



Performance Requirements Verification During Software Systems Development

Using UML model Transformation Approach

Abdullatif M. AlAbdullatif

PhD Candidate

School of Mathematical and Computer Science

Computer Science Department

Supervisor

Prof. R. J. Pooley

The copyright in this thesis is owned by the author. Any quotation from the thesis or use of any of the information contained in it must acknowledge this thesis as the source of the quotation or information.

Requirements verification refers to the assurance that the implemented system reflects the specified requirements. Requirement verification is a process that continues through the life cycle of the software system. When the software crisis hit in 1960, a great deal of attention was placed on the verification of functional requirements, which were considered to be of crucial importance. Over the last decade, researchers have addressed the importance of integrating non-functional requirement in the verification process. An important non-functional requirement for software is performance. Performance requirement verification is known as *Software Performance Evaluation*. This thesis will look at performance evaluation of software systems. The performance evaluation of software systems is a hugely valuable task, especially in the early stages of a software project development. Many methods for integrating performance analysis into the software development process have been proposed. These methodologies work by utilising the software architectural models known in the software engineering field by transforming these into performance models, which can be analysed to gain the expected performance characteristics of the projected system.

This thesis aims to bridge the knowledge gap between performance and software engineering domains by introducing semi-automated transformation methodologies. These are designed to be generic in order for them to be integrated into any software engineering development process. The goal of these methodologies is to provide performance related design guidance during the system development. This thesis introduces two model transformation methodologies. These are the improved state marking methodology and the UML-EQN methodology. It will also introduce the UML-JMT tool which was built to realise the UML-EQN methodology. With the help of automatic design models to performance model algorithms introduced in the UML-EQN methodology, a software engineer with basic knowledge of performance modelling paradigm can conduct a performance study on a software system design. This was proved in a qualitative study where the methodology and the tool deploying this methodology were tested by software engineers with varying levels of background, experience and from different sectors of the software development industry. The study results showed an acceptance for this methodology and the UML-JMT tool. As performance verification is a part of any software engineering methodology, we have to define frame works that would deploy performance requirements validation in the context of software engineering. Agile development paradigm was the result of changes in the overall environment of the IT and business worlds. These techniques are based on iterative development, where requirements, designs and developed programmes evolve continually. At present, the majority of literature discussing the role of requirements engineering in agile development processes seems to indicate that non-functional requirements verification is an uncharted territory. CPASA (Continuous Performance Assessment of Software Architecture) was designed to work in software projects where the performance can be affected by changes in the requirements and matches the main practices of agile modelling and development. The UML-JMT tool was designed to deploy the CPASA Performance evaluation tests.

II

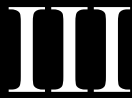
Dedication

In the Name of God Most Grateful Most Merciful

I dedicate this thesis to my family, especially...

My Wife Ebtehaj who is true to her name, the source of my happiness. I thank her for all the support, patience and understanding.

My Parents who supported me materially and morally.



Acknowledgements

Professor Robert J. Pooley has been the ideal thesis supervisor. His sage advice, insightful commentary and patient motivation aided the writing of this thesis. I would also like to thank **Dr. Peter J.B. King** whose steadfast support was greatly needed and deeply appreciated.

This work has been carried out at the department of computer sciences at the school of mathematical and computer science. I would like to thank all the personnel of the department and school for their appreciated support.

Finally, I would like to thank King Saud University in Riyadh, Saudi Arabia, for providing me with the scholarship which supported me financially during my stay here in Edinburgh.

IV

Table of Contents

CHAPTER 1: INTRODUCTION	1
1.1 SOFTWARE PERFORMANCE EVALUATION: WHAT?	1
1.2 SOFTWARE PERFORMANCE EVALUATION: WHY/WHY NOT?	3
1.3 SOFTWARE PERFORMANCE EVALUATION: FINDING THE SOLUTION	3
1.4 CONTRIBUTIONS AND MAJOR RESULTS ACHIEVED	4
1.5 THESIS OUTLINE	7
CHAPTER 2: INFORMATION SYSTEM ENGINEERING.....	10
2.1 SOFTWARE ENGINEERING PARADIGMS	10
2.1.1 <i>Conventional Software Engineering</i>	12
2.1.2 <i>Agile Development Methodology</i>	16
2.1.3 <i>Conventional Vs. Agile Development Methodologies</i>	18
2.2 REQUIREMENTS ENGINEERING.....	19
2.2.1 <i>Gathering and Representing Requirements</i>	20
2.2.2 <i>Validating/ Verification of Requirements</i>	22
2.3 UML SYSTEM AND DATA MODELLING	25
2.3.1 <i>Use-case Diagram</i>	26
2.3.2 <i>Sequence Diagram</i>	26
2.3.3 <i>Deployment Diagram</i>	28
2.3.4 <i>Example: Video Search System</i>	28
2.4 SOFTWARE ENGINEERING MODELLING AND CASE TOOLS.....	31
2.4.1 <i>Drawing Tools</i>	31
2.4.2 <i>CASE and iCASE Tools</i>	32
2.5 CASE TOOL MODEL REPRESENTATION	32
2.5.1 <i>UML XMI Representation</i>	33
2.5.2 <i>Working with an XMI Document</i>	37
2.6 SUMMARY.....	39
CHAPTER 3: SOFTWARE PERFORMANCE EVALUATION	41
3.1 SOFTWARE PERFORMANCE STUDY: MODELLING AND EVALUATION	41
3.1.1 <i>System Abstraction and Performance Models</i>	42
3.1.2 <i>Performance Model Evaluation</i>	44

3.1.3 Performance Analysis	45
3.2 PERFORMANCE EVALUATION: SIMULATION.....	45
3.2.1 Simulation Study Steps.....	46
3.2.2 Simulation: For and Against.....	47
3.3 PERFORMANCE EVALUATION: ANALYTICAL MODELLING	47
3.3.1 Markov Chains	48
3.3.2 Stochastic Petri-Nets.....	52
3.3.3 Queuing Networks	54
3.4 OTHER PERFORMANCE EVALUATION TECHNIQUES.....	62
3.4.1 Process Algebra.....	62
3.4.2 Workload Modelling	64
3.5 PERFORMANCE MODELLING FOR SYSTEM DESIGN	66
3.6 PERFORMANCE MODEL EVALUATION TOOLS.....	68
3.6.1 JMT Queuing Network Solution Tools.....	71
3.6.2 JMT Queuing Network Analysis Tools.....	72
3.7 SUMMARY.....	72
CHAPTER 4: INTEGRATING PERFORMANCE EVALUATION IN SOFTWARE ENGINEERING	74
4.1 SOFTWARE PERFORMANCE ENGINEERING.....	75
4.1.1 PASA.....	77
4.1.2 SPE Methodology.....	78
4.2 PERFORMANCE ENGINEERING IN AGILE DEVELOPMENT	80
4.2.1 CPASA.....	81
4.2.2 CPASA at Work.....	84
4.3 PERFORMANCE EVALUATION OF SYSTEM ARCHITECTURE	86
4.3.1 Evaluating the Methodologies	88
4.3.2 UML to Performance Model Methodologies	90
4.4 SUMMARY.....	92
CHAPTER 5: AN APPLICATION OF THE STATE MARKING METHODOLOGY	94
5.1 THE STATE MARKING METHODOLOGY	95
5.1.1 System Representation	95
5.1.2 State Marking With GSPN.....	96
5.1.3 State Marking With Markov Chains.....	97
5.2 EXTENDING THE METHODOLOGY	98
5.2.1 Input Model Representation	100
5.2.2 Output Performance Model.....	101
5.2.3 Extracting the Performance Model.....	102
5.2.4 Example: Web Video Application.....	106
5.2.5 Evaluating the methodology.....	109

