2.2, are applied to the lock resource allocation in the web application system. The resource allocation patterns are the queue pattern, the abort pattern, the infinite retry pattern and the finite retry pattern respectively.

The performance modeling solution for all these patterns are introduced in the remaining sections. After analyzing and partition the behaviour of the queue pattern of the resource allocation exception handling in the web application system, a LQN model is derived in section 5.3.2. Although this LQN model is not a hybrid solution, the behaviour cells and the related partitions identified are reused for modeling other resource allocation exception handling patterns. The behaviour cell aggregation and sub-scenario constructions are illustrated in developing the GSPN sub-model for the queue pattern and data exchange is defined in section 5.3.3. The identified behaviour cells in behaviour partition and the aggregation of behaviour cells for sub-model construction can be extended for modeling other patterns in the web application system. Further procedures like behaviour partition with model formalism selection and sub-model construction discussed in HPMM, the hybrid performance model constructions are illustrated in modeling the web application system.
with exception handling patterns like aborting (section 5.3.4), infinite retry (section 5.3.5), and finite retry (section 5.3.6).

5.3.2 Queue Pattern: Deriving the LQN Model

This section focuses on the performance model construction of the queue pattern of the resource allocation exception handling in the LQN model formalism. Although it does not require the hybrid performance modeling solution, it provides steps like developing the scenario model and constructing a LQN model. Combined with the hybrid performance modeling solutions of the other resource allocation patterns in the remaining of this chapter 4, this section makes the complete performance modeling solution for resource allocation exception handling.

![Figure 5-13 The Scenario Model of a Web Application with the Queue Pattern](image)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>HardwareResource (CPU)</th>
<th>SoftwareResource (Process/Thread)</th>
<th>LogicalResource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thinking</td>
<td>ClientCPU(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PageRequest</td>
<td>WebServer(1)</td>
<td>WebWorker(10)</td>
<td></td>
</tr>
<tr>
<td>OpQuery</td>
<td>DatabaseServer(1)</td>
<td>DBWorker(5)</td>
<td></td>
</tr>
<tr>
<td>Diskop (for Query)</td>
<td>StorageSystem(1)</td>
<td>Disk(1)</td>
<td></td>
</tr>
<tr>
<td>OpUpdate</td>
<td>DatabaseServer(1)</td>
<td>DBWorker(5)</td>
<td>Lock(1)</td>
</tr>
<tr>
<td>Diskop (for Update)</td>
<td>StorageSystem(1)</td>
<td>Disk(1)</td>
<td>Lock(1)</td>
</tr>
</tbody>
</table>
A scenario model for the three-tier web application described in section 5.3.1 with the queue pattern in the lock allocation mechanism is defined in Figure 5-13. There is a set of responsibilities: Thinking on the clientCPU, PageRequest on the WebServer, OpQuery and OpUpdate on the DatabaseServer, and Diskop on the StorageSystem. Instead of representing a resource as a round rectangle, the resource context is tabulated in Figure 5-13. The first column “Responsibility” lists all responsibilities. The rest columns: “Hardware Resource”, “Software Resource”, “Logical Resource”, contain the resources names and resource counts respectively. The “Hardware Resource” includes a clientCPU, a WebServer, a DatabaseServer, and a StorageSystem. The “Software Resource” includes ten WebWorker threads (or processes) on the WebServer, five DBWorker threads (or processes) on the DatabaseServer, a Disk thread (or process) on the StorageSystem. The “Logical Resource” includes a Lock. After the name of each resource, there is a digital number that indicates the number of the resources. Each row represents that a responsibility in the “Responsibility” deploys one resource listed in the “Hardware Resource”, “Software Resource”, or “Logical Resource” columns and conducts the operations. For instance, the row of the OpUpdate means the responsibility OpUpdate requires a DBWorker thread (process) with the writing lock to run on the DatabaseServer.

The resource context tables are omitted in the scenario models of other exception handling patterns of resource allocation in this chapter since they are the same as in Figure 5-13. The lock resource is highlighted in Figure 5-13 as well as in the rest figures that represent a scenario model in this chapter. The responsibility OpUpdate as well as Diskop for the update operation holds the lock resource until the update operation on the database server. The resource allocation stub with the queue pattern is omitted in Figure 5-13. The lock allocation stub represents the other patterns as described in section 5.2.2. The lock allocation stub can be expanded in different ways for different lock allocation patterns. The stub may lead to an extra path with the OpAbort responsibility for the requests that can’t get the lock resource. For instance, there is the Abort path with the failure responsibility OpAbort in the abort pattern or the finite retry pattern. That causes a slightly different scenario model from the one in Figure 5-13. The expanded scenarios will be discussed in the rest of the sections when the hybrid performance model is constructed for the rest patterns.
Behaviour analysis and partition can be conducted for the scenario model. Following the identification of the behaviour cell in HPMM discussed in section 4.2, all the UCM elements like responsibilities in Figure 5-13 are defined as behaviour cells. Thus there are Thinking Responsibility Cell, PageRequest Responsibility Cell, OpQuery Responsibility Cell, OpUpdate Responsibility Cell, and Diskop Responsibility Cell. The queueing for software resources like threads/processes are identified as the WebWorker Queue Cell, DBWorker Queue Cell and Disk Queue Cell. The queueing for hardware resources are denoted as WebServer Queue Cell, DatabaseServer Queue Cell, and StorageSystem Queue Cell. In the queue pattern, the request is queued for the logical resource, the lock. Thus Queue Pattern Lock Queue Cell is specified. All behaviour cells are shown in Figure 5-14.

For each behaviour cell, the scores of the two formalisms, LQN and GSPN, are listed in Table 5-1 based on the common behaviour cell scores listed in Table 4-1. To avoid repeating the same behaviour cells identified in this web application with the other resource allocation exception handling patterns, the Queue Pattern Lock Queue Cell, specifically for the web application with resource allocation queue pattern, is listed in Table 5-2 rather than Table 5-1. The scores of the Queue Pattern Lock Queue Cell for LQN and GSPN are also listed in Table 5-2.

From Tables 5-1 and 5-2, both GSPN and LQN are suitable for all Responsibility Cells. LQN is more suitable for the Queue Cells. Due to the connection between the
Responsibility Cells and the Queue Cells, LQN is selected to model all behaviour cells, giving a single formalism for the model. In this special case, all behaviour cells are composed to a single behaviour fragment $BF_1$, the complement $\overline{BF_1}$ is empty and its abstraction is not created. The behaviour $B_1$ is the same as $BF_1$, and the scenario model $S_1$ is exactly the original one shown in Figure 5-13.

Table 5-1 Scores of Basic Behaviour Cells of the Web Application

<table>
<thead>
<tr>
<th>Behaviour Cells</th>
<th>GSPN</th>
<th>LQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thinking Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>PageRequest Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>OpQuery Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>OpUpdate Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Diskop Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>WebWorker Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>DBWorker Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>WebServer Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>DBServer Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Storage Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Disk Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 5-2 Scores of Queue Pattern Behaviour Cells in the Web Application

<table>
<thead>
<tr>
<th>Behaviour Cells</th>
<th>GSPN</th>
<th>LQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue Pattern Lock Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

The parameters used to develop the example model are as follows. The workload of this system is “closed” with 20 clients that cycle between requests and thinking. Each client
spends $Z$ seconds in thinking. The web server with 10 threads handles the request from clients in 0.05 second CPU demand, then sends the request to the database server. The database server with 5 threads copes with the requests from the web server. The requests to the database server are the query and update operations. The database server spends 0.02 second CPU demand plus 5 disk operations in a query operation and 0.03 second CPU demand plus 3 disk operations in an update operation. The storage system like disk spends 0.05 second CPU demand per disk operation. An exclusive lock is introduced into the system to guarantee data consistency giving a lock resource with multiplicity 1.

A LQN model for the queue pattern is given in Figure 5-15 (A) according to the scenario model $S_1$ in Figure 5-13. The attributes of each element in the LQN model have been directly obtained from the scenario model, which is also shown in Figure 5-15 (A). The mapping between the entries in the LQN model and the responsibilities in the Use Case Map can be clearly identified. The **hardware resource** in Figure 5-13 is mapped to a processor in the LQN model so that there are ClientCPU, WebServer, DatabaseServer, and StorageSystem processors defined in the LQN model. The **software resource** in Figure 5-13 is mapped to a task in the LQN model. Therefore there are multi-task instances such as WebWorker, DBWorker and Disk with the number as 10, 5, and 1, which correspond to 10 threads on the web server, 5 threads on the database server, and one thread on the storage server. The **logical resource** in Figure 5-13 can be also mapped to a task with an entry in the LQN model. Due to only one lock in the database is one task LockWorker with one entry defined in the LQN model. Therefore the task, LockWorker, is used to capture the behaviour of each request for the lock, such as waiting in a queue and holding the write lock until the update operation is finished. Therefore the purely LQN model in Figure 5-15( A) can be used as the performance model for a system that contains the queue pattern in the resource allocation exception handling, and hybrid modeling is not required.
(A) The LQN Model

(B) The Adjusted LQN Model

Figure 5-15 The LQN Model of the Queue Pattern
Purely as a technique for reporting the response time from the LQN solver tool, a pseudo task with an entry called \textit{DBRequest} was introduced before the \textit{DBLock} entry (see Fig 5-15 (B)). It transparently passes the request through but returns a mean service time which is the waiting plus service time for \textit{DBLock}. The pseudo task (with an infinite thread and an infinite pseudo processor) makes no difference to the model results.

5.3.3 \textit{Queue Pattern: Hybrid Model Solution}

The selected formalism above was all-LQN. Just to show the steps for a more complex situation, this section considers an artificial case where the GSPN formalism is chosen for some behaviour cells, even though its suitability is rated low. Then there are two sub-scenarios, and all the steps of HPMM are carried out for the alternative solution.

\textbf{Figure 5-16} BEHAVIOUR AGGREGATION FOR THE GSPN MODEL OF THE QUEUE PATTERN

GSPN formalism can be selected for the web application with the queue pattern in resource allocation exception handling in Figure 5-13 following behaviour partition defined in HPMM. Although behaviour cells like \textit{DBWorker} Queue Cell and \textit{Queue Pattern Lock} Queue Cell for GSPN formalism are scored low in Table 5-1 and 5-2, GSPN is selected for both behaviour cells. GSPN has the capacity to model the queue cells although it is not
recommended in HPMM. For the GSPN formalism, the behaviour fragment $BF_2$ contains $DBWorker$ Queue Cell and $Queue Pattern Lock$ Queue Cell. The complement of $BF_2$, $BF_2^\text{c}$, includes the remaining behaviour cells other than $DBWorker$ Queue Cell and $Queue Pattern Lock$ Queue Cell.

$BF_2^\text{c}$ can be derived from the behaviour aggregation. The $OpQuery$ on the database server with 5 disk operations on the storage system is aggregated as $Query^\text{*}$. The $OpUpdate$ on the database server with 3 disk operations on the storage system is aggregated as $Update^\text{*}$. The rests such as the thinking on the client side and the $PageRequest$ on the web server are aggregated as $Delay^\text{*}$. All these aggregated responsibilities are surrogate delays in the behaviour consistency of HPMM. Thus Figure 5-16 shows the possible aggregations in $BF_2^\text{c}$ and these aggregations are presented as $BF_2^{\text{c}A}$.

The sub-scenario can be derived after deriving $BF_2$ and $BF_2^{\text{c}A}$ for GSPN formalism. The behaviour $B_2 = BF_2 \cup BF_2^{\text{c}A}$ and the sub-scenario $S_2$ shown in Figure 5-17 is obtained from $B_2$. Based on the sub-scenario $S_2$, a GSPN model is developed for the queue pattern in Figure 5-18.

---

**Figure 5-17** The GSPN Scenario Model of the Queue Pattern
The parameters of the abstract elements as surrogate delays: Delay*, Query*, and Update* in the GSPN sub-model should be defined. The value of the parameters is obtained from the LQN model in Figure 5-15 in the hybrid performance model methodology. The timed transitions Query and Update represent the surrogate delays for the Query* and Update* responsibilities and the transition rates of them are derived from the service time collected from the entries OpQuery and OpUpdate in the LQN model in Figure 5-15. The timed transition Delay is a surrogate delay for the response time for the responsibilities that are not on the database server and can be calculated through the complementary computation defined in section 6.3. Also the initial marking of the places User, CPU, Lock should be consistent with the scenario description, which has 20, 5, 1 token(s) respectively. These are the same definition as in LQN model with a number of instances of the identical tasks such as User, DBWorker and LockWorker. The PageRequest on the web server and the thinking time are aggregated into a timed transition Delay. The behaviour for disk operation contention is also aggregated into the timed transitions Query and Update respectively.

Therefore the following equations are used to define the firing rates of timed transitions in the GSPN model in Figure 5-18:

\[
\mu(Delay) = \frac{Mark(User)}{[N/f - ServiceTime(DBRequest)]} \quad \text{Equ. 5-1}
\]

\[
\mu(Query) = \frac{Mark(P2)}{ServiceTime(OpQuery)} \quad \text{Equ. 5-2}
\]

\[
\mu(Update) = \frac{Mark(P4)}{ServiceTime(OpUpdate)} \quad \text{Equ. 5-3}
\]
All the values of the variables, the firing rates of timed transitions in equations, Equ. 5-1, Equ. 5-2, and Equ. 5-3, are obtained from scenario description or the LQN model in Figure 5-15 (B). The data exchange between the LQN model in Figure 5-15 (B) and the GSPN model in Figure 5-18 can be represented in Table 5-3.

**Table 5-3 Data Exchange between the LQN Model and the GSPN Model of the Queue Pattern**

<table>
<thead>
<tr>
<th>GSPN Sub-Model Def. from LQN Sub-Model</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu(Delay)$</td>
<td>Equ. 5-1</td>
</tr>
<tr>
<td>$\mu(Query)$</td>
<td>Equ. 5-2</td>
</tr>
<tr>
<td>$\mu(Update)$</td>
<td>Equ. 5-3</td>
</tr>
</tbody>
</table>

5.3.4 *Abort Pattern: Hybrid Model Construction*

Figure 5-19 shows the scenario model of the web application with the abort pattern, including a lock allocation stub with the abort pattern (as defined in section 5.2.2) and an *Abort Path* with the *OpAbort* responsibility, which is followed if the resource is unavailable. The definition of the resource context in Figure 5-13 is reused in Figure 5-19. The additional resource context of the responsibility *OpAbort* in the Abort pattern includes the
Databaseserver (a Hardware Resource), and five DBWorker threads (or processes) (a Software Resource) on the Databaseserver. The database server spends 0.02 second CPU demand for OpAbort.

![Diagram of the web application with the abort pattern](image)

**Figure 5-20 Behaviour Cells of the Web Application with the Abort Pattern**

Behaviour analysis gives the same behaviour cells and their scores for LQN and GSPN as in Table 5-1, plus an Abort Pattern Lock Decision Cell (which, as a decision cell has the dynamic performance data in Table 4-1) and an OpAbort Responsibility Cell. All behaviour cells are shown in Figure 5-20. Following the recommended scores listed in Table 4-1, these two cells, Abort Pattern Lock Decision Cell and OpAbort Responsibility Cell are listed in Table 5-4 with the scores for GSPN and LQN.

<table>
<thead>
<tr>
<th>Behaviour Cells</th>
<th>GSPN</th>
<th>LQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort Pattern Lock Decision Cell</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>OpAbort Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
</tbody>
</table>

From the formalism scores in Table 5-1 and Table 5-4, both GSPN and LQN are suitable for all Responsibility Cells. LQN is better for the Queue Cells, and GSPN for Abort Pattern Lock Decision Cell. Following the discussion in section 4.2, LQN is also selected to model
the all behaviour responsibility cells. Thus the LQN behaviour fragment \( BF_1 \) includes all the cells except \textit{Abort Pattern Lock} Decision Cell, which is in the GSPN behaviour fragment \( BF_2 \), as shown by the behaviour partition in Figure 5-21. The behaviour partition is represented by the dashed line named as \( BF\_Boundary \).

![Figure 5-21 Behaviour Partition of the Web Application with the Abort Pattern](image)

Each behaviour fragment and the abstraction of other fragments can be combined to further develop a scenario model. For the given LQN formalism, \( BF_1 \) is united with the abstraction of \( BF_2, BF_2^A \), as \( B_1 \). In \( BF_2^A \), \textit{DBWorker} Queue Cell keeps the same as in \( BF_2 \). \textit{Abort Pattern Lock} Decision Cell is abstracted as two probabilities in \( BF_2^A \). The probability of the state in which Lock resource is not available is defined as \( P_{\text{Abort}} \) and the probability of the state in that Lock resource is available is defined as \( P_{\text{Acquire}} \). The surrogate flow discussed as in behaviour consistency is used in \( BF_2^A \). The probabilities for the paths encapsulate the involved responsibilities, the lock resource availability detection and the path selection behaviour in \textit{Abort Pattern Lock} Decision Cell. The LQN scenario model for the abort pattern, \( S_1 \), is derived in Figure 5-22 from \( BF_1 \) and \( BF_2^A \).
Figure 5-22 The LQN Scenario Model of Abort Pattern

(A) Aggregations of the UCM of Abort Pattern

(B) The GSPN Scenario Model of Abort Pattern

Figure 5-23 Behaviour Aggregation and the GSPN Scenario Model of Abort Pattern
For GSPN formalism, $BF_2$ is united with the abstraction of $BF_1, BF_1^A$, as $B_2$. The similar aggregations for Query*, Update*, and Delay* in section 5.3.3 are applicable here. Meanwhile, the aggregated responsibility Abort* is introduced along the abort path in $BF_1^A$. The OpAbort on the database server system is aggregated as Abort*. Figure 5-23 (A) shows all aggregations as surrogate delays in behaviour consistency. The GSPN scenario model of the abort pattern, $S_2$ in Figure 5-23 (B), is obtained from $BF_2$ and $BF_1^A$.