5.3 REALISATION OF THE METHODOLOGY	110
5.3.1 MARCA Package	111
5.3.2 Performance Model Building Tool	111
5.4 SUMMARY.....	113
CHAPTER 6: UML–EQN METHODOLOGY	114
6.1 EXPLAINING THE METHODOLOGY	114
6.1.1 The Methodology Steps	116
6.1.2 Explanation Example	117
6.2 PERFORMANCE PARAMETERS CAPTURE.....	119
6.2.1 Performance Parameter Required	120
6.2.2 Performance Data Card	124
6.2.3 Example	125
6.3 CONSTRUCTING THE SOFTWARE MODEL	127
6.3.1 Communication Maps.....	127
6.3.2 Communication Map Construction	128
6.4 CONSTRUCTING THE MACHINE MODEL	130
6.5 FINALISING THE PERFORMANCE MODEL	132
6.5.1 Defining Job Classes	132
6.5.2 Connecting the Network	133
6.5.3 Example	134
6.6 EVALUATING THE METHODOLOGY.....	136
6.7 SUMMARY.....	137
CHAPTER 7: REALISATION OF THE METHOD: UML-JMT TOOL.....	139
7.1 USDX PARSER	139
7.1.1 USDX Class Diagram	140
7.1.2 USDX Model Extraction Methods.....	141
7.2 JMT SUITE	147
7.2.1 Queuing Network Solution Tools	148
7.2.2 Queuing Network Analysis Tools	148
7.2.3 Queuing Network Representation	149
7.3 UML-JMT TOOL DESIGN	150
7.3.1 UML-JMT: Components.....	151
7.3.2 UML-JMT: Class Diagram.....	151
7.3.3 UML-JMT: Activity Diagram.....	153
7.3.4 UML-JMT and JMT: the Integration.....	155
7.4 UML-JMT TOOL IMPLEMENTATION	156
7.4.1 Implementation of the Interface.....	156
7.4.2 Implementation of the Network Generation Engine.....	157

7.5 SUMMARY.....	161
CHAPTER 8: QUANTITATIVE EVALUATION	163
8.1 DEMONSTRATING UML-JMT	163
8.1.1 PDC for the IRS	164
8.1.2 Using UML-JMT to study the IRS performance	170
8.1.3 IRS Performance Results	177
8.1.4 UML-JMT as an Experimentation Tool.....	178
8.2 VALIDATING THE RESULTS' DEGREE OF ACCURACY.....	179
8.2.1 Case Study: Payment Switch.....	180
8.2.2 Payment System Architecture and Scenarios.....	181
8.2.3 Payment System Performance Study.....	188
8.2.4 Payment System Performance Results.....	190
8.3 SUMMARY.....	193
CHAPTER 9: QUALITATIVE VALIDATION.....	195
9.1 THE STUDY	196
9.1.1 Objectives	196
9.1.2 Method	197
9.1.3 Experimental Design	199
9.2 GENERAL METHODOLOGY EFFECTIVENESS ANALYSIS.....	201
9.2.1 Pre-orientation Interview Analysis.....	202
9.2.2 Post-orientation Interview Analysis	206
9.3 USABILITY OF THE UML-JMT TOOL	211
9.3.1 Usability Metrics	211
9.3.2 Results.....	212
9.3.3 Analysis	214
9.4 CONCLUSION	216
9.4.1 Study Outcomes.....	216
9.4.2 Suggestions.....	218
9.5 SUMMARY.....	218
CHAPTER 10: CONCLUSION	221
10.1 CONTRIBUTIONS AND ACHIEVEMENTS.....	221
10.2 OPEN PROBLEMS AND FUTURE WORK	223
10.2.1 Model Transformation Methodologies.....	224
10.2.2 CPASA Framework	224
10.2.3 Improving the UML-JMT Tool	225
10.3 RELEVANT PUBLICATIONS	226
APPENDIX A: USDX PARSER DOCUMENTATION	227

APPENDIX B: QUALITATIVE STUDY QUESTIONS.....231

APPENDIX C: QUALITATIVE STUDY RESULTS.....236

MODEL TRANSFORMATION METHODOLOGIES REVIEW240

REFERENCES.....248



Acronyms

AD	Activity Diagram
CASE	Computer Aided Software Engineering
CD	Class Diagram
CoD	Collaboration Diagram
CPASA	Continuous Performance Assessment Of Software Architecture
DD	Deployment Diagram
DOM	Document Object Model
DTD	Document Type Definition
EG	Execution Graph
EQN	Extended Queuing Networks
GSPN	Generalised Stochastic Petri Nets
JMT	Java Modelling Tool
LQN	Layered Queuing Networks
MOF	Meta Object Facility
MVA	Mean Value Analysis
NFRs	Non-Functional Requirements
OMG	Object Management Group
PASA	Performance Assessment Of Software Architecture
PCD	Performance Data Card
PEPA	Performance Evaluation Process Algebra
PFQN	Product Form Queuing Networks
QN	Queuing Networks
QoS	Quality Of Service
SA	Software Architecture
SAX	Simple API For XML
SC	State-Chart Diagram
SD	Sequence Diagram
SPA	Stochastic Process Algebra
SPE	Software Performance Engineering
SPN	Stochastic Petri Nets
TDD	Test Driven Development
UC	Use-Case Diagram
UML	Unified Modelling Language
USDX	Use-Case, Sequence And Deployment Diagrams XMI Parser
XMI	XML Metadata Interchange
XML	Extensible Mark-Up Language

Introduction

Software systems are built according to users' defined specifications, known as system requirements. These requirements describe how the systems are supposed to work. In software engineering, there are two types of system requirements; *functional* and *non-functional*. Functional requirements are the ones defining how the system will react in different scenarios. Non-functional requirements represent the quantitative and qualitative specifications of a system i.e. constraints on time and other resources[4]. Requirement engineering is an essential branch of system engineering and different system engineering schools of thought have diverse views on it. One of the well-known requirement engineering processes consists of elicitation, analysis, specification, validation/verification and management[4]. Other taxonomies exist for requirement engineering process and all of these processes share validation and verification as key tasks. Requirement validation means checking that the given requirement can be implemented (i.e. being realistic and conflict free), while requirements verification refers to the assurance that the implemented system reflects the specified requirements. Requirements verification is a process that continues through the life cycle of the software system.