![Figure 5-24 THE LQN SUB-MODEL OF ABORT PATTERN](image)

The LQN sub-model is defined from the LQN scenario model $S_1$. In the LQN scenario model $S_1$, $P_{Abort}$ and $P_{Acquire}$ are defined. The relationship between them is defined through the equation Equ. 5-4. The mapping strategy of developing a LQN model from the scenario model in section 5.3.2 is applied here. It should be mentioned that the responsibility OpAbort on the Abort Path in Figure 5-20 is mapped to the entry OpAbort in the DBWorker task in Figure 5-24 following the resource context of the Abort Pattern. The LQN sub-model is
shown in Figure 5-24. In Figure 5-24, the mean number of calls from the entry DBRequest to the entry OpAbort and to the entry OpUpdate are \( P_{\text{Abort}} \) and \( P_{\text{Acquire}} = (1 - P_{\text{Abort}}) \) respectively.

\[
P_{\text{Acquire}} + P_{\text{Abort}} = 1 \tag{Equ. 5-4}
\]

The GSPN sub-model is developed from the GSPN scenario model \( S_2 \). The definition of surrogate delays: \( \text{Delay}^* \), \( \text{Query}^* \), and \( \text{Update}^* \), and the definition of the places \( \text{User}, \text{CPU}, \text{Lock} \) including the initial marking in the GSPN sub-model follow the same discussion of section 5.3.3. The timed transitions \( \text{Abort} \) in the GSPN sub-model represents the surrogate delay of the \( \text{Abort}^* \) responsibility. Thus in the database server the request token moves through \( \text{Abort} \) timed transition if the lock resource is not available, otherwise it will move through the \( \text{Update} \) timed transition if available. The GSPN sub-model is presented in Figure 5-25.

![Figure 5-25 The GSPN Sub-Model of Abort Pattern](image)

The parameters of abstracted elements in one sub-model are defined through the performance measures of the other sub-model. The probability \( P_{\text{Abort}} \) is defined, which the LQN sub-model imports from the GSPN sub-model for the abort pattern as equation Equ. 5-5. The rates of all timed transitions are defined in the GSPN sub-model and they are derived from the LQN sub-model through the equations Equ. 5-1, Equ. 5-2, Equ. 5-3 and the following definition for the timed transition \( \text{Abort} \).
The data exchange between the GSPN sub-model and the LQN sub-model is defined through a set of equations. The hybrid performance modeling solution for a web application system with the abort pattern includes the LQN sub-model defined in Figure 5-24 and the GSPN sub-model in Figure 5-25. The LQN sub-model imports the data through equation Equ. 5-5 from the GSPN sub-model for the complete definition. The LQN sub-model exports the data through equations Equ. 5-1, Equ. 5-2, Equ. 5-3, Equ. 5-6 to the GSPN sub-model for the GSPN model definition. The data exchange between the LQN model and the GSPN model can be represented in Table 5-5.

<table>
<thead>
<tr>
<th>GSPN Sub-Model Def. from LQN Sub-Model</th>
<th>LQN Sub-Model Def. from GSPN Sub-Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu(\text{Delay})$</td>
<td>Equ. 5-1</td>
</tr>
<tr>
<td>$\mu(\text{Query})$</td>
<td>Equ. 5-2</td>
</tr>
<tr>
<td>$\mu(\text{Update})$</td>
<td>Equ. 5-3</td>
</tr>
<tr>
<td>$\mu(\text{Abort})$</td>
<td>Equ. 5-6</td>
</tr>
</tbody>
</table>

Table 5-5 Data Exchange Between the LQN Model and the GSPN Model of the Abort Pattern
5.3.5 Infinite Retry Pattern: Hybrid Model Construction

Figure 5-26 shows the scenario model with the infinite retry pattern, a lock allocation stub with the infinite retry pattern (as defined in Section 5.2.2), and the Retry Path with the OpTimer responsibility. The system will try to obtain the lock every 0.4 seconds (the OpTimer delay) until the lock is allocated. The definition of the resource context in Figure 5-13 in section 5.3.2 can be reused in Figure 5-26. The extra resource context of the responsibility OpTimer in the Infinite Retry Pattern implicitly includes a software process Delay and hardware DelayServer, to execute the delay.
Behaviour analysis gives the same behaviour cells and their scores for LQN and GSPN as identified in Table 5-1, plus an Infinite Retry Pattern Lock Decision Cell and an OpTimer Responsibility Cell, with the formalism scores listed in Table 5-6. All behaviour cells are shown in Figure 5-27. In the infinite retry pattern if the lock resource is available, the requests take the Acquire path, otherwise they take the Retry Path and wait for a period at Timer. The Decision Cell chooses between these two paths, and has the dynamic performance data in Table 4-1, including the availability of the resource lock.

Table 5-6 Scores of the Infinite Retry Pattern Behaviour Cells in the Web Application

<table>
<thead>
<tr>
<th>Behaviour Cells</th>
<th>GSPN</th>
<th>LQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinite Retry Pattern Lock Decision Cell</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>OpTimer Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
</tbody>
</table>

From the formalism scores in Tables 5-1 and 5-6, both GSPN and LQN are suitable for all Responsibility Cells. LQN is better for the Queue Cells, and GSPN for Infinite Retry Pattern Lock Decision Cell. Thus the LQN behaviour fragment $BF_1$ includes all the cells except Infinite Retry Pattern Lock Decision Cell, which is in the GSPN behaviour fragment $BF_2$, as
shown by the behaviour partition in Figure 5-28. The behaviour partition is represented by the dashed line named as $BF_{\text{Boundary}}$.

![Diagram](image)

**Figure 5-28 Behaviour Partition of the Web Application with the Infinite Retry Pattern**

Each behaviour fragment and the abstraction of other fragments are combined to further develop a scenario model. For the given LQN formalism, $BF_1$ is united with the abstraction of $BF_2$, $BF_2^A$, as $B_1$. In $BF_2^A$, $DBWorker$ Queue Cell keeps the same as in $BF_2$. Infinite Retry Pattern Lock Decision Cell is abstracted as the mean visit count in $BF_2^A$. The mean times of retry are defined the mean visit count denoted by $V_{\text{Retry}}$. The surrogate flow discussed as in behaviour consistency is used in $BF_2^A$. The visit counts for the paths encapsulate the involved responsibilities, the lock resource availability detection and the path selection behaviour in Infinite Retry Pattern Lock Decision Cell. The LQN scenario model for the infinite retry pattern, $S_1$, is derived in Figure 5-29 from $BF_1$ and $BF_2^A$. 

120
(A) Aggregations of the UCM of the Infinite Retry Pattern

(B) GSPN Scenario Model of the Infinite Retry Pattern

Figure 5-29 The LQN Scenario Model of the Infinite Retry Pattern

Figure 5-30 Behaviour Aggregations and GSPN Scenario Model of the Infinite Retry Pattern
For GSPN formalism, $BF_2$ is united with the abstraction of $BF_1$, $BF_1^A$, as $B_2$. The similar aggregations for $Query^*$, $Update^*$, and $Delay^*$ in section 5.3.3 are applicable here. Meanwhile, the aggregated responsibility $Timer^*$ is introduced along the Retry path in $BF_1^A$.

The $OpTimer$ on the DelayServer server is aggregated as $Timer^*$. Figure 5-30 (A) shows all aggregations as surrogate delays in behaviour consistency. The GSPN scenario model for the abort pattern, $S_2$ in Figure 5-30 (B), is obtained from $BF_2$ and $BF_1^A$.

The LQN sub-model is defined from the LQN scenario model $S_1$. In the LQN scenario model $S_1$, $V_{Retr}$ is defined. The mapping strategy of developing a LQN model from the scenario model described in section 5.3.2 can be applied here. A server DelayServer with an infinite task Delay is introduced in the LQN sub-model following the resource context of the Infinite Retry Pattern. The entry SleepDelay represents the pure delay of the $OpTimer$ responsibility in the LQN sub-model. The LQN sub-model is shown in Figure 5-31. In Figure 5-31, the mean number of calls from the entry DBRequest to the entry SleepDelay is $V_{Retr}$.

![Figure 5-31 The LQN Sub-Model of the Infinite Retry Pattern](image)
The GSPN sub-model is developed from the GSPN scenario model \( S_2 \). The definition of surrogate delays: \( \text{Delay}^*, \text{Query}^*, \) and \( \text{Update}^* \), and the definition of the places \( \text{User}, \text{CPU}, \text{Lock} \) including the initial marking in the GSPN sub-model follow the same discussion of section 5.3.3. The timed transitions \( \text{Timeout} \) in the GSPN sub-model represents the surrogate delay of the \( \text{Timer}^* \) responsibility. Thus the request token moves through \( \text{Timeout} \) timed transition if Lock resource is not available and it will move through the \( \text{Update} \) timed transition if the lock resource in the database server is available. The GSPN sub-model is presented in Figure 5-32.

![Figure 5-32: The GSPN Sub-Model of the Infinite Retry Pattern](image)

The parameters of abstracted elements in one sub-model are defined through the performance measures of the other sub-model. The mean visit count \( V_{\text{retry}} \) is defined, which the LQN sub-model imports from the GSPN sub-model for the infinite retry pattern as equation Equ. 5-7. The rates of all timed transitions are defined in the GSPN sub-model and they are derived from the LQN sub-model through the equations Equ. 5-1, Equ. 5-2, Equ. 5-3 and Equ. 5-8 for the timed transition \( \text{Timeout} \):

\[
V_{\text{retry}} = \frac{f(\text{NoLock})}{f} \quad \text{Equ. 5-7}
\]

\[
\mu(\text{Timeout}) = \frac{\text{Mark}(P8)}{\text{ServiceTime}(\text{SleepDelay})} \quad \text{Equ. 5-8}
\]
The data exchange between the GSPN sub-model and the LQN sub-model is defined through a set of equations. The hybrid performance model solution for web application system with the infinite retry pattern is defined through the LQN sub model in Figure 5-31 and the GSPN sub-model in Figure 5-32. The data exchange among the sub-models is implemented based on a set of equations. The LQN sub-model imports the data for $V_{\text{Retry}}$ through equation Equ. 5-7 from the GSPN sub-model. The LQN sub-model also exports the data through equations Equ. 5-1, Equ. 5-2, Equ. 5-3, Equ. 5-8 to the GSPN sub-model in the firing rate of the timed transitions. The data exchange between the LQN model and the GSPN model can be represented in Table 5-7.

Table 5-7 Data Exchange Between the LQN Model and the GSPN Model of the Infinite Retry Pattern

<table>
<thead>
<tr>
<th>GSPN Sub-Model</th>
<th>Def. from LQN Sub-Model</th>
<th>LQN Sub-Model</th>
<th>Def. From GSPN Sub-Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu(\text{Delay})$</td>
<td>Equ. 5-1</td>
<td>$V_{\text{Retry}}$</td>
<td>Equ. 5-7</td>
</tr>
<tr>
<td>$\mu(\text{Query})$</td>
<td>Equ. 5-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu(\text{Update})$</td>
<td>Equ. 5-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu(\text{Timeout})$</td>
<td>Equ. 5-8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.3.6 Finite Retry Pattern: Hybrid Model Construction

Figure 5-33 shows the scenario model of the web application with the finite retry pattern, including a lock allocation stub with the finite retry pattern (as defined in section 5.2.2) and a Retry Path with the OpTimer responsibility, and an Abort Path with the OpAbort responsibility, which is followed if the resource is unavailable after multiple attempts. The system will try to obtain the lock every 0.4 seconds (the OpTimer delay). With three attempts, the lock is allocated if it is available. Otherwise the request will go along the abort path and spends 0.02 second CPU demand for OpAbort. The definition of the resource context in Figure 5-13 is reused in Figure 5-33. The additional resource context includes the responsibility OpTimer and the responsibility OpAbort. The resource context of the responsibility OpTimer is the same as stated in section 5.3.5 for the infinite retry pattern. The resource context of the responsibility OpAbort is the same as stated in section 5.3.4 for the abort pattern.
Behaviour analysis gives the same behaviour cells and their scores for LQN and GSPN identified in Table 5-1, plus \textit{Finite Retry Pattern Lock} Decision Cell (which, as a decision cell has the dynamic performance data in Table 4-1), \textit{OpTimer} Responsibility Cell, and an \textit{OpAbort} Responsibility Cell. All behaviour cells are shown in Figure 5-34. Following the recommended scores listed in Table 4-1, these three cells, \textit{Finite Retry Pattern Lock} Decision Cell, \textit{OpAbort} Responsibility Cell, and \textit{OpTimer} Responsibility Cell are listed in Table 5-8 with the scores for GSPN and LQN.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
Behaviour Cells & GSPN & LQN \\
\hline
\textit{Finite Retry Pattern Lock} Decision Cell & High & Low \\
\hline
\textit{OpAbort} Responsibility Cell & Medium & Medium \\
\hline
\textit{OpTimer} Responsibility Cell & Medium & Medium \\
\hline
\end{tabular}
\caption{Scores of the Finite Retry Pattern Behaviour Cells in the Web Application}
\end{table}

From the formalism scores in Table 5-1 and Table 5-8, both GSPN and LQN are suitable for all Responsibility Cells. LQN is better for the Queue Cells and GSPN for \textit{Finite Retry Pattern Lock} Decision Cell. Following the discussion in section 4.2, LQN is also selected to model the all behaviour responsibility cells. Thus the LQN behaviour fragment $BF_1$ includes
all the cells except *Finite Retry Pattern Lock* Decision Cell, which is in the GSPN behaviour fragment $BF_2$, as shown by the behaviour partition in Figure 5-35. The behaviour partition is represented by the dashed line named as $BF\_Boundary$.

*Figure 5-35 Behaviour Partition of the Web Application with Finite Retry Pattern*

Each behaviour fragment and the abstraction of other fragments are combined to further develop a scenario model. For the given LQN formalism, $BF_1$ is united with the abstraction of $BF_2$, $BF_2^A$, as $B_1$. In $BF_2^A$, $DB\text{Worker}$ Queue Cell keeps the same as in $BF_2$. *Finite Retry Pattern Lock* Decision Cell is abstracted as the mean time count of retry and two probabilities in $BF_2^A$. The mean time of retry is defined as the mean visit count denoted by $V_{\text{retry}}$. Two probabilities are the probability of acquiring the Lock resource as $P_{\text{Acquire}}$ and the probability of never acquiring Lock resource as $P_{\text{Abort}}$. The surrogate flow discussed as in behaviour consistency is used in $BF_2^A$. The visit count for the Retry path and the probability of acquiring lock encapsulate the involved responsibilities, the lock resource availability detection, the count of times of try for acquiring the lock resource, and the path selection behaviour in *Finite Retry Pattern Lock* Decision Cell. The LQN scenario model for the finite retry pattern, $S_1$, is derived in Figure 5-36 from $BF_1$ and $BF_2^A$. 

127
Figure 5-36 The LQN Scenario Model of the Finite Retry Pattern

(A) Aggregations of the UCM of the Finite Retry Pattern

(B) The GSPN Scenario Model of the Finite Retry Pattern

Figure 5-37 Behaviour Aggregations and the GSPN Scenario Model of the Finite Retry Pattern

128
For GSPN formalism, $BF_2$ is united with the abstraction of $BF_1$, $BF_1^A$, as $B_2$. The similar aggregations for *Query*, *Update*, and *Delay* in section 5.3.3 are applicable here. Meanwhile, the aggregated responsibilities *Timer* is introduced along the *Retry* path and the *Abort* responsibility is introduced along the *Abort* path in $BF_1^A$. The OpTimer on the DelayServer server is aggregated as *Timer*. Figure 5-37 (A) shows all aggregations as surrogate delays in behaviour consistency. The GSPN scenario model for the abort pattern, $S_2$ in Figure 5-37 (B), is obtained from $BF_2$ and $BF_1^A$.

The LQN sub-model is defined from the LQN scenario model $S_1$. In the LQN scenario model $S_1$, $V_{\text{retry}}$, $P_{\text{Abort}}$ and $P_{\text{Acquire}}$ are defined. The relationship between $P_{\text{Abort}}$ and $P_{\text{Acquire}}$ defined through the equation Equ. 5-4 is the same as here. The mapping strategy of developing a LQN model from the scenario model described in section 5.3.2 can be applied here. A server DelayServer with an infinite task Delay is introduced in the LQN sub-model following the resource context of the Finite Retry Pattern. The entry SleepDelay represents the pure delay of the OpTimer responsibility in the LQN sub-model. The responsibility OpAbort on the Abort Path in Figure 5-34 is mapped to the entry OpAbort in the DBWorker task in Figure 5-38. The LQN sub-model is shown in Figure 5-38. In Figure 5-38, the mean number of calls from the entry DBRequest to the entry SleepDelay is $V_{\text{retry}}$ and the mean number of calls from the entry DBRequest to the entry OpAbort and to the entry OpUpdate are $P_{\text{Abort}}$ and $P_{\text{Acquire}} = (1 - P_{\text{Abort}})$ respectively.
The GSPN sub-model is developed from the GSPN scenario model $S_2$. The definition of surrogate delays $\text{Delay}^*, \text{Query}^*$, and $\text{Update}^*$, and the definition of the places $\text{User}$, $\text{CPU}$, $\text{Lock}$ including the initial marking in the GSPN sub-model follow the discussion given in Section 5.3.3. The aggregated responsibility $\text{Abort}^*$ along the abort path in Figure 5-37 (B) can be mapped to the timed transition $\text{Abort}$ in the GSPN model. The aggregated responsibility $\text{Timer}^*$ along the retry path is mapped to two timed transitions with the same name $\text{Timeout}$ in Figure 5-39. The first timed transition with the name $\text{Timeout}$ as the input for the place $P9$ describes the action after the first try to acquire the lock and fail. The second timed transition with the name $\text{Timeout}$ as the input for the place $P11$ describes the action after the second try to acquire the lock and fail. The request token from customer starts with the first try in place $P3$. If the lock resource is available, the request token moves to the place $P4$ and passes through the timed transition $\text{Update}$ to perform the update operation. If the lock is not available, the request token moves to the place $P8$ then the place $P9$ for the second try. On the second try, the request token move to the place $P4$ if the lock is available or the place $P10$ if the lock is unavailable. If the request token is in the place $P10$, it makes up to three attempts to obtain the resource from the place $P11$. On the third failure, the request token moves to the place $P6$ and passes the timed transition $\text{Abort}$. Otherwise the request token
moves to place $P4$ for the *update* operation. Thus, the GSPN sub-model is presented in Figure 5-39.

![Figure 5-39 The GSPN Sub-Model of the Finite Retry Pattern (Three Attempts)](image)

The parameters of abstracted elements in one sub-model are defined through the performance measures of the other sub-model. The probability $P_{\text{Abort}}$ and the mean visit count $V_{\text{Abort}}$ are defined, which the LQN sub-model imports from the GSPN sub-model for the abort pattern. The mean visit count $V_{\text{Abort}}$ is derived through the equation Equ. 5-9 and the mean visit count $V_{\text{Abort}}$ is derived from the equation Equ. 5-10. The rates of all timed transitions are defined in the GSPN sub-model. The rates of the timed transition like $\text{Delay}$, $\text{Query}$, and $\text{Update}$ are derived from the LQN sub-model through the equations Equ. 5-1, Equ. 5-2, and Equ. 5-3. The timed transitions like $\text{Timeout}$ and $\text{Timeout1}$ are derived through the equations like Equ. 5-11 and Equ. 5-12. The timed transition like $\text{Abort}$ is derived from the equation Equ 5-13.
The data exchange between the GSPN sub-model and the LQN sub-model is defined. The hybrid performance model solution for web application system with finite retry pattern is defined by the LQN sub-model in Figure 5-38 and the GSPN sub-model in Figure 5-39. The data exchange among sub-model is defined through a set of equations. The LQN sub-model imports the data through equations Equ. 5-9 and Equ. 5-10 from the GSPN sub-model for $P_{Abort}$ and $V_{Retry}$. The LQN sub-model also exports the data through equations Equ. 5-1, Equ. 5-2, Equ. 5-3, Equ. 5-11, Equ. 5-12, Equ. 5-13 to the GSPN sub-model for the rates of all the timed transitions. The data exchange between the LQN model and the GSPN model can be represented in Table 5-9.

**Table 5-9 Data Exchange Between the LQN Model and the GSPN Model of the Finite Retry Pattern**

<table>
<thead>
<tr>
<th>GSPN Sub-Model Def. from LQN Sub-Model</th>
<th>LQN Sub-Model Def. from GSPN Sub-Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu(Delay)$</td>
<td>Equ. 5-1</td>
</tr>
<tr>
<td>$\mu(Query)$</td>
<td>Equ. 5-2</td>
</tr>
<tr>
<td>$\mu(Update)$</td>
<td>Equ. 5-3</td>
</tr>
<tr>
<td>$\mu(Timeout)$</td>
<td>Equ. 5-11</td>
</tr>
<tr>
<td>$\mu(Timeout1)$</td>
<td>Equ. 5-12</td>
</tr>
<tr>
<td>$\mu(Abort)$</td>
<td>Equ. 5-13</td>
</tr>
<tr>
<td>$P_{Abort}$</td>
<td>Equ. 5-9</td>
</tr>
<tr>
<td>$V_{Retry}$</td>
<td>Equ. 5-10</td>
</tr>
</tbody>
</table>
5.4 Hybrid Performance Model Results and Analysis

To investigate the accuracy of HPMM, the hybrid models were solved and the results were compared to simulation. The hybrid performance model solver in Section 4.7.3 was used, combining LQNS [Franks00][Franks95] and SPNP [Trivedi98]. The simulation was written in CSIM [Schwetman86] and solved with 95% confidence intervals.

The evaluation of HPMM applied in the resource allocation exception handling will cover:

- All the resource allocation exception handling patterns which requires HPMM like the Abort Pattern, the Infinite Retry Pattern, and the Finite Retry Pattern as described in section 5.3;
- A range of load intensity from the very low load to the high, which eventually almost saturates one of the resource
- All cases which the system bottleneck is the logic resource (the lock) and the cases where the lock is not the bottleneck

To vary the intensity of the load in the hybrid model, the number of customers was varied from 2 (low load) through increment by 2 until a resource is 90% utilized. The experiments are organized as the first case: disk bottleneck, and the second case: the lock bottleneck. As the number of customers is increased, the resource like the disk turns to be the bottleneck of the system in the first case and the lock is first saturated in the second case.

The accuracy of the hybrid performance model was examined through the lock acquisition probability (the abort pattern and the finite retry pattern), the lock acquisition retry (the infinite retry pattern and the finite retry pattern), the lock holding time, the utilization of the bottleneck, the response time of the web server and the useful throughput (the throughput of requests which acquire the lock to conduct the update operation). The first case shows
the disk usage in the utilization of the bottleneck and the second case presents the utilization of the lock in it.

5.4.1 First Case: Disk Bottleneck

In this case, the thinking time as Z in the LQN sub-model is 10 seconds. As described in section 5.3.2, the database server spends 0.02 second CPU demand plus 5 disk operations in a query operation and 0.03 second CPU demand plus 3 disk operations in an update operation. The disk spends 0.05 second device demand per disk operation.

5.4.1.1 First Case: Abort Pattern, Hybrid Model Results and Analysis

The hybrid model for the abort pattern was discussed in section 5.3.4. The two sub-models are shown in Figures 5-24 (LQN) and Figure 5-25(GSPN), and the data exchange is given in Table 5-5.

In Figure 5-40 the decline of the lock acquisition probability is shown in the simulation results (in the Figure, “Number of Users” represents the number of customers in the model). The hybrid model faithfully reproduces the trend with good accuracy in the lock acquisition probability. The similar comments on the accuracy apply to the lock holding time (Figure 5-41), the bottleneck utilization (Figure 5-42), and the useful throughput (Figure 5-44). Also the hybrid model shows the accurate agreement (Figure 5-43) in the response time of the web server at low and moderate load, but slightly larger errors at high load (the number of customers is greater than 20 and the bottleneck utilization is greater than 65%). In summary, the hybrid analytic model of the abort pattern gives reasonable-to-excellent accuracy in all except the response time of the web server at the high load (the bottleneck utilization is above 65%).
Figure 5-40 Lock Acquisition Probability for the Abort Pattern (First Case)

Figure 5-41 Mean Lock Holding Time for the Abort Pattern (First Case)
**Figure 5-42 Utilization of the Bottleneck (The Disk) for the Abort Pattern (First Case)**
Web Response Time (Abort) - Disk

- CSIM SIM 95% Range Low
- CSIM SIM 95% Range High
- ▲ Hybrid Analytical Solution

![Graph showing Web Response Time (Abort) for different conditions.

Number of Users: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
Response Time: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6

Figure 5-43 Response Time of the Web Server for the Abort Pattern (First Case)

Useful Throughput (Abort) - Disk

- CSIM SIM
- ▲ Hybrid Analytical Solution

![Graph showing Useful Throughput for different conditions.

Number of Users: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
Throughput: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4

Figure 5-44 Useful Throughput for the Abort Pattern (First Case)
It is noticed the error of the response time of the web server for abort pattern (the response time curve in Figure 5-43) is quite large (from 15% to 30%) for loads above 20 customers. Especially in the sample of 30 customers, the response time is 0.93 second in the hybrid model result and 1.34 seconds in the simulation. Since the lock acquisition probability results have small errors (less than 4%), it seems that this divergence might have another source in the analytic model solver.

To examine this question, the lock acquisition probability of 0.478, found in the hybrid solution for 30 customers, was applied to both the simulation and the analytic solver. The LQN model results show 0.0759 s. as the waiting time for the Diskop operation while the simulation results show 0.1076 s. There are 5 Diskop operations for database OpQuery and 3 for OpUpdate. Overall, it shows the LQN model is responsible for the response time error in the 30 customer sample. This results in the lock holding time being underestimated by 4% (Figure 5-41), and this appears in the response time results (Figure 5-43).

Further investigation of this model showed that the LQN model provides accurate predictions for the waiting time for Diskop operation when the StorageSystem is utilized under 70% and above 90%, but not for utilizations in the range 70%-90%. The divergence in the results is thus due to the approximations in the analytic solver, and the hybrid approach itself is not the source of the problem in this case. Since the hybrid approach depends on the LQN analytic solver, its accuracy will be limited by the LQN approximations.

Even with the errors of the response time of the web server for abort pattern (Figure 5-43), the overall results are good (less than 10%) when the load is low and moderate. As shown in Figure 5-43, the bottleneck (disk) is approaching to the saturation due to increasing load. The errors are mostly caused by underestimating the service time of database OpQuery, which includes 5 Diskop operations. The errors are disappointing but not due to the hybrid techniques itself, and the accuracy is sufficient for many purposes.
5.4.1.2 First Case: Infinite Retry Pattern, Hybrid Model Results and Analysis

The hybrid model for the infinite retry pattern was discussed in section 5.3.5. The two sub-models are shown in Figures 5-31 (LQN) and Figure 5-32 (GSPN), and the data exchange is given in Table 5-7.

Figure 5-45 shows the increase of the retries to acquire a lock as the load increases. The hybrid model overestimates the retries slightly when the load is low and moderate but the errors become larger when the load is greater than 16 customers. The similar comment on the accuracy is applicable to the lock holding time (Figure 5-46). The hybrid model shows the accurate agreement in the bottleneck utilization (Figure 5-47) and the useful throughput (Figure 5-49) at all experimented loads and the response time of the web server at low and moderate load, but slightly large errors (greater than 20%) the response time of the web server at the high load (Figure 5-48). Generally speaking, the hybrid analytic model for infinite retry pattern gives reasonable-to-excellent accuracy in all except number of retries, the lock holding time, the response time of the web server at the high load.

As in section 5.4.1.1, some experiments were designed to check the accuracy of LQNS, for retries introduced in scenarios. Two cases were selected, one with 18 customers and one retry, and the other with 24 customers and two retries. Both cases demonstrate the same underestimation of the queueing delay in the Diskop operation, as found before. This factor will make the LQN sub-model underestimate the service time of database OpQuery and OpUpdate operations, and the lock holding time; however this does not directly explain the errors in this case, in which the holding time in the solution is overestimated. Some other factors, to do with the GSPN model or the interaction of the two, must be responsible.

The overestimation of the retries, in the hybrid solution, is consistent with the overestimated lock holding time and web response time. If the GSPN model, for some reason, were to underestimate the probability of obtaining the lock, this would be enough to cause the observed error in the response time. The excess of failures leads to sleep delays, which are
longer than the lock holding time in all cases, this tends to further reduce the throughput and increase the response times. One possible reason for the GSPN model to underestimate the success probability is the approximation of the surrogate delays in the GSPN by exponentially distributed delays. This reasoning does not address the error in estimating the lock holding times, which appears to be due to processor waiting at the database host processor. This must be due to a more subtle interaction between the models. However the lock holding time errors are not large (they are overestimated by only about 10-15%).

![Retry to Acquire Lock (Inf Retry) - Disk](image)

**Figure 5-45 Mean Number of Retries for the Infinite Retry Pattern (First Case)**
Figure 5-46 Mean Lock Holding Time for the Infinite Retry Pattern (First Case)

Figure 5-47 Utilization of the Bottleneck (The Disk) for the Infinite Retry Pattern (First Case)
**Web Response Time (Inf Retry) - Disk**

![Graph showing Web Response Time](image)

- CSIM SIM 95% Low Range
- Hybrid Analytical Solution
- CSIM SIM 95% High Range

**Number of Users**

2 4 6 8 10 12 14 16 18 20 22 24 26 28

**Figure 5-48 Response Time of the Web Server for the Infinite Retry Pattern (First Case)**

**Useful Throughput (Inf Retry) - Disk**

![Graph showing Useful Throughput](image)

- CSIM SIM
- Hybrid Analytical Solution

**Number of Users**

2 4 6 8 10 12 14 16 18 20 22 24 26 28

**Figure 5-49 Useful Throughput for the Infinite Retry Pattern (First Case)**
5.4.1.3 First Case: Finite Retry Pattern, Hybrid Model Results and Analysis

The hybrid model for the finite retry pattern was discussed in section 5.3.6. The two sub­models are shown in Figures 5-38 (LQN) and Figure 5-39 (GSPN), and the data exchange is given in Table 5-9.

The slight decline of the lock acquisition probability (Figure 5-50) and the increase of the retries to acquire a lock (Figure 5-51) are shown against the increase of the load in the simulation results. The hybrid model reproduced the trend of the lock acquisition probability with good accuracy (Figure 5-50) at all experimented loads but provided increasing errors in predicting the number of retries (Figure 5-51) as the load increases. The hybrid model provides good accuracy for the lock holding time when the load is low and moderate but the accuracy is not so good when the load is high (Figure 5-52). As for infinite retries, it was overestimated by about 10-15%. The hybrid model accurately predicted the bottleneck utilization (Figure 5-53), the response time of the web server (Figure 5-54), and the useful throughput (Figure 5-55) at all experimented loads. In summary, the hybrid analytic model for the finite retry pattern gives good accuracy in all except the number of retries.

The errors follow the same pattern as for the infinite retry pattern and so probably have the same causes. The errors in the finite retry pattern are smaller that those in the infinite retry pattern (Figure 5-51). The errors in the locking holding time (Figure 5-52) are limited within 12% and the errors in web response time (Figure 5-54) are limited in 9%. These are all smaller than the errors in the infinite retry pattern.
Figure 5-50 Lock Acquisition Probability for Finite Retry Pattern (First Case)

Figure 5-51 Mean Number of Retries for the Finite Retry Pattern (First Case)
**Lock Holding Time (Finite Retry) - Disk**

![Graph showing lock holding time for different number of users.]

- **Figure 5-52 Mean Lock Holding Time for the Finite Retry Pattern (First Case)**

**Bottleneck Utilization (Finite Retry) - Disk**

![Graph showing bottleneck utilization for different number of users.]

- **Figure 5-53 Utilization of the Bottleneck (The Disk) for the Finite Retry Pattern (First Case)**

145
**Figure 5-54 Response Time of the Web Server for the Finite Retry Pattern (First Case)**

**Figure 5-55 Useful Throughput for the Infinite Retry Pattern (First Case)**
5.4.2 Second Case: Lock Bottleneck

The performance impact of a resource like the lock will be greater if it is the system bottleneck. In the previous examples, the bottleneck was at the disk and the lock was not very heavily used. This second set of cases is designed to make the lock itself the system bottleneck. The parameters of the hybrid model have been adjusted so that the lock will be saturated first as the number of customers increases. The thinking time $Z$ is 1 second. The database server spends 0.02 second CPU demand plus 1 disk operation in a query operation and 0.12 second CPU demand plus 6 disk operations in an update operation. The disk has 0.02 second device demand per operation. Other parameters are kept the same as the first case: disk bottleneck.

5.4.2.1 Second Case: Abort Pattern, Hybrid Model Results and Analysis

The hybrid model for the abort pattern was discussed in section 5.3.4. The two sub-models are shown in Figures 5-24 (LQN) and Figure 5-25 (GSPN), and the data exchange is given in Table 5-5.

In Figure 5-56 the decline of the lock acquisition probability is shown in the simulation results as the load is increased. The hybrid model reproduces the consistent trend with good accuracy in the lock acquisition probability. The similar comments on the accuracy apply to the bottleneck utilization (Figure 5-58). The hybrid model shows the accurate agreement in the response time of the web server at low and moderate load, but slightly larger errors at high load (Figure 5-59). The accuracy in the lock holding time (Figure 5-57) and the useful throughput (Figure 5-60) is worse than for the bottleneck utilization (Figure 5-58) and the lock acquisition probability (Figure 5-56). Generally, the hybrid analytic model of the abort pattern gives reasonable-to-excellent accuracy in all except the lock holding time, the useful throughput and the response time of the web server at the high load.
**Figure 5-56 Lock Acquisition Probability for the Abort Pattern (Second Case)**

**Figure 5-57 Mean Lock Holding Time for the Abort Pattern (Second Case)**
Figure 5-58 Utilization of the Bottleneck (The Lock) for the Abort Pattern (Second Case)
Figure 5-59 Response Time of the Web Server for the Abort Pattern (Second Case)

Figure 5-60 Useful Throughput for the Abort Pattern (Second Case)
5.4.2.2 Second Case: Infinite Retry Pattern, Hybrid Model Results and Analysis

The hybrid model for the infinite retry pattern was discussed in section 5.3.5. The two sub-models are shown in Figures 5-31 (LQN) and Figure 5-32 (GSPN), and the data exchange is given in Table 5-7.