Since the discovery of the software crisis in 1960, much attention has been given to the verification of functional requirements as it was identified as being particularly significant [5]. Most of the methods that were used in verification, such as prototyping, were focused only on functional requirements. Even the modelling languages that were used to model these requirements focused only on modelling functional specification. Over the past decade, researchers have addressed the substance of integrating non-functional requirement in the development process. One of the principal non-functional requirements for software is performance. Performance requirement verification is known as *Software Performance Evaluation*.

1.1 Software Performance Evaluation: What?

Software performance evaluation is defined as the process of analysing and optimising a system under study, in order to ensure this system satisfies the performance

requirements specified in the performance non-functional requirements specifications [6]. Performance evaluation involves the description of the system through the process of modelling; using this model, the system under study can be analysed according to observations gathered from the dynamic (time dependent) behaviour of the system, and the data flow between the components composing the system. Using data gathered from this analysis (i.e. *throughput*, *resource utilisation* and *bottlenecks* in the system), we can optimise the design of the system to be effective from a performance point of view. In the process of creating the models used in performance analysis, an abstract view of the system under study is first selected. This view is chosen to cover performance critical scenarios; these scenarios will be parameterised to define the points that affect the performance of the system. This abstract view will be used to construct the *performance model* of the system. The goal of these methodologies is to provide performance related design guidance during the system development.

There are a variety of performance analysis techniques that can be classified according to how they are described or solved. The leading solution techniques for performance models are *analytic*, *numerical* and *simulation*. Simulation is the most general and flexible means of performance modelling. It has many uses, but its results are usually only approximations and the cost of increased accuracy is longer execution times of the simulation performance model. Analytical techniques provide models which can be solved symbolically for the average (steady state) behaviour of a system. Unfortunately, only a very limited set of models have such solutions. Even fewer have exact solutions. Numerical techniques involve deriving an underlying model, typically a continuous time Markov chain, which can be solved for a given set of parameters by solving a set of simultaneous equations. These are somewhere between analytical and simulation models, being more general but slower than analytic techniques and less general but faster than simulation [7]. We will discuss the performance evaluation methodologies in more detail in Chapter 3 of this thesis.

These performance evaluation techniques are known as stochastic performance evaluation methods. The term “stochastic” refers to a stochastic variable which is used to emulate the effect of the external environment on the non-deterministic behaviour of the modelled software system. These performance predicting techniques are also known as non-deterministic performance evaluation techniques due to the behavioural nature of

the system they evaluate. Throughout the rest of the thesis, we will use these two terms to describe performance evaluation techniques.

1.2 Software Performance Evaluation: Why/Why not?

As the size and complexity of modern software systems increases, the need for methods to assist in design decisions and the assurance of the design quality is becoming more significant. Currently most of the software engineering processes require continuous verification of the implementation and design of the system against the functional and non-functional requirements. As we said earlier, research has only concentrated on the methodologies and tools for the verification of functional requirements. As Smith et al. explained[8], the earlier the performance verification process is undertaken, the more certain we are of finding any design faults that may affect the quality of the final software product. Despite its importance in the software design process, it is widely acknowledged that the lack of performance requirement verification is mostly due to the knowledge gap between software engineers/architects and performance engineering experts. In addition, most of the well known performance evaluation processes require an extra budget required to fulfil the performance evaluation task. This budget will be invested in hiring professional system modellers or in programming simulation models for the system. This overhead in financial and time resources can cause the exclusion of this task from the software project plans.

1.3 Software Performance Evaluation: Finding the Solution

The lack of utilisation of non-deterministic performance evaluation techniques has inspired researchers to find comprehensible, cost efficient techniques that will allow system architects to complete the performance analysis task without any of the additional costs listed in the previous section. One approach, which has been investigated widely, is to use the system architectural and behavioural characteristics represented in software modelling languages (e.g. UML) as the source to generate an equivalent performance model for the system under study. These methodologies utilise the structural and behavioural aspects of the system represented in different notations of a UML model, in addition to expected workload characterisation of the projected system, to generate a performance evaluation model that can be solved or simulated to assess the expected QoS specifications of a suggested design. Literature reports a number of methodologies for transforming UML diagrams to different types of performance models. Although these methodologies can help in capturing the performance aspects of the designs that they represent, the simplicity of these methodologies and the degree of automation of the

performance evaluation test provided by them, will affect the ability to merge these methodologies in the non-functional requirements verification task in any of the software development processes. We will discuss these methodologies and their evaluation in detail, in Chapter 4 of this thesis.

The main problem with similar methodologies is that they cannot be generalised to all system types; this is due to the fact that different systems, with different architectures, require specific performance evaluation techniques, and different performance measures require different modelling paradigms. This leaves us in a stalemate in the process of bridging the knowledge gap between software performance evaluation and software engineering. The solution is a straightforward method that will assist the software engineer in conducting software evaluation without any extended knowledge of performance analysis terminology. This can be achieved by designing a model transformation methodology that will assist the user of this methodology from the beginning of performance parameter capturing through to the analysis of the performance characteristics gained from solving the resulting performance model. This methodology will be designed to produce a performance model general enough to be capable of representing the architectural and behavioural aspects of a wide-rang of software information systems.

The main objective of this thesis is to design this methodology and to use it to create a tool capable of assisting software engineers in software performance evaluation tasks. The objective behind designing such a methodology and the associated tool is to black box the performance evaluation process so that the user of the methodology will only be concerned with representing the projected system architecture and behaviour and gathering the required performance characteristics and workload, and setting the objective of the performance study. This methodology will be designed with a view to it being deployable in major software engineering paradigms. As the methodology deploys non-deterministic performance evaluation techniques, the methodology will be designed to fit into the design validation phase of the software development cycle. We will discuss the deployment of software performance validation in the software development methodologies in Chapter 4 of this thesis.

1.4 Contributions and Major Results Achieved

The need for a methodology that will assist the user in choosing the performance study, capture the required performance variable and simplify the build and analysis of the

performance model, inspired the author of this thesis to come up with the UML-EQN methodology discussed in Chapter 6. This methodology adopts the SPE (Software Performance Engineering) framework[6] in dividing the architectural model of the system under study into two meta-models; software and machine models. This will give the designer the benefit of testing different alternatives of structural and behavioural configurations. This performance study would help the designer in deciding an initial design for the projected system. The UML-EQN methodology (published in [9]) takes advantage of the use-case, sequence diagrams to build the software model and deployment diagrams to structure the machine model. The resulting performance model is an EQN (Extended Queuing Network) performance model.

The methodology introduces an assisted method for gathering the performance related variable essential for the performance evaluation process. This method is called the Performance Data Card (PCD) [9]. PCD is a data sheet used for supporting the capture of the performance variables used in the build and analysis of the performance model. With the help of automatic design models to performance model algorithms, introduced in the UML-EQN methodology, a software engineer with basic understanding of performance modelling paradigm can conduct a performance study on a software system design.