Figure 5-61 shows the increase of the retries to acquire a lock as the load increases. The hybrid model predicted the trend of both performance results with reasonable accuracy in the number of retries when the load is low but the errors in the hybrid solution get larger when the load is moderate and high (Figure 5-61). In the simulation result, the lock holding time (Figure 5-62) increase a little when the load is low, then remains almost flat when the load is moderate and high (the number of customers is greater than 4 and the bottleneck utilization is greater than 70%). The hybrid model (Figure 5-62) shows the same trend, increasing gradually for low and moderate loads and being almost flat at higher loads. The errors are larger when load is high (Figure 5-62). The hybrid model shows good agreement in the bottleneck utilization (Figure 5-63) at all loads, and in the response time of the web server (Figure 5-64) and the useful throughput (Figure 5-65) at low and moderate load, but has larger errors in the response time of the web server (Figure 5-64) and the useful throughput (Figure 5-65) at high loads.

In summary, the hybrid analytic model for the infinite retry pattern gives reasonable-to-excellent accuracy in all except number of retries, the lock holding time, the response time of the web server, and the useful throughput at a high load. The pattern of errors again agrees with that observed in the first case, for the finite and infinite retry patterns.
Figure 5-61 Mean Number of Retries for the Infinite Retry Pattern (Second Case)
Figure 5-62 Mean Lock Holding Time for the Infinite Retry Pattern (Second Case)
Figure 5-63 Utilization of the Bottleneck (The Lock) for the Infinite Retry Pattern (Second Case)
Figure 5-64 Response Time of the Web Server for the Infinite Retry Pattern (Second Case)
5.4.2.3 Second Case: Finite Retry Pattern, Hybrid Model Results and Analysis

The hybrid model for the finite retry pattern was discussed in section 5.3.6. The two submodels are shown in Figures 5-38 (LQN) and Figure 5-39 (GSPN), and the data exchange is given in Table 5-9.

The decline of the lock acquisition probability (Figure 5-66) and the increase of the retries to acquire a lock (Figure 5-62) are shown as the load increases. The hybrid model reproduced the trend of the lock acquisition probability with good accuracy (Figure 5-66). The errors in the number of retries (Figure 5-67) are largest in the middle range of loads, and smaller at the extremes. In the simulation result, the lock holding time (Figure 5-68) increases linearly when the load is increased, and the hybrid model also shows the upward trend, but with increasing errors for moderate and heavy loads (Figure 5-68). The comment for the lock holding time is also applicable for the useful throughput (Figure 5-71). The hybrid model
demonstrated the accurate prediction in the bottleneck utilization (Figure 5-69) and the response time of the web server (Figure 5-70) at all the experimented loads. In summary, the hybrid analytic model for the finite retry pattern in the second case gives excellent accuracy in all except number of retries, the lock holding time, the response time of the web server, and the useful throughput at the high load. The pattern of errors is the same as the previous case.

**Lock Acquisition Probability (Finite Retry) - Lock**

![Graph showing lock acquisition probability](image)

**Figure 5-66** Lock Acquisition Probability for the Finite Retry Pattern (Second Case)
Figure 5-67 Mean Number of Retries for the Finite Retry Pattern (Second Case)

Figure 5-68 Mean Lock Holding Time for the Finite Retry Pattern (Second Case)
**Figure 5-69 Utilization of the Bottleneck (The Lock) for the Finite Retry Pattern (Second Case)**

**Figure 5-70 Response Time of the Web Server for the Finite Retry Pattern (Second Case)**
5.4.2.4 Second Case: Result Discussion

Overall, it is shown in the second case that the accuracy of the lock holding time (Figure 5-57, Figure 5-62, and Figure 5-68) has substantial errors in the hybrid model results. The estimate of the lock holding time is high from 6% to 20% in the abort pattern (Figure 5-57). It is also observed that the estimate of the lock acquisition probability is low (Figure 5-56 from 3% to 19%). On the other hand, the estimate of the failure probability is higher and causes the entire throughput to increase. The service time of the OpAbort is much less than that of the OpUpdate (in the 20 customer example, 0.0402 sec vs 0.467 sec). It would not correct the impact of overestimated lock holding time (Figure 5-57) to the web response time (Figure 5-59) and even accelerate the errors when the load is high (the number of customers is greater than 12). But for the retry patterns, the service time/demand of the sleep delay is some multiple of 0.4 sec, which is longer than the lock holding time (0.25-0.3 second). In these retry patterns, the estimate of the failure probability is higher but causes
the entire throughput to be lower. It explains the longer response time of the hybrid solution in Figure 5-64 (the infinite retry pattern) and Figure 5-70 (the finite retry pattern). The overestimation of the lock holding time shown in Figures 5-57, 5-62, and 5-68 result in the underestimation of the useful throughput shown in Figures 5-60, 5-65, and 5-71.

Thus the data in the hybrid models in the second case explain the results and the connection between them as above, but it is less clear why the lock acquisition probability is underestimated in the hybrid solution. The possible sources of the errors are listed here. In the LQN sub-model, the mean number of rendezvous from entry to entry is used to approximate the number of retries (in the retry patterns). The mean number of rendezvous also approximates the sequential dependency of the retries. The state-dependent lock acquisition decision is represented just by its probability in the LQN sub-model.

The surrogate delay has some complex distribution, but is used as the exponential distribution in the GSPN sub-model for saving the state space of the problem and seeking the efficient solution. Thus given the approximations, the exponential distribution approximation of the surrogate delay of the \textit{OpUpdate} is a plausible cause for the underestimated lock acquisition probability. Errors of this magnitude are not uncommon in analytical modeling.

Although the surrogate delay causes up to 30\% errors in the locking holding time shown in Figure 5-57, Figure 5-62, and Figure 5-68, which is not ideal, it is close enough for many purposes, particularly for early analysis. Thus the hybrid approach can provide the preliminary evaluation as the analytical modeling is most suitable for. When a system is running and the data can be collected, the sources for errors can be judged and the models can be tuned for better predictions.

The experiments have some limitations. They do not include cases with multiple decision points, or other types of exceptions depending on resource queue lengths, waiting times, or
utilization. Examples include timeouts on calls, exceptions due to memory or buffer overflows, and routing to the least busy processor.
6. Aspect-Oriented Performance Modeling

Chapter 5 addresses a single logical decision behaviour partition for exception handling. The solution estimates exception probabilities and retry counts in performance models. Exception handling is a common example of the crosscutting concerns that motivate Aspect-Oriented Programming (AOP) [Lippert00] [Filho06]. But aspects are typically woven into multiple locations in a program, and similarly there can be multiple logical-behaviour partitions on a computer system. To accommodate multiple partitions and to generalize HPMM, this chapter generalizes HPMM into Aspect-Oriented Performance Modeling (AOPM), with aspect sub-models for all kinds of logical decision behaviour partitions.

6.1 Generalizations

Pervasive behaviour as a crosscutting concern is common and can occur in multiple places in a computer system. It is termed as an aspect in Aspect-Oriented Programming (AOP) [Elrad01][Kiczales97] and in Aspect-Oriented Modeling (AOM) [France04]. Exception handling is a typical aspect [Lippert00][Filho06]. The resource exception handling discussed in chapter 5 may occur in multiple places in a computer system. Each resource exception handling involves a decision behaviour cell with dynamic performance. In the procedure of HPMM in chapter 4, those decision behaviour cells are combined into one behaviour fragment, which may lead to a state explosion issue for the GSPN sub-model. This chapter defines AOPM to use multiple GSPN sub-models with relatively small state spaces, and thus reduce the state space to avoid the state explosion issue.

Some research has been developed on weaving aspect models, then developing the performance model from the composed model. In [Petriu07a], the model composition is to integrate the context-specific aspect scenario model (activity diagram in UML) with the primary model (high level activity diagram) in the architecture view, the deployment view and the behaviour view. In [Petriu07b], the generic aspect advice models (Core Scenario
Model) are integrated into the primary model through composing the CSMs. Both papers present scenario-based weaving and consider the resource deployment for performance model development. Other aspect-oriented modeling researches also apply the scenario composition as a means for the model composition such as Aspect Oriented UCMs (AoUCMs) [Mussbacher07] through Use Case Maps. The same approach to weaving the aspect models to derive the composed model then further developing the performance model(s) is also applied in AOPM.

HPMM in chapter 4 addresses the relationship between the decision behaviour depending on dynamic performance data, and the rest of the system. Both parts are described in a UCM. This approach has been demonstrated through resource exception handling in chapter 5 by modeling the decision making behaviour in a GSPN sub-model and the rest in a LQN sub-model. When there are multiple resource exceptions in a system, the decision making behaviour occurs in multiple places and can be bound into the main behaviour as a behaviour aspect. Rather than manually model the behaviour of each exception as in chapter 5, the behaviour aspects for exceptions can be inserted into the UCM base model following the rules for UCMs proposed in AoUCM [Mussbacher07]. The approach of HPMM can be applied to partition the resulting UCM of the entire system by identifying decision cells within the aspect behaviours. HPMM is generalized as AOPM by modeling each aspect in a GSPN sub-model. Thus in a hybrid model of AOPM, multiple GSPN sub-models are constructed for aspect if necessary together with a LQN sub-model. Given the exception handling aspect, the results of the GSPN sub-models like probabilities or visit counts are inserted into the LQN sub-model just as in HPMM.

Another example apart from exceptions is a network delay aspect. The delay of a sub-model for the communication protocol such as TCP, can be plugged into the LQN sub-model. Thus with multiple sub-models for aspects, AOPM improves the scalability of the hybrid model and the reusability of sub-model construction for aspect.

AOPM is illustrated through a system with multiple exceptions:
• Aspects and composition: AOPM is concerned with aspects that deal with decision behaviour. This is a much narrower scope that aspects in AOP or AoUCM. However, the composition of the aspect behaviour and the base behaviour in AOPM uses the methods developed for AoUCM.

• Partitioning (Aspect Identification): Partitioning follows the algorithm for HPMM as described in section 4.2. It identifies cells in the composed UCM, that come from the aspect, as decision cells, and creates a behaviour fragment for each instance of weaving the aspect, labeled as being in focus for the state-based (GSPN) formalism.

• Sub-Model Construction: Given the behaviour fragments, sub-models are constructed following the steps of HPMM sub-model construction. The sub-scenarios are constructed with approximation and the sub-models are developed from sub-scenarios as a hybrid model in the AOPM. The sub-model for the aspect behaviour can be reused. The consistency of HPMM and its framework are applied to AOPM through the same construction principles and data exchange among sub-models. The same hybrid performance model solver is used to solve the aspect-oriented performance model.

6.2 Example of Composition and Aspect Identification

Aspects in AOPM inherit the features of aspects in AOP and AOM. An aspect in AOPM is the behaviour which can describe the crosscutting concern in the application. Moreover, the aspect in AOPM includes only the decision behaviour, which can be identified as a decision cell with high score so that GSPN is selected. Following AoUCM, the aspect behaviour can be modeled by elements of UCMs with the startpoint and endpoint. The UCM for the aspect behaviour is referred to as an advice map. The behaviour other than the aspect behaviour can be also modeled by UCMs, which is referred to as the base model. Thus the advice models are plugged into the base model at the join points to derive the composed model. The composition rules are defined in AoUCM [Mussbacher07], which are applicable in the behaviour composition of AOPM.
An example of a Hotel Booking System with resource allocation exceptions is used to illustrate the composition of advice maps and the base model in AOPM. The Hotel Booking System (HBS) allows customers to browse the availability of rooms and book a room if available. The HBS has the components as follows:

- **User**: HBS customer and browser;
- **WebServer**: the web portal to the HBS, offers room booking service;
- **Database**: the database server records information retrieved by requests from the web portal;

A customer can access the HBS to browse rooms by displaying the schedule and the room availability, and to book an available room. In a typical scenario a customer chooses an available room for a specific period and reserves it by updating data in the database. Once the room booking transaction is completed, the confirmation is sent back to the web server, which relays it back to the customer. The customer in the model (now representing a new person reserving a room) repeats the transaction after the thinking time. There are 10 processes on the web server to handle the requests from the customer and 5 processes on the database to handle the requests from the web server. Resource allocation exception handling is deployed for managing the software resource like the number of processes on the web server and the database server. The abort pattern described in section 5.2.2 is applied for exception handling. When the process is not available on the web server, it is reported the web failure. If the process cannot be obtained by the request on the database server, the database failure is shown up.

Following the definition of a UCM in AoUCM, the use case maps of HBS are developed in Figure 6-1. The base model includes one start point and multiple end points. Along the success path, responsibilities like **Thinking**, **BookRoom**, and **DbUpdate** are identified. If all processes of the web server are used up, the request performs the **WebFail** operation. The **DbFail** operation is conducted due to the unavailable database process. The advice map presents the map of the exception concern, which contains a pointcut stub. More specifically, this map includes the decision making as described in section 4.1. The decision
making is applicable for process allocation exception handling on both the web server and the database server. As presented in figure 6-1, the point cut map matches the responsibility BookRoom along the success path, which is along shown the base model.

**Base Model**

![Base Model Diagram]

**Advice Map**

![Advice Map Diagram]

**Figure 6-1 The AoUCM Model of HBS**

The composed map is shown in figure 6-2 following the composition rules of AoUCMs. The aspect stub is placed at the insertion point identified as diamonds with a letter “A” inside in the base model. The advice map can be plugged into the insertion points through a plug-in binding. The aspect sub-model presented as a UCM describes the relevant segment of the advice map. Thus two aspect stubs are identified and they represent the resource allocation exception handling on the web server and the database server respectively. Thus two plug-in bindings are shown as dashed arrows in Figure 6-2, which connect advice maps to insertion points in the base model. Therefore the composed map is derived and presented in Figure 6-2.

Based on the composed map, the behaviour analysis and partition are conducted in AOPM, which is the same as the behaviour partition in HPMM. The concept of behaviour cell and the scores for different formalism selection in HPMM can be applied in AOPM. The algorithm discussed in section 4.2 is followed in AOPM. Like HPMM, AOPM in this
research is also restricted to two formalisms: LQN and GSPN. Thus the scores of the common behaviour cells, identified in Table 4-1, are applicable in AOPM. A performance model formalism is selected to model behaviour cells according to these scores.

**Figure 6-2: The Composed Map of HBS**

HBS is used to demonstrate the behaviour partition in AOPM. Following the identification of the behaviour cell in section 4.2, all responsibility elements in the UCM in Figure 6-2 are defined as behaviour cells. Thus there are Thinking Responsibility Cell, BookRoom Responsibility Cell, and DbUpdate Responsibility Cell, and WebFail Responsibility Cell, and DbFail Responsibility Cell. The queueing behaviour for hardware resources gives the WebServer Queue Cell and the Database Queue Cell. The queueing behaviour for software resources like threads/processes gives the WebWorker Queue Cell and the DbWorker Queue Cell. Also the WebDecision Decision Cell is identified in the aspect stub, in which the decision is made upon the dynamic performance data such as the availability of the web server processes. A DbDecision Decision Cell has the dynamic performance data as the availability of the database server processes. All behaviour cells are shown in Figure 6-3. For each behaviour cell, the scores of two candidate performance modeling formalisms, LQN and GSPN, are listed in Table 6-1 derived from Table 4-1. Thus GSPN is selected for the WebDecision Decision Cell and the DbDecision Decision Cell and LQN is chosen for the other behaviour cells.
Composed Map

Figure 6-3 The Behaviour Cells of HBS
Table 6-1 Scores of Behaviour Cells for the HBS

<table>
<thead>
<tr>
<th>Behaviour Cells</th>
<th>GSPN</th>
<th>LQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thinking Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>BookRoom Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>DbUpdate Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>WebFail Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>DbFail Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>WebWorker Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>DBWorker Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>WebServer Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Database Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>WebDecision Decision Cell</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>DbDecision Decision Cell</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

6.3 Differences in Sub-Model Construction in AOPM

Aspect weaving often inserts aspects in multiple places, and AOPM enhances the sub-model construction process of HPMM in these cases to improve its scalability. Like HPMM, the sub-scenario of AOPM is based on behaviour fragment, which is derived from the behaviour partition of the composed model. In HPMM, there are $N$ behaviour fragments if $N$ performance modeling formalisms are selected. Consider the reusability of performance models in AOPM, an additional rule is added to enhance the construction process already developed in HPMM. If the selected performance modeling formalism of the behaviour cells is the same but the behaviour cells are located in different maps, the behaviour cells should be in separated behaviour fragments.
Once the behaviour fragments are determined, the approximation of the remaining behaviour fragments and further sub-scenario derivation follow from HPMM in section 4.4. The sub-model derivation however is further refined for AOPM. It is assumed that there are \( N \) behaviour fragments in total. And the behaviour fragment \( BF_1 \) is for the base model and the rest behaviour fragments \( BF_2, BF_3, \ldots, BF_N \) for the aspect. Thus the sub-scenario \( S_i \) is derived from the union behaviour \( B_i = BF_1 \cup (\bigcup_{j=2}^{N} BF_j^A) \) for the base model. The sub-scenario \( S_i \) is obtained from the union behaviour

\[
B_i = BF_1 \cup BF_1^A \cup (\bigcup_{j=2}^{i-1} BF_j^A) \cup (\bigcup_{j=i+1}^{N} BF_j^A), (i = 2, \ldots, N)
\]

for each advice map.

Following sub-scenario construction, the performance sub-models are developed from each sub-scenario. The performance sub-model construction and data exchange definition in AOPM are the same as those in HPMM. This chapter will not repeat this part.

6.3.1 Example Continued (Sub-Scenario Construction)

HBS in section 6.2 is used to illustrate the sub-scenario construction in AOPM. Section 6.2 has already identified that GSPN is for the WebDecision Decision Cell and the DbDecision Decision Cell and LQN is for the rest in AOPM. Following the identification of behaviour fragment in HPMM, behaviour fragment \( BF_1 \) including all the responsibility cells and all queue cells can be identified. Based on the extra rule specific in AOPM, WebDecision Decision Cell and DbDecision Decision Cell are in separate advice maps. They are in different behaviour fragments as \( BF_2 \) and \( BF_3 \). Thus all three fragments are identified as shown in Figure 6-4.
The sub-scenario $S_1$ is derived from the union of behaviour fragment $BF_1$ and the approximation of $BF_2$ and $BF_3$. The approximation technique, surrogate flow as probability, is applied for both $BF_2$ and $BF_3$. The $WebDecision$ Decision Cell in $BF_2$ can be abstracted as the probabilities $P_{web\_s}$ and $P_{web\_f}$ to acquire a process of the web server successfully or failed respectively. The probabilities $P_{db\_s}$ and $P_{db\_f}$ for successfully obtaining a process of the database server and failing doing it are derived from the abstraction of the $DbDecision$ Decision Cell in $BF_3$. The sub-scenario $S_1$, which binds the probabilities $P_{web\_s}$, $P_{web\_f}$, $P_{db\_s}$ and $P_{db\_f}$ along the paths in UCM, is shown in Figure 6-5. When the sub-scenario $S_1$ is developed as the LQN sub-model $M_1$, these probabilities are inserted as call parameters in the LQN sub-model.

**Figure 6-4 The Behaviour Partitions of HBS**
The sub-scenario $S_2$ is obtained based on the union between the behaviour fragment $BF_2$ and the approximation of $BF_1$ and $BF_3$. Like the abstraction of $BF_3$ in the sub-scenario $S_1$, probability $Pdb_s$ and $Pdb_f$ for whether a process of the database server is acquired are applied to the $DbDecision$ Decision Cell. The probabilities are shown in a UCM in Figure 6-6 (A). The surrogate delay approximation technique is used for $BF_1$ as the aggregations of a set of responsibilities shown in Figure 6-6 (A). There are aggregated responsibilities $WebPreDec^*$, $WebPostDecSuccess^*$ and $WebPostDecFail^*$ as surrogated delays. Thus the sub-scenario $S_2$ is derived as shown Figure 6-6 (B).
(A) Aggregations of UCM of HBS for sub-scenario $S_2$

(B) Sub-scenario $S_2$ of HBS

Figure 6-6 The Sub-scenario $S_2$ of HBS in AOPM
(A) Aggregations of UCM of HBS for sub-scenario $S_3$

(B) Sub-scenario $S_3$ of HBS

Figure 6-7 The Sub-scenario $S_3$ of HBS in AOPM

The sub-scenario $S_3$ is developed from the union of behaviour fragment $BF_3$ and the approximations of $BF_1$ and $BF_2$. Both $BF_2$ and $BF_3$ are from advice map. Thus the idea of constructing sub-scenario $S_2$ like approximation techniques can be reused in the construction of sub-scenario $S_3$. The abstraction of $BF_3$, probability $P_{web \_s}$ and probability $P_{web \_f}$ for success for the decision of the $WebDecision$ Decision Cell is derived from the
surrogate flow approximation technique. The probabilities $P_{web_s}$ and $P_{web_f}$ are shown in a UCM in Figure 6-7 (A). The same surrogate delay approximation technique is applied for $BF_1$ to derive $S_3$, like constructing $S_2$, this approximation technique is also used for $BF_1$ in sub-scenario $S_3$. The surrogate responsibilities are shown in Figure 6-6 (A) and they are $DbPreDec^*$, $DbPostDecSuccess^*$ and $DbPostDecFail^*$. Moreover, only the requests which have acquired the process of web server are allowed to access the $DbDecision$ Decision Cell. Thus the surrogate customer approximation technique should be applied for $BF_1$. The sub-scenario $S_3$ is shown Figure 6-7 (B).

The sub-scenarios $S_1$, $S_2$, and $S_3$ are constructed as in the above discussion and are shown in Figure 6-5, 6-6(B), and 6-7(B) respectively.

The performance sub-model construction follows the same principles as before (Section 4.3.3). A LQN sub-model is constructed from sub-scenario $S_1$, and GSPN sub-models are constructed for the web server resource exception (developed from sub-scenario $S_2$), and for the database resource exception (derived from sub-scenario $S_3$). For the data exchange, the probabilities obtained from the GSPN sub-model results are inserted into in the LQN sub-model as probabilities of call messages, and the surrogate delay and surrogate customers from the LQN sub-model results are used in the GSPN sub-models, as discussed in chapter 4.
7. Performance Modeling Solution for TCP

This chapter moves beyond exception-handling to apply HPMM to model communication delays under the Transmission Control Protocol (TCP). TCP poses many challenges to analytical performance modeling because of its complex logical rules, which makes it a good demonstration of HPMM. This chapter addresses the steady state performance of a single TCP connection within a simple network topology. This chapter describes hybrid performance modeling of TCP in a network with end to end connections in steady state. The hybrid performance models are validated through the simulation. The integration of TCP connections into a large software system as an aspect (woven into client-server calls) is discussed briefly at the end.

7.1 TCP Background

TCP is a core protocol in the Internet Protocol Suite [Tanenbaum02], and is widely used for reliable delivery of data in the correct order. When data is sent as a stream of packets across a network, packets may be delivered out of order, delayed due to contention for transmission links, or dropped due to buffer overflows. Dropped packets and excess delay lead to retransmission and potentially some packet may be duplicated. TCP detects these critical issues, requests retransmission of lost packets, deletes duplicate packets, rearranges out-of-order packets, and helps to minimize network congestion. Once the TCP receiver has reassembled a perfect copy of the data originally transmitted, it passes the data to the receiving application program. TCP is used for web access, E-mail, FTP, Secure Shell, Peer to Peer file sharing and some streaming media applications on the Internet [Tanenbaum02].

To tackle the congestion problem, different algorithms for congestion avoidance and control were proposed in [Jacobson88] as essential parts of TCP. They are: slow start, congestion avoidance, fast retransmission and fast recovery algorithms (see the book [Stevens93] for
more detailed information). These algorithms handle one connection between one sender and one receiver without taking into account other traffic, with the goal to minimize the reaction time to packet losses by means of congestion avoidance and to recover from packet loss efficiently by means of fast retransmission and fast recovery algorithms.

These algorithms are implemented in versions of TCP called TCP Tahoe, TCP Reno, TCP New-Reno, TCP Vegas, and TCP SACK, which are compared in papers [Lee01][Fall96]. It is hard to say that any one protocol performs best in all cases. This research is focused on TCP Reno due to following reasons: First it is widely adopted in TCP protocol. Second it employs four transmission phases: slow start, congestion avoidance, fast retransmission, and fast recovery and is a typical congestion control and avoidance protocol. Third the major part of TCP Reno protocol, slow start and congestion avoidance for flow control, are employed in other versions such as TCP Tahoe, TCP New-Reno, TCP Vegas, and TCP SACK. Thus there is an opportunity to extend the performance modeling approach of TCP Reno to other versions.

7.2 TCP Reno Algorithm

When a connection starts up or restarts after an initial TCP connection on an overwhelmed network, the TCP Reno connection should go through the phase called “slow start”. Slow start suggests that the sender sets the congestion window size \( cwnd \) to 2 segments. Then for each acknowledgement, the sender increases \( cwnd \) by 1 segment size (SS) so that the congestion window effectively doubles for every round trip (RT). Therefore in the first RT, one segment is sent, in the second, two segments and in the third, four segments. Thus \( cwnd \) increases exponentially in the slow start phase until either the slow-start threshold \( ssthresh \) is reached, or a loss (taken to indicate the congestion) is detected by the sender.

TCP Reno moves to its congestion avoidance phase after the slow start. Usually the first slow start ends after a loss since the initial \( ssthresh \) is large. The sender receives an acknowledgement after one or two segments are received. When the sender receives a
duplicate acknowledgement, it means the expected packet in the sequence has been delayed too long, or it has been lost. In congestion avoidance, the window size $cwnd$ is incremented by one per RT as long as non-duplicate ACKs are received. If multiple duplicate acknowledgements are received, it means this packet was lost since the sufficient time has passed and the packet should be received even through a long path. When triple duplicate (TD) acknowledgements are received, the $cwnd$ is halved and the lost segment is retransmitted. This is "fast retransmission" instead of waiting. If the acknowledgement times out, the TCP Reno resets to slow start state.

After the $cwnd$ is halved and the missing segment is resent, TCP Reno enters "fast recovery" phase. TCP Reno waits for an acknowledgment of the entire transmitted window and then returns to congestion avoidance. If there is no acknowledgment, TCP Reno experiences a timeout and enters the slow-start state. This timeout is termed as retransmission timeout (RTO). TCP Reno provides the initial value of RTO as $T_o$. RTO doubles to $2T_o$ if there is unsuccessful retransmission next time. This doubling is repeated for each unsuccessful retransmission until $64T_o$ is reached. After this, the timeout $RTO$ remains constantly at $64T_o$. The simplified description of the TCP Reno algorithm from RTT and $cwnd$ can be presented as follows to convey the understanding of TCP Reno. Figure 7-1 shows the congestion window size variation of TCP Reno.

1. After every non duplicate acknowledgement, TCP Reno works as:
   a. If $cwnd < ssthresh$, set $2cwnd \rightarrow cwnd$ for every RT in the slow start phase;
   b. If $cwnd > ssthresh$, set $cwnd + 1 \rightarrow cwnd$ for every RT in the congestion avoidance phase;

2. When the triple duplicated acknowledgement happen,
   a. The expected segment is retransmitted;
   b. set $cwnd/2 \rightarrow ssthresh$, then set $ssthresh \rightarrow cwnd$, which halves the window size;
   c. Resumes congestion avoidance once retransmission is acknowledged.
3. Once the acknowledgement of a transmission times out, TCP Reno goes to slow start:
   a. The expected segment is retransmitted;
   b. If $RTO < 64T_{aw}$, $2RTO \rightarrow RTO$;
   c. set initial $cwnd, 2 \rightarrow cwnd$;

**Figure 7-1 Congestion Window Size of TCP Reno**
7.3 TCP Reno Network

7.3.1 Topology and Behaviour of TCP Reno

The study network has the dumbbell topology shown in Figure 7-2, with source node $N_0$ and destination node $N_1$. The duplex link between $N_0$ and $N_1$ has capacity $C$ and propagation delay $PDT$. Each node has a FIFO buffer of size $B$. Each of $N$ FTP traffic generators creates a traffic flow via its own TCP agent attached to $N_0$ as the source. A connection is established from the source to a TCP sink agent at node $N_1$, which is the destination. The default size of a data packet that a TCP agent sends is 1KB. The size of the segment including TCP header, represented as $DATA$, is 1040B. Once a segment is received on the destination node $N_1$, a 40-byte acknowledgment, $ACK$, is generated and sent by the TCP sink agent, acknowledging $b \, DATA$ segments (default $b=1$). The size of an $ACK$ segment is 40B. The generator, source agent and TCP sink agent are numbered from 1 to $N$.

![Network Topology Diagram](image)

**Figure 7-2** THE NETWORK TOPOLOGY
Two run patterns were studied, a "short run" connection and a "long run" connection. A file for transmission on a short run connection has a finite number of *DATA* segments. While FTP is used as the application protocol over the TCP in a long run connection, the file size is unspecified and there are always data sent along the TCP. The long run connection describes the asymptotic behaviour of a connection with very large messages, like a streaming media application or a remote desktop connection on Windows.

TCP is established and the FTP is the application protocol over the TCP. In the short run connection, the TCP connection is torn down after the file transmission is finished. The FTP requests follow the Poisson distribution and the inter-arrival time is given as *INTERVAL*. Thus $N$ should be infinite or very large so that the transient behaviour of each connection can be repeated. In the long run connection, there are a fixed number of connections as $N$. Each connection has a live connection or the traffic generators always have data to transfer. Thus the very large file or the infinite data sources provide a steady stream of traffic so that the steady state behaviour can be analyzed.

Figure 7-3 shows the agents, node queues and message paths, considering the TCP agents as source and destination. The flows of DATA and ACK segments are shown by solid and dashed lines respectively. The transmitters are represented as a single server (a circle) with a buffer shown as a queue, and a rectangle represents a propagation delay (PDT). The transmission delay is made up of the service demand of the server, waiting in the buffer, and PDT. The propagation delay of a segment is independent of the segment size. The propagation delay for a segment and the service demand to transmit a segment are deterministic in the NS-2 simulation [Ns2].
The behaviour of the transmission of a DATA segment from the source to the destination is shown in Figure 7-4, using the same notation as Figure 7-3. Due to limited link capacity (represented by the server service rate), the buffer may become full, so that congestion occurs. Any arriving segment will be lost under FIFO drop-tail queueing [Yajnik99][Paxson99]. The packet loss makes the congestion window size vary. The dropped segment is regenerated and retransmitted. Therefore the behaviour of a segment includes the segment transmission and the corresponding acknowledgement if the segment is transmitted successfully, and regeneration and retransmission if it is dropped. From the high level view, all packets are ultimately delivered to the destination due to the reliability of TCP if the connection is torn down.
TCP Reno behaviour includes dynamic window size, dynamic timeouts, multiple phases like slow start, congestion avoidance, fast retransmit, and fast recovery as well as queueing, execution times (including transmission time), propagation delays and buffer overflow. To simplify this for analytic modeling, some assumptions and calculations will be described, for use in section 7.4. These assumptions and calculations are the same as [Padhye98][Yajnik99] with some corrections.

The terms and assumptions of the network and the file transfers are discussed. When TCP Reno is modeled, a term “round” is used. A round starts with the back to back transmission of $W$ segments, where $W$ is the congestion window size. All segments in a congestion window are sent and no other segment is sent until the first ACK segment is received. The
reception of ACK is marked as the end of this round and the beginning of the next round. Thus the duration of a round is independent of the window size. It is assumed that the segment loss is only due to a full buffer. The probability of the segment loss, $p$, is independent of $W$. It is also assumed that segment loss occurs only in the direction from the source to the destination, justified since the DATA flows in this direction are more congested than the ACK flows in the reverse direction. The DATA flows expect to suffer a few segment losses, or zero loss so that the connection is dominated by the connection establishment and the slow start phase in a short run connection and the steady state congestion avoidance phase in a long run connection. The average round trip delay between the source and the destination is defined as RTT. For the sake of simplicity, it is assumed there is no maximum congestion window size constraint.

The TCP delay is the sum of the delay of the “three way handshake” connection establishment, and of the data transfer. To facilitate the elaboration of TCP Reno calculation, the notations specific for TCP modeling and calculation are listed as below.

$DSS$: size of the DATA segment (1040B).

$SSS$: size of the SYNC segment (40B).

$ASS$: size of the ACK segment (40B).

$B$: buffer size.

$C$: the capacity of the link from node $N_0$ and node $N_r$.

$b$: number of segments one ACK segment acknowledges ($b = 1$).

$d$: number of data segments one file can be converted.

$RTT$: round trip time

$PDT$: propagation delay

$p$: the probability of segment loss

$j$: the index number of failures during the “three way handshake” connection establishment.

$P_{ce}(j)$: the probability of having $j$ failures in the three-way handshake.
$L_{CE}(j)$: the latency of connection establishment with $j$ failures in the three-way handshake connection.

$E[L_{CE}]$: the expected latency of the TCP connection establishment. $E[L_{DT}]$: the expected data transfer latency,

$T_{ss}$: the latency of the slow start phase.

$W_{ss}$: the congestion window size at the end of slow start phase.

$d_{ss}$: the data segments sent and received by the receiver before a loss occurs.

$i$: the index number of the round.

$cwnd_i$: the congestion window size of a specific round $i$.

$W_i$: the initial window size.

$SS\_data_i$: the amount of data sent by the end of slow start round $i$.

**Connection Establishment Calculations**

Every TCP connection starts with a "three way handshake" to establish the connection. The sender transmits a SYN segment and expects to receive an ACK segment within the SYN timeout, $T_s$. If the ACK is not received, the sender resends its SYN and waits for a response twice as long. This process is repeated at most 6 times until the sender receives an ACK. Once the sender receives an ACK, it moves to data transfer. The TCP implementations abort after 6 failures.

Let $j \geq 0$ be the times of the unsuccessful transmission from the sender in a connection. Let $P_{CE}(j)$ be the probability of having $j$ failures in the three-way handshake. Let $L_{CE}(j)$ be the latency of connection establishment with $j$ failures in the three-way handshake connection. Since segment losses are independent, $P_{CE}(j)$ follows a geometric distribution.
Thus the expected latency of the TCP connection establishment, $E[L_{CE}]$, can be estimated as

$$E[L_{CE}] = \sum_{j=0}^{\infty} (P_{CE}(j) \times L_{CE}(j)) \approx RTT + T_s \left( \frac{1-p}{1-2p} - 2 \right)$$ Equ. 7-3

**Data Transfer Calculations**

The data transfer starts from the generation of first data segment and ends with the reception of the acknowledgement of the last data segment. It is assumed that data are already ready for transmission or retransmission (data transfer does not include delay and resource consumption for generating and regenerating a data segment). The data transfer latency $E[L_{DT}]$, a function of $d$ data segments, can be decomposed into three steps: slow start phase, first segment loss, and transferring remaining data.