The UML-EQN methodology was implemented as a tool which builds on one of the queuing network solving and analysis tools named the JMT suite[10]. The tool which realises the UML-EQN methodology is called the UML-JMT tool (published in [11]). The UML-JMT tool works as a wizard that assists the user in identifying and collecting the required architectural and performance specifications of the system under study, then aggregates these inputs and builds an output performance model, formatted to be solved using the analysis tools provided in the JMT suite. The JMT analysis tools provide the user with abilities to conduct different types of performance studies that will assess a projected system architectural design task.

As performance verification is a part of any software engineering methodology, we have to define frame works that would deploy performance requirements validation in the context of software engineering. Agile development paradigm was the result of changes in the overall environment of the IT and business worlds. These techniques are based on iterative development, where requirements, designs and developed

programmes evolve continually. This paradigm depends on continuous automated testing for the purpose of verifying the implementation of the current release against the current set of requirements. At present, the majority of literature discussing the role of requirements engineering in agile development processes seems to indicate that non-functional requirements verification is an uncharted territory. CPASA was designed to work in software projects where the performance can be affected by changes in the requirements and matches the main practices of agile modelling and development. The author of this thesis has suggested the CPASA -Continuous Performance Assessment of Software Architecture-(published in[12]) framework for the assessment of a system performance during the development of this system, using incremental and agile development paradigms. The UML-JMT was designed to implement the performance evaluation tests specified in this framework. Continuous assessment of software performance requires a comprehensible tool that provides the user with performance characteristics of a design. The UML-JMT is designed to be used as an automatic testing tool for the verification of performance non-functional requirements. This functionality is essential in incremental and agile software engineering processes. In software developed using these development processes, continuous verification of the requirements is a fundamental operation. This comes back to the fact that these software development paradigms allow continuous change in requirements. These changes may have effects on the overall performance of the system.

The UML-EQN methodology and tool were validated both quantitatively and qualitatively. The quantitative validation was derived by comparing the results gained from the UML-JMT tool to the results provided by another performance evaluation paradigm. The qualitative validation aimed to study the attitude of software engineers with different backgrounds, levels of experience and from different sectors of the software development industry, toward the tool and the methodology. The main objective of conducting this qualitative study is to investigate the efficiency of the general methodology, and the usability of the UML-JMT tool. The efficiency of the methodology will be investigated by identifying the challenges faced when deploying the performance evaluation in real software system projects in the industry. These challenges are represented in the knowledge gap between software performance and system engineering, and the availability of tools which assist software engineers in automating the build and analysis of the required performance models. The study also investigates any other factors that may lead to disregarding the performance evaluation

at the system design stage, such as the system size and the ability to interpret the resulting performance data gained from the performance study. The study will investigate the usability of the UML-JMT tool from the perspective of learnability, effectiveness and user satisfaction. The work described in this thesis has appeared in some publications which are explicitly detailed in 10.3.

1.5 Thesis Outline

This Thesis is composed of ten chapters. The first three chapters are classified as background chapters, aiming to set the context of this thesis. In Chapter 2, we will discuss relevant terminology related to the software engineering domain. This is important as the main objective of the thesis is to provide a performance evaluation methodology in the context of software engineering terminology. In that chapter, we will define software systems engineering, discuss some of the software engineering schools of thought and the different development paradigms available. Then we will discuss requirement engineering and validation (as a task of software engineering) and explain the importance of validation of performance non-functional requirements. As this thesis discusses the UML model transformation approach; we will need to define UML modelling as it is the standard modelling paradigm used for representing the behavioural and architectural aspects of a software system. This chapter will also provide background about CASE and modelling tools as well as background about XMI model representation.

In Chapter 3, we will provide background information related to software performance evaluation technologies. In that chapter, we will define the software performance evaluation process and its importance. We will then explain the process of software performance evaluation, describe the fundamental terminologies used in the process, and detail the main techniques used to perform this process. In that chapter, we will also discuss the use of these modelling terminologies in software systems performance modelling, in the context of the “best” paradigm that can model these systems. We will also clarify why EQN was chosen as the output model in the UML-EQN methodology, and justify the reason for choosing the JMT suite for performance model analysis in the UML-EQN tool.

Chapter 4 will discuss integrating performance evaluation in software engineering paradigms. These are represented in the software performance engineering frameworks. We will define the role of performance evaluation in software development and describe

how to integrate performance evaluation into the software engineering process for the two main software engineering paradigms (conventional development and agile development). For the conventional (i.e. waterfall development paradigm), we will describe the PASA framework [13], whereas for the agile development paradigm, we will introduce the CPASA framework. In this chapter, we will also provide a literature review of performance model building methodologies based on the model transformation technique. We will evaluate these based on a set of criteria that we will define and justify.

Chapter 5 explains a methodology that was the result of the author's first work in the field of performance evaluation automation techniques. When developing this methodology, the work involved automating the extraction of a generic performance model. The methodology extended in this chapter is based on the *state marking* methodology originally developed by King and Pooley[14]. The state marking methodology concentrates on capturing the behavioural aspects of the modelled system in a behaviour oriented performance model. The original state marking methodology proposed a method for extracting GSPN performance models from a meta-model composed of collaboration and state-chart models. The limited generality of the GSPN and the non-standard input model used, motivated the author to extend the state marking methodology. The extended methodology proposes a systematic approach for extracting Markov chain models from *performance annotated sequence* UML models[1].

Chapters 6 and 7 represent the main contributions of this thesis. In Chapter 6, we introduce a methodology dedicated to assisting software engineers in conducting performance studies from the early stages of the systems life cycle. The UML-EQN methodology includes steps which begin with gathering performance parameters needed to build the performance model. The methodology provides systematic algorithms that are designed to facilitate the process of converting the design model to an EQN performance model. In Chapter 7, we will discuss the UML-JMT tool, the tool that implements the UML-EQN methodology and act as a UML interface for the queuing network solving tools in the Java Modelling Tools (JMT) suite. The UML-JMT Tools is a graphical user interface tool that will help users in building a performance model for their software system, in a wizard like approach. The user will supply the tool with the performance data card entries in a question and answer format. The tool will then use the UML-EQN conversion algorithms to construct a performance model based on the

user entries. This model can be solved and analysed in a simulation based queuing network solver provided by the JMT suite. This chapter represents a full technical specification for the development of the UML-JMT tool.

As we discussed earlier, the methodology and the tool described in Chapters 6 and 7 were validated quantitatively and qualitatively. The validation Chapters 8 and 9 will discuss the validation of the methodology and tool discussed in this thesis from both the qualitative and the quantitative points of view. In the qualitative validation in Chapter 9, we will investigate the attitude of a sample from software engineers toward the methodology and the tool. In Chapter 8, we will investigate the methodology and the tool from the context of the results provided by the UML-JMT tool. What we are searching for is to investigate whether the performance indices provided by performance models built by the UML-JMT tool are valid to a degree of accuracy. The methodology we are deploying for the quantitative validation is by comparing the results gained from a performance model produced by the UML-JMT tool and analysed by the JMT suite, to the same performance indices provided by a similar tool. We will also compare the results provided by the UML-JMT tool to the results gained by a real benchmarking exercise. This chapter will also be used to demonstrate the use of the UML-JMT in two case studies. In the last chapter, we will conclude the thesis by summarising the contributions and the results gained during the study, and outlining the open areas of research and the future work in this research domain.