**Slow Start Phase**

The slow start phase begins after the connection establishment until a segment loss is detected. Let $T_{ss}$ be the latency of the slow start phase, $W_{ss}$ be the congestion window size at the end of slow start phase, and $d_{ss}$ be the data segments sent before a loss occurs. $E[d_{ss}]$ can be calculated, which is shown in Equ. 7-4.

$$E[d_{ss}] = \sum_{k=0}^{d-1} (1-p)^k \times p \times k + (1-p)^d \times d = \frac{(1-(1-p)^d)(1-p)}{p}$$ Equ. 7-4
In the slow start phase, TCP Reno increases the congestion window, \( cwnd \), by 1 per ACK. In each round the sender gets approximately \( cwnd/b \) ACKs (default \( b = 1 \) as stated in section 7.3.1). Let \( cwnd_i \) denote the congestion window for the round \( i \). Thus, the exponential growth of \( cwnd_i \) in the slow start phase can be shown as Equ. 7-5.

\[
cwnd_{i+1} = cwnd_i + cwnd_i/b = (1 + 1/b)cwnd_i = \gamma * cwnd_i = \gamma^i * cwnd_i, (\gamma = 2) \tag{7-5}
\]

If a sender starts with an initial window size as \( W_i \) segments, then \( SS_{\text{data}_i} \), the amount of data sent by the end of slow start round \( i \), can be closely approximated as

\[
SS_{\text{data}_i} = W_i + W_i * \gamma + W_i * \gamma^2 + \cdots + W_i * \gamma^{i-1} = W_i * \frac{\gamma^i - 1}{\gamma - 1} \tag{7-6}
\]

Thus from Equ. 7-6, \( i \) is given by Equ. 7-7.

\[
i = \log_\gamma \left( \frac{SS_{\text{data}_i}(\gamma - 1)}{W_i} + 1 \right) \tag{7-7}
\]

So \( W_{ss} \) in Equ. 7-8 can be derived from Equ. 7-7 by replacing \( SS_{\text{data}_i} \) with \( d \) (\( d \) is the number of data segments).

\[
W_{ss}(d) = W_i * \gamma^{i-1} = W_i * \left( \frac{d(\gamma - 1)}{W_i} + 1 \right) * \gamma^{i-1} = \frac{d(\gamma - 1)}{\gamma} + \frac{W_i}{\gamma} \tag{7-8}
\]

From Equ.7-4 and Equ. 7-8, the expected window size of \( W_{ss} \) can be obtained as Equ. 7-9.

\[
E[W_{ss}] = \frac{E[d_s](\gamma - 1)}{\gamma} + \frac{W_i}{\gamma} \tag{7-9}
\]

Based on Equ. 7-7, the time to send \( E[d_s] \) data segments in the slow start phase is calculated as Equ. 7-10. In Equ. 7-10, RTT represents the expected round trip time.

\[
E[T_{ss}] = RTT * \log_\gamma \left( \frac{E[d_s](\gamma - 1)}{W_i} + 1 \right) \tag{7-10}
\]
First Segment Loss

The slow start phase ends with the detection of a segment loss or the end of the data. The probability of this segment loss occurrence in slow start can be calculated as

\[ t_{ss} = 1 - (1 - p)^d \]  
Equ. 7-11

There are two ways to detect the segment losses: retransmission timeout (RTOs) and triple duplicate loss indication (TD). The probability of detecting a segment loss with an RTO, \( Q(p, W) \), is calculated as Eq. 7-12 from [Padhye98]. Assuming all losses are detected, the probability to detect a loss via TD is \( 1 - Q(p, W) \).

\[ Q(p, W) = \min\left(1, \frac{(1 + (1 - p)^3(1 - (1 - p)^{w-3}))}{(1 - (1 - p)^w)/(1 - (1 - p)^3)}\right) \]  
Equ. 7-12

The expected delay of an RTO is derived in [Padhye98] as Eq. 7-13.

\[ E[Z^{TO}] = \frac{G(p)T_0}{1 - p} \]  
Equ. 7-13

\( T_0 \) is the duration of the first timeout in a sequence of one or more successive timeouts. \( G(p) \) is given by

\[ G(p) = 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6 \]  
Equ. 7-14

Thus following [Cardwell00], the expected cost for any RTOs and the fast recover from TD at the end of the slow start phase is given as Eq. 7-15 with the same assumption in [Padhye98]

\[ T_{loss} = t_{ss} \times (Q(p, E[W_{ss}]) \times E[Z^{TO}] + (1 - Q(p, E[W_{ss}]) \times RTT) \]  
Equ. 7-15
Transferring Remaining Data

The actual amount of the remaining data depends on where in the window the loss occurs, how many segments are lost, the congestion window size at the time of loss, and the recover. The amount of data $E[d_{ca}]$, left after the slow start phase and any loss recover is approximated as Equ. 7-16.

$$E[d_{ca}] = d - E[d_{ss}]$$  \hspace{1cm} \text{Equ. 7-16}

When $E[d_{ca}] > 0$ segments left to send, the throughput of transferring $E[d_{ca}]$ is given in [Padhye99]. The throughput is denoted as $R$, which is calculated in Equ. 7-17.

$$R = \frac{1 - p + W_{ca}(p)}{p} + \frac{Q(p, W_{ca}(p))}{2} \frac{\text{RTT}(bW_{ca}(p) + 1) + \frac{Q(p, W_{ca}(p))G(p)}{1 - p}}{1 - p}$$  \hspace{1cm} \text{Equ. 7-17}

There are $Q(p, W)$ in Equ. 7-12, $G(p)$ in Equ. 7-14, and $W_{ca}(p)$, which is the expected window size of the congestion avoidance calculated in Equ. 7-18.

$$W_{ca}(p) = \frac{2 + b}{3b} + \sqrt{\frac{8(1 - p)}{3bp} + \left(\frac{2 + b}{3b}\right)^2}$$  \hspace{1cm} \text{Equ. 7-18}

Thus the expected latency of sending the remaining data $E[d_{ca}]$ is calculated in Equ. 7-19 by Little's theorem. The expected latency of sending the remaining data is represented by $E[T_{ca}]$.

$$E[T_{ca}] = E[d_{ca}] / R$$  \hspace{1cm} \text{Equ. 7-19}

Like the statement in the beginning of this part, the expected latency in data transfer, $E[L_{DT}]$, is the sum of all steps. Therefore,
\[ E[L_{DT}] = E[T_{ss}] + E[T_{loss}] + E[T_{ca}] \]  

Equ. 7-20

7.3.3 Why is a Hybrid Modeling Solution Needed

The analytical performance modeling solution is considered for modeling TCP Reno. The short run connection and the long run connection discussed in section 7.3.1 employ TCP Reno for the file transfers. Both involve the connection establishment and the data transfer including multiple phases like slow start, congestion avoidance and fast recovery. As shown in section 7.3.2, the performance measures such as the latency of file transmission and the segment transmission rate depends on the congestion window size. The congestion window varies its size to control the flow of segments according to different phases and the segment loss occurrence. Congestion window size depends on the segment loss probability \( p \), which is a performance measure of the bottleneck link. The segment loss is caused by the full buffer in the TCP link, which is the decision cell with dynamic performance data. Therefore a hybrid model is needed as a performance modeling solution.

The hybrid modeling solution discussed in this chapter can be applied to modeling a bottleneck link, in which the segment loss occurs. The TCP Reno connection is established for the short run connections and the long run connection on the bottleneck link. The assumptions for both connections are summarized as follows:

- There always are sufficient data to send for the sources as defined in the short and long run connections;
- Both the short run connection and the long run connection can saturate the link from \( N_0 \) and \( N_i \);
- The model solution is focused on the steady state of the network.
  - In the short run connection, the file size for each connection should not be too large and the inter-arrival time, \( INTERVAL \) should be small. Thus both
are able to keep the network steady; The connection establishment and the slow start phase dominate the behaviour of TCP Reno;

- The number of flows $N$ is large enough to keep the network steady for the long run connection. The congestion avoidance phase dominates the behaviour of TCP Reno.

- Losses are due only to congestion as the buffer is full but the segment loss probability should be low;
- ACKs are never dropped before the transmission;
- All the dropped segments are correctly identified for retransmission and there is no cost for regenerating;

### 7.4 Sub-Scenario Constructions

Following the procedure of hybrid performance modeling, the behaviour cells are identified with the ranked performance modeling formalism from behaviour analysis and the model formalisms selection. The sub-scenario model is constructed from the behaviour fragment derived from behaviour partition.

![Figure 7-5 The Use Case Map of TCP Reno Segment Behaviour](image-url)
The segment behaviour in TCP Reno is described through a Use Case Map [Buhr96] and a UCM derived from Figure 7-4 is shown in Figure 7-5. This UCM includes two paths: one is Success Path and the other is Abort Path. Along each path, there are responsibilities like segment transmission \textit{Transmission01}, segment propagation \textit{Propagation01}, ACK transmission \textit{Transmission10}, ACK propagation \textit{Propagation10}, and retransmission delay \textit{LostDetectionDelay}. The decision is based on the performance data like whether the buffer is full. The decision for each segment as \textit{BufferDecision} leads different paths like Success path and Abort path. The scenario contains the segment transmission component \textit{DataTrComponent} and the ACK transmission component \textit{AckTrComponent}. Responsibility \textit{Transmission01} is located on the component \textit{DataTrComponent}. And Responsibility \textit{Transmission10} sits on the component \textit{AckTrComponent}. The start point is that the segment is generated or regenerated if it was lost before. The end point is that the sender receives the acknowledgement of that segment.

Behaviour analysis and partition can be conducted on the UCM in Figure 7-5. Following the identification of the behaviour cells in HPMM in section 4.2, all the UCM elements like responsibilities in Figure 7-5 like are defined as behaviour cells. Thus there are responsibility cells \textit{Transmission01}, \textit{Propagation01}, \textit{Transmission10}, \textit{Propagation10}, and \textit{LostDetectionDelay}. The queueing components are identified as queue cells \textit{DataTrComponent}, and \textit{AckTrComponent}. The decision can be isolated as a decision cell \textit{BufferDecision}.

Following the recommended scores listed in Table 4-1, the scores of responsibility cells are given as medium for LQN and GSPN. The scores for queue cells are high for LQN and low for GSPN. The score of the \textit{BufferDecision} Decision cell is high for GSPN and low for LQN. The scores are listed in Table 7-1.
Table 7-1 Scores of Behaviour Cells of TCP Reno Segments Behaviour

<table>
<thead>
<tr>
<th>Behaviour Cells</th>
<th>GSPN</th>
<th>LQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission01 Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Propagation01 Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Transmission10 Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Propagation10 Responsibility Cell</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>DataTrComponent Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>AckTrComponent Queue Cell</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>BufferDecision Decision Cell</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

From Table 7-1, both GSPN and LQN are suitable for all Responsibility Cells, but LQN is better. GSPN has to be selected for BufferDecision Decision Cell. Following the discussion in section 4.2, LQN is selected to model all Queue Cells and Responsibility Cells. Thus the LQN behaviour fragment $BF_1$ includes all responsibility cells and all queue cells in Table 7-1 and the BufferDecision Decision Cell in the GSPN behaviour fragment $BF_2$. Figure 7-6 shows behaviour partition through the dashed line named as $BF\_Boundary$ and two behaviour fragments $BF_1$ and $BF_2$. 
Each behaviour fragment and the abstraction of other fragments can be combined to further develop a scenario model. For the given LQN formalism, $BF_1$ is united with the abstraction of $BF_2$, $BF_2^A$, as $B_1$. The BufferDecision Decision Cell is abstracted as two probabilities in $BF_2^A$. These probabilities are the segment loss probability and the segment transmission probability. The probabilities encapsulate the involved responsibilities, the buffer availability detection and the path selection between abort path and Success path in the BufferDecision Decision Cell. The probabilities impact the load of the system like regenerating and retransmitting segments. The LQN scenario model, $S_1$, is derived in Figure 7-7 from $B_1$.

For GSPN formalism, $BF_2$ is united with the abstraction of $BF_1$, $BF_1^A$, as $B_2$. The Transmission01 Responsibility Cell in $BF_1^A$ is kept the same as in $BF_1$. The remaining behaviour cells are abstracted as different flows at the start point of the UCM. The GSPN scenario model, $S_2$, is obtained from $B_2$ and $S_2$ is shown in Figure 7-8. The GSPN scenario model focuses on the segment loss model.

**Figure 7-6 Behaviour Partition of Segment Behaviour**
TCP Reno includes complicated behaviour beyond the presentation of the UCM in Figure 7-5. The dynamic congestion window size could be exponential growing in the slow start phase and increment by one in congestion avoidance phase. The congestion window size is halved once a loss is detected. Thus the dynamic congestion window size demonstrates the saw tooth pattern in Figure 7-1. The dynamic congestion window size with different grow patterns is a self adaptive mechanism to provide the reliable transmission by retransmission while TCP takes advantage of bandwidth aggressively. When a segment is sent, the
congestion window size is adjusted according to the ACK of the previous segment and the round trip. The segment loss means an extra segment is generated to be enqueued and the load is adjusted. Therefore all these complexities like batching transmission within dynamic congestion window size, retransmission of segment due to loss, timeout or delay for retransmission are encapsulated as the traffic loads in the performance model.

Considering different congestion window size growth patterns in the slow start phase and the congestion avoidance phase, the hybrid performance modeling solutions for the long run connection and the short run connection are treated separately. In the short run connection, the data was sent in the slow start phase and can be completed through a couple of round trip cycles in congestion avoidance phase if there are segment losses. Thus the majority behaviour is dominated in the slow start phase. The latency of the behavior is more interesting. In the long run connection, the data could be extremely large or the connection is always alive. The data transmission turns to be a long process and the congestion avoidance phase dominates the behaviour. The segment loss probability would be more interesting. The hybrid performance modeling solution for the long run connection is discussed in section 7.5 and for the short run connection in section 7.6.

7.5 Hybrid Performance Modeling Solution for Long Run Connection

This section is focused on hybrid performance modeling solution for the steady state of long run connection with TCP Reno. The performance sub-models are developed from the LQN scenario model (Figure 7-7) and the GSPN scenario model (Figure 7-8).
In the long run connection, the congestion avoidance phase dominates the behaviour of TCP Reno. A TD period (TDP) is defined as a period between two TD loss indications. Thus Figure 7-9 shows the evolution of congestion window size in TDPs. In paper [Padhye98], the relationship between the congestion window size and segment loss probability $p$ can be calculated for the steady state of the long run connection with TCP Reno. The calculation about the expected congestion window size of TCP Reno of long run connection is presented as Equ. 7-18 and the congestion window size impacts the throughput of the TCP Reno long run connection.

Several approximations and abstractions are applied during sub-model construction. Triple-duplicate loss indication is the only loss indication used in the long run connection. It is assumed that the chance with timeout loss is low when segment loss probability $p$ is small enough. As segment loss probability increases, timeout loss occurs multiple times, which tears down the connection. This case would be out of the scope of the steady long run connection. All behaviour like regenerating and retransferring the lost segments is encapsulated into the expected congestion window size in the congestion avoidance phase and the loss probability $p$. Both the average congestion window size and the loss probability $p$ mutually impact each other (see Equ. 7-18). The scenario of each ACK from being generated by the destination to being received by the source is abstracted as a pure delay for
simplicity. This matches the TCP NS-2 simulation model. In the simulation model, the transmission and propagation delays of *DATA* and *ACK* are deterministic. In this simple network, it is assumed that there is no traffic other than the *ACK* flow from the destination node \( N_i \) to the source node \( N_0 \). Thus there is no segment loss in the *ACK* flow. All these make the simplicity of the *ACK* behaviour as a pure delay reasonable and feasible.

The LQN sub-model is focused on the window control mechanism of TCP Reno, which includes regenerating the lost segments with no cost, varying the congestion window size and aborting the segments when the buffer is overflowed. There is a set of \( N \) flows. Each flow sends segments by means of average congestion window size. Some segments go through abort path and the others go through success path. The abort path and success path are shown in Figure 7-7. The congestion window size is calculated through formula shown in equation Equ. 7-18. The segment loss probability \( p \) is derived from the result of the GSPN sub-model.

![Figure 7-10](image)

**Figure 7-10 High-level View of the TCP Hybrid Model for the Long Run Connection**

The GSPN sub-model is focused on the segment loss model. The GSPN sub-model models a set of \( N \) flows. All flows send segments by means of average congestion windows. There
would be the arrival rate contention for the data buffer. The arrival rate is calculated from triple duplicate period and average congestion window size derived from the LQN sub-model.

Therefore a high level view of TCP long run connection hybrid performance is defined in Figure 7-10. Rectangles are deployed to represent sub-models and parallelograms are used to stand for the data exchange.

7.5.1 The GSPN sub-model

![Diagram of GSPN sub-model](image)

**Figure 7-11 The GSPN sub-model for the long run connection**

The GSPN sub-model for segment loss shown in Figure 7-11 is developed from the scenario model in Figure 7-8. The tokens in place $P_{\text{Buffer}}$ represent the size of the available buffer in the format as the number of segments in the link from node $N_0$ and node $N_r$.