Software information systems can be defined as the software used for the representation, processing, and distribution of information. Software information systems are a class of software that have a number of homogenous users interacting with the system through an interface. These systems are usually run on a single or multiple servers[15]. These specifications distinguish information systems from other classes of software such as real-time systems, scientific systems or expert systems. In this thesis, we are only concerned with information systems; therefore, in the following chapters “software system” will correspond to information systems software. Software engineering can be defined as the systematic process of development and maintenance of software systems. This chapter will cover software engineering concepts and terminologies. In addition, it will provide the reader with technical knowledge regarding software engineering and software modelling which will be needed in the following chapters. At the beginning of Section 2.1, we will discuss some of the key software engineering paradigms. In Section 2.2, we will discuss one of the important branches of software engineering, requirement engineering. In this section, we will also discuss the gathering and verification process of requirements, specifically performance requirements. In Section 2.3, we will discuss the UML modelling notations[16]. UML is a standard modelling notation used to represent data, process, scenarios and architectural aspects of the system. This notation is becoming a common language in most of the software engineering methodologies and tools. These notations are used as the input for the performance model generation methodologies discussed in this thesis. Section 2.4 will discuss the tools used in the deployment of software engineering methodologies. Finally, Section 2.5 will cover the technical aspects of the standard representation of UML models’ notations in CASE tools.

2.1 Software Engineering Paradigms

The IEEE standard defines software engineering as “the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software” [17]. The software engineering field was introduced after the introduction of the high-level oriented

programming languages problem. The so-called *software crisis* arose as the size of software systems and resources expanded. The need for a methodology for managing the production, development and maintenance of software systems, inspired the suggestion of a number of different methodologies. Each of these methodologies represents a system of techniques and principles which implement the rules defined by a specific *school of software engineering*. There are a number of formal published methodologies, and a larger number of informal company defined methodologies. The formal methodologies are the ones defined with standard specifications. Defining the “*Best*” methodology for developing software systems is a controversial issue. Some opinions argue that a fixed methodology will limit the designer’s ability to generate professional, independent and creative designs. Other opinions attempt to define the best methodology depending on the nature and domain of the developed project.

Currently there are two main trends in software engineering that might be distinguished as old and new schools of software engineering. The old school (we will call it *conventional* software engineering) follows a discrete process, with a pre-defined set of deliverables after each phase. It requires a comprehensive understanding and specification of the problem domain and the system's requirements before the system implementation can begin. Most of the methodologies that are classified as conventional are either generalisations or related to the *waterfall* system development process. The main title of the new school of software engineering is "requirements are meant to change". *Agile* software development methodology is a trend that propagates from the industry of software development. The main goals of the agile development methodologies are:

- Increasing the business value of developed software systems, and
- Providing a realistic method of development in a world where customers change their requirements continually.

This is made possible by the development practices adopted by the agile development disciplines, such as just-in-time requirements, short-frequent releases, test-driven development ... etc. In the following subsections, we will discuss each of these trends by describing the techniques and practices for each of these development paradigms, in general.

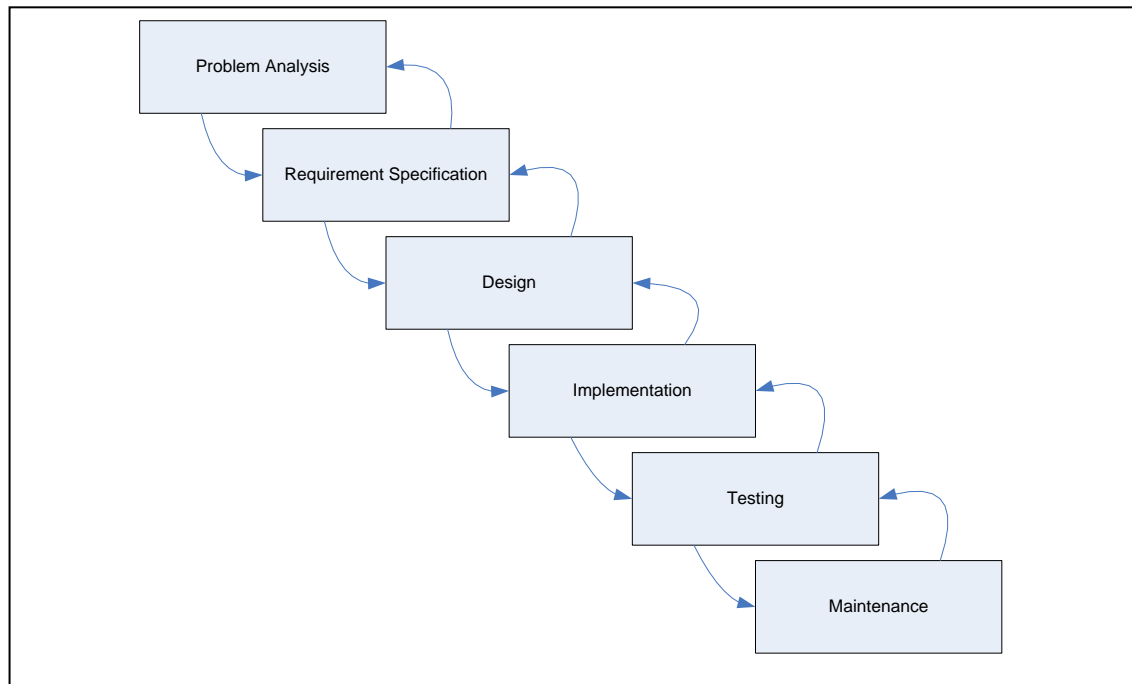


Figure 2.1: The Waterfall model phases

2.1.1 Conventional Software Engineering

The first software engineering paradigm was the waterfall model[18], which was inspired by older engineering disciplines, such as civil and mechanical engineering. The main motive behind the introduction of the waterfall model is to manage the complexity of the design of software systems as it defines the design as a process that goes through multiple phases. Each of these phases must be *signed-off* before progressing to the next phase. When a problem arises in any phase, the process will be *backtracked* to the previous phase(s) where it will be investigated and solved before repeating the sign-off procedure(s). W. Royce introduced the waterfall model in 1970. Since then, several improvements of the original model have been introduced (e.g. Spiral Model [19], Chaos Model [20], V-Model [21]). Although these improvements touched the sign-off and minimised the risk of backtracking, the phases were generally similar. The phases of the conventional waterfall model are shown in Figure 2.1. This diagram distinguishes the main phases of the waterfall model as follows:

- *Problem analysis:* In this phase, an explicit model of the environment where the system is going to be deployed is created.
- *Requirement specification:* In this phase, after the analysis of the organisation model, the requirements of the system needed for the organisation will be collected. This part involves the introduction of the broad solution of the problem in hand, and it is usually classified as one of the hardest and most

significant phases of the system development. The criticality of this phase caused the introduction of processes and techniques dedicated for this phase. We will discuss later in Section 2.2.

- *Design*: This phase involves designing the system that will realise the functional requirements defined in the previous phase. Usually the decomposition approach is used to define components that will compose the system.
- *Implementation*: In this phase, the different components defined in the design phase are coded in the chosen programming language.
- *Testing*: In this phase, the functional and non-functional requirements are tested in the system implemented in the previous phase.
- *Maintenance*: New requirements and functionality are usually added to the software system as the users start using these systems. In this phase, the developed system is added with new requirements. It is generally agreed that most of the effort is spent in this phase.

This methodology provides a structured and disciplined engineering approach in the development of software systems. In addition, it distinguishes the phases of software development, which can help in the specification of problematic phases. This is essential as the cost of solving software defects increases exponentially with time. The waterfall model adopts what is called *Big Design Up Front* approach. This means that there will be time emphasis in the beginning of the project on understanding the problem domain and the customers' requirements, and on producing consistent design. The documentation sign-off process between phases helps in the limitation of problems caused by users who do not know their exact requirements or developers not understanding the project domain.

On the down side, in practice it is rare to go straight from requirements, to design, to implementation, without backtracking. The validation of the requirement against the design starts just before the end of the implementation phase, and that will cause a costly backtrack if there are any unsatisfied requirements[22]. This comes back to the low *visibility* of the *end-product* in the phases prior to the implementation phase. This problem was tackled by most of the improvement models built upon the waterfall model. In Spiral software development, system visibility is improved via continuous prototyping process. The prototypes produced in each of the developments are the result of improvement of an initial prototype of the suggested design, and of the prototype of

the previous stage. The V-model provides a means of reducing backtrack risks by merging the requirement validation and verification processes as early as possible. The *Unified Process* (UP) defined one of the important improvements to the original waterfall models. It improved visibility of project outputs by providing deliverables to the customer at the end of pre-defined fixed time iterations.

The Unified Process (UP)

One of the modifications of the waterfall model that was aiming to avert backtracking is the Unified Process (UP). The UP represents a process framework that provides an infrastructure for developing software projects. The UP provides essential guidelines for development such as software development framework, lifecycle model, collaboration, and interaction. Therefore, The UP needs to be customised for specific organisations or projects[23]. One of the widely used customisations is the Rationale Unified Process (RUP) developed by IBM[24]. The development life of a project in UP consists of a series of *cycles*. Each cycle concludes with a product *release*. This release increases the visualisation of the project. These releases can be blue prints for the project, documentations or developments of working functionalities. Each cycle consists of four phases:

- *Inception* - where the plans of the work are committed.
- *Elaboration* - where the basic architecture of the work is decided, construction plans are laid and risk assessment is done.
- *Construction* - phase in which a beta-release of the system is provided.
- *Transition* - where the customer is introduced to the system.

These phases are further divided into fixed time *iterations*. Each cycle is developed through the core waterfall workflow model (requirements, analysis, design, implementation and test).

The main features of the UP models can be summarised in the following points:

- *Model based*: The UP model uses UML for all of the systems' blue prints as well as an assessment problem solving. It provided models for capturing the requirement and visualising the solution of the problems (requirements). Figure 2.2 illustrates the model-based phases that the UP models have and the types of models produced in each phase. We will discuss the UML modelling notations further in Section 2.3.

- *Use-case driven*: A use-case specifies a series of activities and behaviours which the system can perform, and that will provide functionalities and a result of value to a particular actor[23]. Use-cases are used to define the functional requirement. In the UP development; each iteration involves implementing by defining the scenarios and alternatives of this specific use-case, then it will take this use-case through requirements all the way to implementation, test and deployment.
- *Architecture centric*: this means that an architectural view of the system is essential through the development.
- *Iterative and incremental*: All of the phases of the UP are divided into a series of iterations. Each of the iterations ends with an increment to the system; this is presented as a release of the system that contains added or updated functionalities.

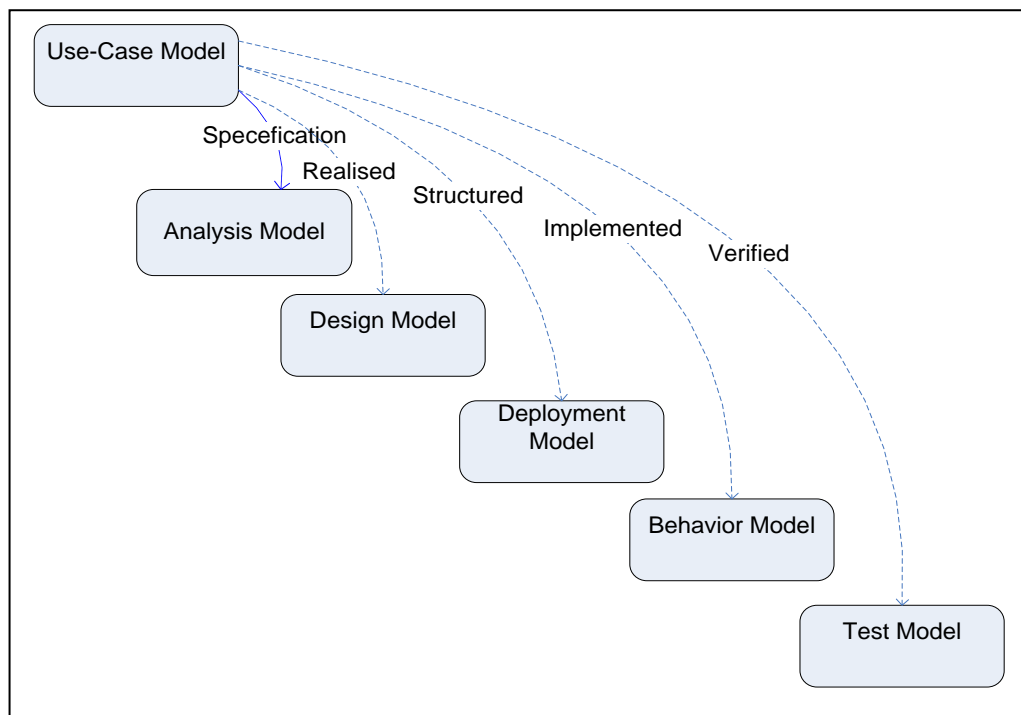


Figure 2.2: Model produced after each phase of the UP development

The main disadvantage of the UP resides in the heavy weight of the process. This is caused by the training, documentation and tools required to deploy the process. In addition, visibility of the project is not always gained; this return to the modelling artifices used in UP (UML) may not be understood by the customer. As in the waterfall model, requirements change in later stages of development is still exceedingly difficult.