The timed transition $TT_{\text{Arrival}}$ model the arrival rates of all $N$ flows as segments per second during the triple duplication period ($TDP$). The timed Transition $TT_{\text{Node01}}$ models the transmission of $DATA$ segments from node $N_0$ and node $N$, derived from Responsibility $Transmission01(N_0, N)$ in the UCM in Figure 7-8. The transition rate of the timed transition $TT_{\text{Arrival}}$ is $1/(DSS*8/C)$ defined in Equ. 7-21 and this is a single server.
The transition rates of the timed transitions \(TT_{Arrival}\) and \(TT_{Node01}\) are specified in Equ. 7-21 and Equ. 7-22 respectively. Each flow has the load defined as \(W_{ca}(p)/TDP\) based on Little's theorem. Thus Equ. 7-22 defines the transition rates of the timed transitions \(TT_{Arrival}\). TDP is derived from the LQN sub-model and \(W_{ca}(p)\) is from Equ. 7-18, passed back from the LQN sub-model.

\[
\mu(TT_{Node01}) = 1/(DSS*8/C) \quad \text{Equ. 7-21}
\]

\[
\mu(TT_{Arrival}) = W_{ca}(p)*N/TDP \quad \text{Equ. 7-22}
\]

7.5.2 The LQN Sub-model

In the LQN sub-model, a set of \(N\) flows are abstracted as sending segments through the average congestion window size. The computation of average congestion window size is derived from the equation Equ. 7-18, using segment loss probability \(p\) obtained from the result of the GSPN sub-model. It is assumed that the average congestion window size is \(size1\) \((size1 = i)\). LQN model could not support to model the buffer size distribution but uses the segment loss probability \(p\) along the Success path and the Abort path shown in Figure 7-7.

The LQN sub-model for a specific congestion window size \(size1\) is shown in Figure 7-12. Each flow is sourced at an entry \(E_{Think}\) of the set of \(N\) tasks \(T_{Flows}\) (on \(N\)-processor \(P_{Clients}\)) and sends segments along the link from node \(N_0\) and node \(N_i\). The entry has zero thinking time since there are always data to be sent continuously. To gather the service time for sending the congestion window size segments, delay period for triple duplicate loss detection, acknowledgement behaviour, and propagation delay, an infinite pseudo task \(T_{Pseudo1}\) with an entry \(E_{Ps1\_1}\) is introduced. To gather the service time for the transmission time of the congestion window size segment and waiting in the buffer, excluding other behaviour, a second set of an infinite pseudo task \(T_{Pseudo2}\) and an entry \(E_{Ps2\_1}\) is included. Since the pseudo task \(T_{Pseudo2}\) with entry \(E_{Ps2\_1}\) is to gather the
service time of segments, which are enqueued in the buffer, they are attached to the success path.

The responsibilities defined in Figure 7-7 are modeled by the LQN sub-model. Along the physical link from node N_0 and node N_1, only one segment is processed at a time, so there is only one task \( T_{\text{Link01}} \) with entry \( E_{\text{T01}_1} \) (modeling responsibility \( \text{Transmission01(N0, N1)} \)) and one processor \( P_{\text{Link01}} \). The congestion window size, \( \text{size1} \) is inserted as the call messages to entry \( E_{\text{T01}_1} \). A single infinite pool \( P_{\text{Servers}} \) is defined for all infinite tasks executed run on it. The responsibility \( \text{Propagation01(N0, N1)} \) is modeled as a pure delay represented as infinite tasks \( T_{\text{Link01P}} \) with entry \( E_{\text{P01}_1} \). Responsibility \( \text{Transmission10(N1, N0)} \) and responsibility \( \text{Propagation10(N1, N0)} \) for ACK segments are approximated as a pure delay for simplicity. Thus the Responsibility \( \text{Transmission10(N1, N0)} \) and responsibility \( \text{Propagation10(N1, N0)} \) for ACK segments are modeled as an infinite task \( T_{\text{Ack}} \) with entry \( E_{\text{Ack}_1} \).

When segments are lost due to a full buffer, the abort path retransmits them after a loss indication. The delay in detecting the loss indication, represented as responsibility \( \text{DelayRetransmission} \), is pure delay. Responsibility \( \text{LossDetectionDelay} \), is modeled as an infinite task \( T_{\text{Drop}} \) with the entry \( E_{\text{LDD}} \). All infinite tasks \( T_{\text{Pseudo1}}, T_{\text{Pseudo2}}, T_{\text{Link01P}}, T_{\text{Ack}}, \) and \( T_{\text{Drop}} \) are located on the infinite processor \( P_{\text{Servers}} \). The number of calls to entry \( E_{\text{LDD}}, P_{\text{Drop}_1} \), is just the segment loss probability \( p \) for the abort path. The number of calls to entry \( E_{\text{Ps2}_1} \) is \( \text{Padmit}_1 \) (Equ. 7-23) which is the probability for the segments enqueued into the buffer.

\[
\text{Padmit}_1 + \text{Pdrop}_1 = 1 \quad \text{Equ. 7-23}
\]
Generally the average congestion window size is not an integer. The LQN sub-model in Figure 7-12 considers only integer window sizes. A non-integer value can be realized by providing two window sizes \( size_1 \) and \( size_2 \) which straddle the mean value by choosing them with probabilities \( p_1 \) and \( p_2 \) (\( p_2 = 1 - p_1 \)) respectively. The LQN sub-model is revised in Figure 7-13 for this, with separate entries for the two window sizes. Probabilities \( p_1 \) and \( p_2 \) are the number of calls to entries \( E_{Ps1,1} \) and \( E_{Ps1,2} \) in pseudo tasks \( T_{Pseudo1} \) from the pure client task \( T_{Flow} \). Given a mean window size \( W_{ca} \) and two predefined integer sizes \( size_1 < W_{ca} < size_2 \), the relationship between \( p_1 \) and \( p_2 \) is shown in Equ. 7-24 and 7-25.

\[
p_1 + p_2 = 1 \quad \text{Equ. 7-24}
\]

\[
size_1 * p_1 + size_2 * p_2 = W_{ca} (p) \quad \text{Equ. 7-25}
\]

FIGURE 7-12 THE LQN SUB-MODEL OF THE LONG RUN CONNECTION WITH A SPECIFIC WINDOW SIZE
The probabilities $p_1$ and $p_2$ are calculated as Equ. 7-26 and Equ. 7-27.

\[
p_1 = \frac{\text{size}2 - W_{ca}(p)}{\text{size}2 - \text{size}1} \quad \text{Equ. 7-26}
\]

\[
p_2 = \frac{W_{ca}(p) - \text{size}1}{\text{size}2 - \text{size}1} \quad \text{Equ. 7-27}
\]

Along the success path, entries $E_{Ps1_1}$ and $E_{Ps1_2}$ in pseudo tasks $T_{Pseudo1}$ have calls to entries $E_{Ps2_1}$ and $E_{Ps2_2}$ in Task $T_{Pseudo2}$. The numbers of calls are $Padmit_1$ and $Padmit_2$ respectively. The entry $E_{LDD}$ could be replicated in the infinite task $T_{Drop}$ when the average congestion window size is straddled into two congestion window sizes. However one entry $E_{LDD}$ can be used since both cases have the same delay. The LQN sub-model is shown in Figure 7-13. The relationship between $P_{drop_1}$ and $Padmit_1$ (Equ. 7-23) is applied between $P_{drop_2}$ and $Padmit_2$ (see Equ. 7-28).

\[
Padmit_2 + P_{drop_2} = 1 \quad \text{Equ. 7-28}
\]

In Equ. 7-23 and Equ. 7-28, $Padmit_1$, $Padmit_2$, $P_{drop_1}$, and $P_{drop_2}$ are unknown variables. Every segment in a sliding window is equivalent and they have the same loss probability $p$. $Padmit_1$, $Padmit_2$, $P_{drop_1}$, $P_{drop_2}$ are defined as blow.

\[
Padmit_1 = (1 - p) \quad \text{Equ. 7-29}
\]

\[
Padmit_2 = (1 - p) \quad \text{Equ. 7-30}
\]

\[
P_{drop_1} = p \quad \text{Equ. 7-31}
\]

\[
P_{drop_2} = p \quad \text{Equ. 7-32}
\]
Calculating from the performance results of the LQN sub-model, Equ. 7-33 is used to derive \( TDP \), which is used in Equ. 7-23.

\[
TDP = \text{servicetime}(E_{Ps1_1}) \times p1 + \text{servicetime}(E_{Ps1_2}) \times p2
\]  

Equ. 7-33

7.5.3 Data Exchange between the LQN Sub-model and the GSPN Sub-model

The data exchange between the GSPN sub-model and the LQN sub-model is summarized as it follows:
The data exchange between the GSPN sub-model and the LQN sub-model shown in Table 7-2 is defined. The hybrid performance model solution for the long run connection of TCP Reno includes the LQN sub-model in Figure 7-13 and the GSPN sub-model in Figure 7-11. The LQN sub-model imports the data through equations Equ. 7-26, Equ. 7-27, Equ. 7-29, Equ. 7-30, Equ. 7-31, and Equ. 7-32 for \( p_1, p_2, \text{Padmit}_1, \text{Padmit}_2, \text{Pdrop}_1, \) and \( \text{Pdrop}_2 \). The LQN sub-model exports the data through equations Equ. 7-23 to the GSPN sub-model.

The congestion window sizes \( \text{size1} \) and \( \text{size2} \) must be defined before the hybrid model is solved, with some values satisfying \( \text{size1} < \text{size2} \), and ideally straddling the actual average value. A rather wide range, from about 2/3 of 4/3 of the expected average, was used, with the initial segment loss probability \( p \) as 1%, \( p_1 \) and \( p_2 \) as 50% and 50% respectively. These values will be adapted by the iteration. If the estimated mean trends to one end of the range \( (\text{size1}, \text{size2}) \) then the iteration should be stopped and the range should be shifted or extended in that direction.
7.5.4 Validation and Analysis

To validate the analytical hybrid performance modeling solution for the long run connections, experiments were performed with \( N \) in the range 10 to 30 flows (when the number of flows is less than 10, there is no segment loss in the simulation and the average window size of the congestion avoidance phase in Eqn. 7-18 is not applicable). The physical link is 100% utilized. The percentage of successful transmissions is validated by the NS-2 simulation.

The percentage of successful transmissions from the hybrid performance model and the NS-2 simulation result are tabulated in Table 7-3. They show that, as the number of flows increases, there is more contention for the buffer and segment transmission and lower percentages of successful transmissions. The analytical hybrid performance model predictions have the errors within 1% compared to the simulation results.

As the number of flows is further increased to be greater than 30, time out losses are more frequent and would have to be included in the model. Eventually this would cause the tear-down of the connection, so it would not be in a steady state. These high-loss cases are out of the scope of this model, which is intended to model routine TCP usage.
Table 7-3 Comparison of Percentages of Successful Transmissions between Hybrid Model Solution and NS-2 Simulation in the Long Run Connection

<table>
<thead>
<tr>
<th>N</th>
<th>Simulated Percentage of Successful Transmissions</th>
<th>Analytical Percentage of Successful Transmissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>99.594%</td>
<td>99.087%</td>
</tr>
<tr>
<td>12</td>
<td>99.469%</td>
<td>98.850%</td>
</tr>
<tr>
<td>14</td>
<td>99.088%</td>
<td>98.660%</td>
</tr>
<tr>
<td>16</td>
<td>98.699%</td>
<td>98.470%</td>
</tr>
<tr>
<td>18</td>
<td>98.483%</td>
<td>98.280%</td>
</tr>
<tr>
<td>20</td>
<td>98.310%</td>
<td>98.154%</td>
</tr>
<tr>
<td>22</td>
<td>98.070%</td>
<td>98.017%</td>
</tr>
<tr>
<td>24</td>
<td>97.884%</td>
<td>97.911%</td>
</tr>
<tr>
<td>26</td>
<td>97.358%</td>
<td>97.700%</td>
</tr>
<tr>
<td>28</td>
<td>97.119%</td>
<td>97.558%</td>
</tr>
<tr>
<td>30</td>
<td>96.873%</td>
<td>97.463%</td>
</tr>
</tbody>
</table>

7.6 Hybrid Performance Modeling Solution for a Short Run Connection

This section describes a hybrid model for a short run connection of TCP Reno. The performance sub-models are also derived from the LQN scenario model in Figure 7-7 and the GSPN scenario model in Figure 7-8.

When a small file is transmitted, it gives a short run connection, whose behaviour includes connection establishment and the initial stages of data transfer. Once the connection is established, the slow start phase follows. If a segment loss occurs, the connection moves to
the congestion avoidance phase. The calculations in section 7.3.2 are used to derive the latency of the transmission of the short run connection.

Approximations and abstractions are applied during sub-model construction. From the transmission behaviour described in section 7.2, it can be abstracted as a batch operation which sends \textit{cumul} segments in one RTT. Thus all behaviour cells of the behaviour fragment \( BF_1 \) in focus by LQN are modeled as the delay as counts of RTTs and RTOs. The LQN sub-model is a delay model. The behaviour cell of the behaviour fragment \( BF_2 \) in focus by GSPN is to derive the segment loss probability. The GSPN sub-model is as a segment loss model.

The LQN sub-model computes the delay from the establishment of the connection to the disconnection. The segment loss probability \( p \) and the delay of RTTs are derived from the GSPN sub-model results. The GSPN sub-model describes segment losses in the \textit{DATA} flows and the \textit{SYN} flows. The surrogate flow in section 4.4.3.2 is used in the GSPN modeling. The arrival rate of \textit{SYN} and the arrival rate of \textit{DATA} are computed from the LQN sub-model result.

A high level view of the TCP analytical hybrid performance model for the short run connection is given in Figure 7-14, in which rectangles represent sub-models and parallelograms are used to stand for the data which is exchanged.
The GSPN sub-model for segment loss is defined in Figure 7-15 based on the scenario model in Figure 7-8. Responsibility $Transmission_{01}(N0, N1)$ is mapped to the transmission
along the link from node $N_0$ and node $N_i$. There are SYN segments and DATA segments. The transmission of SYN segments is defined as a timed transition $TT\_SYN\_Node01$ with transition rate $1/(SSS*8/C)$. A timed transition $TT\_DATA\_Node01$ for the transmission of DATA segments has transition rate $1/(DSS*8/C)$. The load is modeled through two timed transitions $Est\_Arrival$ and $Data\_Arrival$. The timed transition $Est\_Arrival$ models the total arrival rate of SYN segments including the regenerated ones and the timed transition $Data\_Arrival$ for the total arrival rate of DATA segments and regenerated ones. The immediate transitions $IT\_EDrop$ and $IT\_SYN$ model the decision to drop or enqueue SYN segments. The immediate transition $IT\_F Drop$ and $IT\_FTP$ are for dropping or enqueueing DATA segments. A set of places like Place $P\_Est$, Place $P\_FTP$, Place $P\_Buf$, Place $P\_SYN$, Place $P\_Data$, and Place $P\_Link$. The initial tokens of Place $P\_Buf$ are the buffer size $B$. The initial token in Place $P\_Link$ is one, which indicates there is only one segment transferred through the link from node $N_0$ and node $N_i$ at a time. The arc from timed transition $Est\_Arrival$ to Place $P\_Est$ and the one from $Data\_Arrival$ to $P\_FTP$ show multiplicity.

The transition rates of the timed transitions $TT\_FTP\_Node01$, $TT\_SYN\_Node01$, $Est\_Arrival$ and $Data\_Arrival$ are specified in Equ. 7-34, Equ. 7-35, Equ. 7-36 and Equ. 7-37 respectively.

\[
\mu(TT\_FTP\_Node01) = 1/(DSS*8/C) \quad \text{Equ. 7-34}
\]
\[
\mu(TT\_SYN\_Node01) = 1/(SSS*8/C) \quad \text{Equ. 7-35}
\]
\[
\mu(Est\_Arrival) = f_{SYN} \quad \text{Equ. 7-36}
\]
\[
\mu(Data\_Arrival) = f_{DATA} \quad \text{Equ. 7-37}
\]

$f_{SYN}$ in Equ. 7-36 and $f_{DATA}$ in Equ. 7-37 are derived from the result of the LQN sub-model.