2.1.2 Agile Development Methodology

The agile development process evolved in the software development industry. It was introduced after numerous challenges caused by the stakeholders' requirements, which tend to change in the majority of software projects. The agile development process concentrates on the business value of the software system by allowing customers to concentrate only on the requirements that they will utilise. Also, in the computer world, the value of the technology decreases rapidly with time; delivering the software on an incremental basis will increase the value of a software system. Another advantage of incremental releases of software systems is the increased visibility, which will allow customers to point out any unanticipated features early. This will reduce the project's risk. Examples of software development methods which deploy the agile development methodology include DSDM[25] (Dynamic Systems Development Method), SCRUM [26] and XP[27] (eXtreme Programming).

There are a set of principles which distinguish agile development from conventional development methodology, which are as follows:

- Deliverables are full working functionalities.
- The Software project is delivered in incremental releases.
- Requirements are allowed to change until their functionalities are delivered.
- Development teams have to be complete and empowered to make decisions.

Agile development defined a set of techniques and practices that will allow the deployment of the above principles, some of which are:

- Just in time requirements: Requirements of functionality are only specified at the iteration in which this functionality's requirements are developed.
- Test Driven Development (TDD): Any functionality developed will be tested using a predefined test provided by the customer. These tests represent the system specification.
- Pair programming: This involves pairing the programmers in the development on the same development station, where one of them is coding; the other will be considering the development strategy.
- Stakeholder involvement: A representative of the stockholders has to be at the development site to assist in decision making.
- System consistency kept by refactoring the design.

Figure 2.3 [28] illustrates a model outlining the agile software development process. As with any software development process, it consists of the requirements, design, implementation and testing tasks. It differs from the conventional phase oriented development process in the continuous deployments of these tasks on small-scale projects, each of which represents a functional component(s) of the overall system. The agile development process is a component oriented development process, in which the functionalities of the system are decomposed to simpler, easy to develop components. Each of these components is developed in a time scaled phase called *iteration*. At the beginning of each iteration, the requirements for this particular component are identified, and at the end of the iteration, a fully tested, approved segment of the projected system is released.

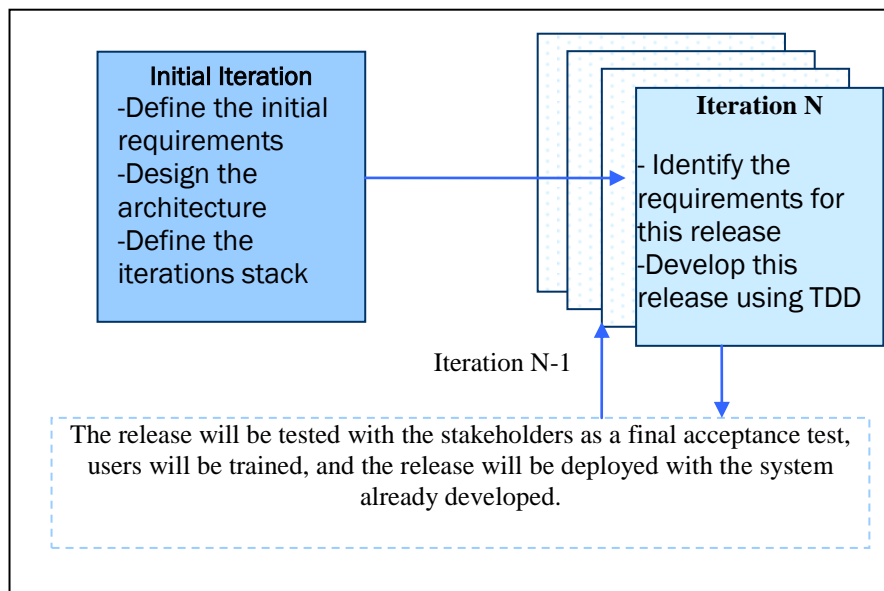


Figure 2.3: Agile development process

Each agile project starts with an initial iteration. In this iteration, an initial requirements and architecture *envisioning* process takes place [28]. At this stage, software engineers only identify the basic functionalities needed in the system. These functionalities can be modelled as abstract use-cases. Based on these requirements, an initial architectural representation of the system can be suggested. This architectural definition of the system clarifies the components composing the system. This will be used to construct the project iterations plan. In most of the agile developments methodologies, the iterations are arranged in a priority stack, sorted according to the value of functionality provided by the iteration and the number of details available to the stakeholder. At the end of the initial iteration, the software engineer is supposed to have a broad understanding of the project domain, initial functionalities list (use-case diagram) and a project iterations plan.

In the implementation iterations, each iteration ends with some functionalities of the projected system, tested and released to be used by the system's users. At the beginning of each iteration, the detailed requirements of the functionalities developed in this iteration are identified by the stakeholders. These requirements are usually presented as *user stories*. The user stories are textual representations of the use-cases of the system. The user stories do not cover all possible exceptions and pathways that are explained in the use-cases. For that reason, agile development rules insist that the customer (someone who understands the business case) is always available with the design team in order to clarify the business purpose, to help with conducting tests and to make small-scale decisions. The implementation of these functionalities is done by close-knit design/coding teams working together to implement the functionalities defined by the user stories[28].

The verification of the developed segment of software against the functional requirement is carried out at the end of each iteration. This is done using tests that are written before the implementation starts. These tests are usually provided by the customer, and any code written will be validated through the tests. These tests are seen as the system specification. The consistency of the design is gained through *refactoring*. The designer in an agile methodology will always start with a simple user story design and then build on the design of this story.

One of the main disadvantages of the agile methodology in the refactoring strategy is that it only covers the system itself and not the published interfaces which are essential in any system. It is not practical to refactor these APIs and some other codes, and therefore, thinking for today and forgetting about tomorrow is not always adequate. Some of the practices required by the agile methodology cannot be practiced either because of human resistance (programmers refuse to work in pairs), or for physical reasons (the client does not live in the same geographical area as the programmers).

2.1.3 Conventional Vs. Agile Development Methodologies

Conventional and agile development methodologies provide strong and structured approaches for the development of software projects. Each of the methodologies has its strengths and weaknesses. From a project manager's point of view, choosing between these methodologies will depend primarily on the nature of the project and the development team. Selecting a development methodology that developers are familiar with, or are willing to consider is essential. Literature reports methods for classification

of the best development methodology depending on the projects' characteristics. Some of these techniques come in the form of a decision tree [29], where the answers for a set of questions about the nature of the project, will offer a proposal for the best development methodology. Boehm and Turner[30] suggested a process for selecting the best development methodology with respect to risk assessment. They set up five variables characterising the project nature, these are:

- *Dynamism*: This variable represents the degree of change expected in the requirement, along the development period. As this variable increases, the project leans more toward agile development.
- *Size of the development team*: This variable represents the number of developers involved in the project. Agile development works best with small-scale development teams; therefore, this variable increases as the project moves towards the conventional development.
- *Criticality of the project*: This variable measures the degree of criticality of the project, based on a scale starting from “many lives” to “comfort”. As the criticality of the project increases, the project manager is advised to adopt a conventional development methodology.
- *Personnel*: The variable concerned with the experience of the development team. As the development team is more experienced, agile methodology can be used with reduced risk.
- *Culture*: The culture of the development team is a decisive factor when selecting a methodology. This variable concentrates on the discussion making culture of the development team. As management and discussions move toward central management, the best development methodology is conventional.