In the result of the GSPN sub-model, the segment loss probability $p$ is derived as Equ. 7-38. RTT for both SYN segment and DATA segment are calculated as SRTT and DRTT (see Equ. 7-39 and 7-40).
\[ p = \frac{f(TT_{EDrop}) + f(TT_{FDrop})}{f(TT_{SYN}) + f(TT_{FTP}) + f(TT_{EDrop}) + f(TT_{FDrop})} \]

Eqn. 7-38

\[ SRTT = SSS \times 8/C + \text{Mark}(P_{SYN}) \times SSS \times 8/C + \text{Mark}(P_{Data}) \times DSS \times 8/C + 2 \times PDT + ASS \times 8/C \]

Eqn. 7-39

\[ DR7T = DSS \times 8/C + \text{Mark}(P_{SYN}) \times SSS \times 8/C + \text{Mark}(P_{Data}) \times DSS \times 8/C + 2 \times PDT + ASS \times 8/C \]

Eqn. 7-40

7.6.2 The LQN Sub-Model

The LQN sub-model estimates the delay for connection establishment and data transmission, which are calculated through the number of rounds and timeouts discussed in section 7.3.2. In a given congestion window, the transmission of segments is abstracted as a batch operation in each round. However LQN cannot model the batch operation per segment, so the further abstraction is applied to all behaviour cells defined in Figure 7-7. The processing of an entire batch, sending segments and receiving ACKs, is abstracted as RTT along the success path shown in Figure 7-7. This delay hides all the TCP Reno behaviour like dynamic congestion window size, automatically generated segments after a loss, and connection establishment. The timeouts for SYN segment and DATA segment are modeled in the LQN sub-model along the abort path of the scenario model in Figure 7-7. The LQN sub-model also collects the expected latencies defined in Eqn. 7-3 and Eqn. 7-20.
To facilitate the discussion of the LQN sub-model construction, Equ. 7-3 and Equ. 7-20 are rewritten as Equ. 7-41 and Equ. 7-42 having the coefficients of round trip time and timeout for both SYN and DATA segments as $\alpha_s$, $\beta_s$, $\alpha_d$, and $\beta_d$. These coefficients are derived from Equ. 7-3 and Equ. 7-20 based on segment loss probability $p$ in Equ. 7-38.

\[
E[L_{CE}] = \alpha_s SRTT + \beta_s T_s \\
E[L_{DT}] = \alpha_d DRTT + \beta_d T_0
\]

Figure 7-16 The LQN Sub-Model for the Short Run Connection

A set of tasks is defined in the LQN sub-model. The reference task, $T_{Arr}$, is defined on the processor $P_{Clients}$ with the thinking time as arrival interval time. A pseudo task of data collection is $T_{PseudoF}$ to collect the expected transmission time of a message, $T_{PseudoS}$ to collect the expected latency of establishing a TCP connection, and $T_{PseudoD}$ to collect the expected latency of transferring data. Each pseudo task has an entry with zero processor demand. Task $T_{RTT}$ has entries $E_{SRTT}$ and $E_{DRTT}$, which have the processor demand.
demands like the expected RTT of SYN segment and DATA segment, SRTT and DRTT. Task $T_{TO}$ is to model the timeout of the SYN segment and the DATA segment with entries $E_{STO}$ and $E_{DRTO}$. Entries $E_{STO}$ and $E_{DRTO}$ have the initial time out of the SYN segment and DATA segment. All pseudo tasks like $T_{PseudoF}$, $T_{PseudoS}$, and $T_{PseudoD}$ and task $T_{RTT}$ and task $T_{TO}$ are on an infinite processor $iServers$.

The entry-to-entry calls include an asynchronous call from $E_{Rate}$ to $E_{FTP_F}$ (frequency $= 1$), and synchronous calls from $E_{FTP_F}$ to $E_{S_SYN}$ (Rgs calls) and to $E_{S_DATA}$ (Rgd calls). There are synchronous calls for the SYN segment from $E_{S_SYN}$ to $E_{SRRT}$ (Rts calls) along the success path, and to $E_{STO}$ (Ds calls) along the abort path, and synchronous calls for the DATA segment from $E_{S_DATA}$ to $E_{DRRT}$ (Rtd calls) along the success path, and to $E_{DRTO}$ (Dd calls) along the abort path. The count of calls, Rgs, Rgd, Rts, Rtd, Ds, Dd, are defined through Equ. 7-43, Equ. 7-44, Equ. 7-45, Equ. 7-46, Equ. 7-47, and Equ. 7-48,

\[
Rgs = 1/(1 - p) \quad \text{Equ. 7-43}
\]
\[
Rgd = 1/(1 - p) \quad \text{Equ. 7-44}
\]
\[
Rts = (1 - p)\alpha_s \quad \text{Equ. 7-45}
\]
\[
Rtd = (1 - p)\alpha_d \quad \text{Equ. 7-46}
\]
\[
Ds = (1 - p)\beta_s \quad \text{Equ. 7-47}
\]
\[
Dd = (1 - p)\beta_d \quad \text{Equ. 7-48}
\]

In the result of the LQN sub-model, the rate of generating SYN segments $f_{SYN}$ and the rate of generating DATA segments $f_{DATA}$ can be derived.

\[
f_{SYN} = \text{throughput}(T_{PseudoC}) \quad \text{Equ. 7-49}
\]
\[
f_{DATA} = \text{throughput}(T_{PseudoD}) \quad \text{Equ. 7-50}
\]
7.6.3 Data Exchange between the LQN sub-model and the GSPN sub-model

The data exchange between the GSPN sub-model and the LQN sub-model is summarized as it follows:

Table 7-4 Data Exchange between the GSPN Sub-model and the LQN Sub-model for the TCP Reno Short Run Connection

<table>
<thead>
<tr>
<th>GSPN Sub-Model Def. from LQN Sub-Model</th>
<th>LQN Sub-Model Def. from GSPN Sub-Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu(TT_FTP_Node01) ) Equ. 7-34</td>
<td>SRTT Equ. 7-39</td>
</tr>
<tr>
<td>( \mu(TT_SYN_Node01) ) Equ. 7-35</td>
<td>DRTT Equ. 7-40</td>
</tr>
<tr>
<td>( \mu(Est_Arrival) ) Equ. 7-36</td>
<td>( p ) Equ. 7-38</td>
</tr>
<tr>
<td>( \mu(Data_Arrival) ) Equ. 7-37</td>
<td>Rgs Equ. 7-43</td>
</tr>
<tr>
<td></td>
<td>Rgd Equ. 7-44</td>
</tr>
<tr>
<td></td>
<td>Rts Equ. 7-45</td>
</tr>
<tr>
<td></td>
<td>Rtd Equ. 7-46</td>
</tr>
<tr>
<td></td>
<td>Ds Equ. 7-47</td>
</tr>
<tr>
<td></td>
<td>Dd Equ. 7-48</td>
</tr>
</tbody>
</table>

The hybrid performance model for the short run connection of TCP Reno includes the LQN sub-model in Figure 7-16, the GSPN sub-model in Figure 7-15, and the data exchange in Table 7-4. The LQN sub-model imports the data through Equ. 7-39, Equ. 7-40, Equ. 7-43, Equ. 7-44, Equ. 7-45, Equ. 7-46, Equ. 7-47, and Equ. 7-48 for SRTT, DRTT, Rgs, Rgd, Rts, Rtd, Ds, and Dd through the segment loss possibility \( p \) derived from the GSPN sub-model and the calculated coefficient \( \alpha_s, \beta_s, \alpha_D, \) and \( \beta_D \). The LQN sub-model also exports the data through Equ. 7-49 and Equ. 7-50 to the GSPN sub-model for the arrival rate of the timed transition \( Est\_Arrival \) and \( Data\_Arrival \).
7.6.4 Validation and Analysis

To validate the analytical hybrid performance modeling solution of the short run connection, the short run size and the interval arrival time of the short runs should be considered and they have the joint impact on the load. As we know, the majority of the traffic over the Internet constitutes of file transfers. The average transferred file is around 10KB. This research uses the average size \( d = 10 \). To get the reasonable interval arrival time, there is an equation (Equ. 7-51) to calculate and the \( \text{INTERVAL} \) is greater than 0.33408 \((C=250\text{kb})\).

\[
(DSS \times d + SSS) \times 8 / \text{INTERVAL} < C
\]

Equ. 7-51

The interval arrival time is varied from 0.34 second to 0.72 second by a step 0.02 second (When the interval arrival time increases above 0.68, there is no segment loss in both simulation and analytical solution and no further investigation.). The segments of the short run are transmitted or retransmitted to the destination and acknowledged. The time to transmit a message is defined from the generation of the first \( \text{SYN} \) segment to the reception of the \( \text{ACK} \) segment for the last \( \text{DATA} \) segment. The percentage of successful transmissions and the transmission latency of the hybrid performance model are validated by the NS-2 simulation result.

The percentage of successful transmissions and latency of the analytical hybrid performance model results and the NS-2 simulation result are tabulated in Table 7-5. As the interval arrival time increases (from 0.340 to 0.720), the increment of the percentage of successful transmissions is shown in the simulation results. The hybrid model reproduces the trend with good accuracy in the percentage of successful transmissions. On the other side, as the interval arrival time is varied, it hints that the load decreases and there is less contention to the buffer. Thus the segment loss probability declines and it would take less time (transmission delay) to transfer a short message. This is demonstrated in the simulation result. The hybrid analytical model shows the same trend. But the agreement in the transmission delay is not as good as that in the percentage of successful transmissions,
because the transmission delay heavily depends the segment loss probability and the queue length. Like the transmission delay computation discussed in section 7.3.2, the delay is sensitive to the loss probability. The errors in the percentage of successful transmissions are less than 1% except the 0.340 interval case. If the segment loss probability is examined, the errors are escalated. Thus the errors of the percentage of successful transmissions are augmented in the errors of transmission delay.

Overall the errors are that the hybrid performance solution over-estimates the transmission delay of the file transfer. This is caused by the nature of the problem. When the NS-2 simulation is used, the delay is deterministic. Given the DATA segment size 1040B along the path of the link from node N₀ and node N₁ with the capacity 250Kb, the transmission delay of the DATA segment is always 0.03328, which has deterministic distribution. However the GSPN model could not model this deterministic delay in the timed transition and the substitution is an exponential distribution for this delay. Also the same substitution also happened to the SYN segment. These substitutions contribute the error like overestimated delay in the file transmission.
Table 7-5 Comparison of Percentages of Successful Transmissions and Delay between Hybrid Model Solution and NS-2 Simulation in the Short Run Connection

<table>
<thead>
<tr>
<th>Inter-Arrival</th>
<th>Simulated Percentage</th>
<th>Analytical Percentage</th>
<th>Simulated Transmission Delay (Sec)</th>
<th>Analytical Transmission Delay (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.720</td>
<td>100%</td>
<td>100%</td>
<td>2.430</td>
<td>2.53309</td>
</tr>
<tr>
<td>0.700</td>
<td>100%</td>
<td>100%</td>
<td>2.456</td>
<td>2.56538</td>
</tr>
<tr>
<td>0.680</td>
<td>100%</td>
<td>99.99%</td>
<td>2.463</td>
<td>2.60141</td>
</tr>
<tr>
<td>0.660</td>
<td>100%</td>
<td>99.99%</td>
<td>2.476</td>
<td>2.64185</td>
</tr>
<tr>
<td>0.640</td>
<td>100%</td>
<td>99.99%</td>
<td>2.464</td>
<td>2.68749</td>
</tr>
<tr>
<td>0.620</td>
<td>100%</td>
<td>99.99%</td>
<td>2.504</td>
<td>2.73936</td>
</tr>
<tr>
<td>0.600</td>
<td>100%</td>
<td>99.98%</td>
<td>2.502</td>
<td>2.79872</td>
</tr>
<tr>
<td>0.580</td>
<td>100%</td>
<td>99.98%</td>
<td>2.550</td>
<td>2.86718</td>
</tr>
<tr>
<td>0.560</td>
<td>100%</td>
<td>99.97%</td>
<td>2.552</td>
<td>2.94681</td>
</tr>
<tr>
<td>0.540</td>
<td>100%</td>
<td>99.96%</td>
<td>2.591</td>
<td>3.04027</td>
</tr>
<tr>
<td>0.520</td>
<td>100%</td>
<td>99.94%</td>
<td>2.644</td>
<td>3.14887</td>
</tr>
<tr>
<td>0.500</td>
<td>100%</td>
<td>99.92%</td>
<td>2.702</td>
<td>3.28027</td>
</tr>
<tr>
<td>0.480</td>
<td>100%</td>
<td>99.88%</td>
<td>2.696</td>
<td>3.43888</td>
</tr>
<tr>
<td>0.460</td>
<td>100%</td>
<td>99.84%</td>
<td>2.755</td>
<td>3.63223</td>
</tr>
<tr>
<td>0.440</td>
<td>99.99%</td>
<td>99.77%</td>
<td>2.910</td>
<td>3.87042</td>
</tr>
<tr>
<td>0.420</td>
<td>99.98%</td>
<td>99.67%</td>
<td>3.124</td>
<td>4.1666</td>
</tr>
<tr>
<td>0.400</td>
<td>99.94%</td>
<td>99.51%</td>
<td>3.414</td>
<td>4.53802</td>
</tr>
<tr>
<td>0.380</td>
<td>99.96%</td>
<td>99.29%</td>
<td>3.869</td>
<td>5.00662</td>
</tr>
<tr>
<td>0.360</td>
<td>99.51%</td>
<td>98.97%</td>
<td>4.867</td>
<td>5.59414</td>
</tr>
<tr>
<td>0.340</td>
<td>93.0%</td>
<td>98.49%</td>
<td>11.78</td>
<td>6.33599</td>
</tr>
</tbody>
</table>
7.7 Strategy to Apply Hybrid Performance Modeling Solution of TCP Reno for a Large System

FTP is representative of internet applications using TCP and is used to demonstrate the hybrid modeling approach in the communication network. The hybrid performance modeling solutions of both the long run connection and the short run connection in FTP can be applied for modeling the other applications like web service, email, remote administrations. These applications require reliable data stream service.

The long run connection is the connection staying alive for a long time. Thus the hybrid model discussed in section 7.5 can be applied for the applications like remote administrations like remote desktop connection on windows, VNC, secure shell, and streaming media applications.

The short run connection is the more common connection with the short life cycle. The delay of the file transmission along short run connection is more interesting. The hybrid model in section 7.6 is used to model the applications like HTTP requests in web service, email and etc.

In a software system, calls for service may be made using TCP connections (this is the case in web services, for instance). Given a base model for the system in the LQN formalism, the communications behaviour using the TCP protocol can be defined as an aspect and the approach in Chapter 6 can be applied. The TCP behaviour is woven into each call that uses TCP, with appropriate parameters for the network used by the call, the behaviour partitions are identified, and sub-models are constructed. The sub-models for the calls with TCP could have the same structure as the sub-models identified here. The duration of the TCP short run connection can be inserted into the base model.
8. Conclusions and Future Work

This thesis presents a hybrid performance modeling methodology which is systematically introduced in chapter 4. The methodology is illustrated through four aspects: scope, system partitioning, consistent sub-model construction, and solution tool development. The hybrid performance modeling methodology is further generalized to aspect-oriented performance modeling in chapter 6 for the aspects of the decision behaviour. The hybrid performance modeling methodology is applied for the performance modeling of different exception handling cases in chapter 5 and of the network communication connections with the TCP Reno protocol in chapter 7.

8.1 Conclusions

The Hybrid Performance Modeling Methodology (HPMM) has been created to extend Layered Queuing Network (LQN) to model resource-acquisition exceptions, and similar behaviour involving decisions based on the performance variables such as delay and resource utilization. Without HPMM, the exception probability must be assumed, which may ignore the important performance impacts of exceptions.

Although hybrid methods have often been used before [Beilner89] [Pooley92] [Fricks96] [Moorsel98], the systematic and correct construction of sub-models has not been addressed before this research. In HPMM, the systematic approach of the consistent sub-model construction (section 5.5) is employed. This approach includes four techniques: surrogate delay, probabilities and visit counts, and surrogate customers. Having the selected formalisms according to the score of behaviour cells, the entire behaviour of the system can be partitioned into behaviour fragments. The mean values of the elements like delay, throughput, and population are used for the approximated elements. The consistency has been kept naturally since each sub-model (from the scenario model) is consistent with the whole behaviour.
The power of the LQN formalism has been enhanced through HPMM. LQN is extended to model the decision making as the lock probabilities and the visit counts in chapter 5 and the segment loss probability in chapter 7, which are calculated from GSPN.

The scalability of the GSPN formalism has been improved in HPMM. Using the surrogate delays (in chapter 5) derived from the LQN sub-model, the state space of the GSPN sub-model can be reduced and the sub-model can be solved within a reasonable duration. HPMM is generalized as AOPM through multiple GSPN sub-models for the pervasive behaviour. Instead of one GSPN sub-model in HPMM, multiple GSPN sub-models can be developed for locations of an aspect with the smaller state space. The scalability of the GSPN formalism is maintained in AOPM.

HPMM as combining LQN and GSPN in this research provides adequate accuracy performance evaluation of a system with decision making. The performance evaluation is demonstrated through the resource allocation exception handling cases (section 5.4) and the network communication cases with TCP Reno (section 7.5.4 and section 7.6.4). However errors are substantial. Especially errors in the decision probability like the lock acquisition probability and the delay like the lock holding time are up to 15% in section 5.4.1 and 30% in section 5.4.2. The approximations of HPMM could be the sources of the errors.

HPMM is a practical and useful addition to the modelers' toolbox. Compared to single formalism solution, HPMM provides the correct modeling (without assumptions) and the solvable solution (scalability) for the computer systems with decisions on performance data.
8.2 Generalization of HPMM

This research has discussed one generalization of HPMM as AOPM in chapter 6. HPMM is a general approach which can combine different model formalisms. HPMM can be applied without any change to performance modeling in the situations as:

- Any kinds of sub-models, which are not limited to the decision model (chapter 4)
- Any number of sub-models, which is shown as in AOPM (chapter 6);
- Any formalisms, as long as the solver tools of which have the suitable interface to the tool defined in section 4.7.3.

The benefit of such generalizations would depend on the cases. The example of generalization like AOPM would improve the scalability of the underlying GSPN formalism. Other benefits could be like extending the capability of the formalisms and solving the problem which is unsolvable by single formalism alone.

8.3 Limitation of HPMM

Currently LQN is extended with hybrid sub-models and the idea of HPMM is implemented. The limitations of the LQN extension are:

- The accuracy is limited by the underlying tools. Eg. The underestimated disk waiting time in LQNS results in the underestimation of response time of the web in the hybrid model (section 5.4.1.1). Also, the exponential distribution of the delay is used in GSPN but it has other distribution in the real world (section 7.6.4)
- The accuracy is limited by the use of mean values. The distributions and dependencies are ignored in the approximation.
- Scalability constraint: Although there is an improvement from GSPN and HPMM, the effect of GSPN scalability limitations was noticed in some cases (a long solution time).
Reference