2.2 Requirements Engineering

Software requirements can be defined as the functions, constraints and actual goals of the software systems[4]. Requirements engineering is the branch of software engineering which manages and controls the requirements, and requirement related activities. As requirement engineering is a part of the software engineering process, different software engineering schools have diverse views of the requirement engineering methodology. However, the main and most important activities in any requirement engineering methodology are requirement gathering, validation and verification. Requirement gathering includes elicitation and analysis of the user specified requirements. Requirement validation is concerned with checks made on the

requirements to verify if they can be implemented (i.e. being realistic or not conflicting with each other). Requirements verification involves the assurance that the implemented system reflects the specified requirements. Requirement verification is a process that continues through the life cycle of the software system. Requirements are generally classified into two categories depending on the nature of these requirements. These categories are *functional* and *non-functional* requirements. Functional requirements define the functions, behaviours, inputs and outputs of a software system. Non-functional requirements determine the qualitative and quantitative aspects of the system (i.e. performance, security, availability, usability etc).

In this section, we will discuss the gathering and verification processes for both types of requirement. We will concentrate on non-functional requirements, as this thesis covers performance requirement verification. Subsection 2.2.1 will cover requirements gathering and representation. Section 2.2.2 will discuss the process of requirements verification. We will discuss these two aspects from the point of view of the software development methodologies discussed in the previous section.

2.2.1 Gathering and Representing Requirements

Requirements gathering process is concerned with clarifying the requirements of the projected system from the system's users, and turning the view of these requirements from vague to specific by representing these requirements in a form understandable by the developers and customers. There are many techniques used to gather requirements from the user which include:

- *Interviewing/ questionnaires*: The main stakeholders using the system are interviewed/questioned (depending on the number of users) and asked about their expectations of the system functionalities, and the expected qualitative and quantitative specifications.
- *Observation*: Most software systems are developed to replace another system; this system might be old software system or a manual system. One method used in understanding the requirement of a system involves observing the process of the existing system, and analysing the documents that describe the processes of this system.

In conventional development methodology, detailed requirements are gathered in a specification before the implementation starts. On the other hand, agile development

requires full specification for any functionality at the time of development of this functionality. Therefore, requirement gathering and representation can be seen as a phase in conventional development and as a *continuous* activity in agile development.

Representation of functional requirements is a widely researched area, and there are abundant examples of standards used to represent the functional requirements. One of the standards used to represent functional requirements is UML (Unified Modelling Language) which we will discuss in more detail in the next section. As we said in the definition of functional requirement, these requirements include functional specifications, data structures and behaviours. Modelling languages provide diagrams that are used to represent and abstract each of the aspects defined by the functional requirements. These models are used to increase the visibility of the project for both the customer and the developer. Another way of representing functional requirements is through prototyping, where a sample of the system is provided for the user in order to determine whether it reaches their expectations.

Non-functional requirements can be seen not as requirements, but rather constraints on the functional requirements defined in the specification documents. However, for them to be managed and tested in accordance with the requirement engineering methodologies, they are considered to be requirements. When defining non-functional requirements there is a main principle that should be satisfied, which is “it should be testable”. A requirement that could not be tested should not be classified as a requirement. IEEE-Std 830 – 1993 [31] listed 13 non-functional requirements to be included in any software requirements document. Examples of these requirements are performance, acceptance, security, reliability... etc. The non-functional requirement we are concerned with in this thesis is the performance requirement. We need to know what the performance requirements are for a software system, and how are they expressed in the requirement specification sheet.

Performance requirements are one of the fundamental non-functional requirements that need to be documented and tested from the early stages of the system development life cycle. If a system does not satisfy stakeholders’ performance expectations, it is deemed to be “non-functional”. There are three principal classes of performance requirements that need to be captured and documented for any software system[32], these are:

- *Response Time*: It will describe how fast the system handles individual requests. The response time requirement should define the maximum satisfactory time that the user should experience when performing a task. It is measured by calculating the time from when the user is given the “Go” command, until he/she has received enough feedback to proceed toward the next task. A response time should be supplied for each class of job and user.
- *Throughput*: This requirement will describe the maximum number of requests that the system should handle.
- *Concurrency*: Accounting for how many threads of work should be serviced simultaneously.

Most of the original, well-known, standard modelling languages do not tend to have specific models for presenting non-functional requirements (i.e. UML use-case diagrams). Non-functional requirements were presented as notes in other modelling notations. This will minimise the chance of automating requirement engineering activities (i.e. verification) in CASE tools. In UML, the performance and time information were later introduced to the standard in the UML Profile for Schedulability, Performance and Time[1]. This profile is an extension of the UML standard to accommodate UML quantitative performance annotations. These annotations allow the association of performance related quality of service (QoS) characteristics with selected elements of a UML model [1]. The profile explains these extensions to the UML standard in the context of the standard itself. It defines stereotypes, tagged values and constraints that represent the performance requirements and resource allocation of the modelled system[33].

2.2.2 Validating/ Verification of Requirements

As we said earlier, validation is concerned with checks for errors, conflicts and ambiguities in the requirements before these requirements are committed to design and implementation. On the other hand, verification concentrates on checking that the design/ implementation reflects these requirements. Because requirement validation is a task associated with requirement gathering, the validation of requirements is seen with respect to software development methodology. Requirement validation includes checking the requirement specification documents for problems that may affect the design and implementation of the system. These problems include:

- *Clarity of requirements*: These will be checks for ease of understanding, as the requirements might be inadequately expressed, or parts of the requirement have been omitted accidentally.
- *Missing Requirements*: In some cases, specific requirements are missed and not declared in the requirement specification.
- *Conflicting requirements*: Some requirements might conflict with other requirements.
- *Unrealistic requirements*: In some cases, a requirement cannot be implemented with the available technology, or with the restrictions applied.

These problems are identified in the *requirement review meetings*; these meetings involve stakeholders and software and requirements engineers. During these meetings, the requirement specification documents are continually reviewed and checked for the above problems. Once a problem is found, the meeting committee will decide a solution for it. In agile development methodology, these meetings will take place during the development iterations in the form of iteration initial meetings and urgent meetings with onsite stakeholders for requirements clarification.

Requirements verification is a task associated with software testing. In conventional development methodologies, requirements verification is done after the implementation of the system's functionalities. For agile development, the verification process is carried out throughout the software development life span. There are multiple techniques used in the requirement verification process. Ways for verifying functional requirements include prototyping, manual writing and model verification[22]. Model verification is an essential process of requirement verification that will ensure that all the models representing the system are consistent with the requirement specifications. Some CASE tools provide automated model verification functionalities. Not all requirements can be verified, and some requirements are classified as hard to test and verify. Examples of such requirements are those which affect the system as a whole, such as performance and other non-functional requirements. For these requirements, particular tests are required in order to perform the verification process.

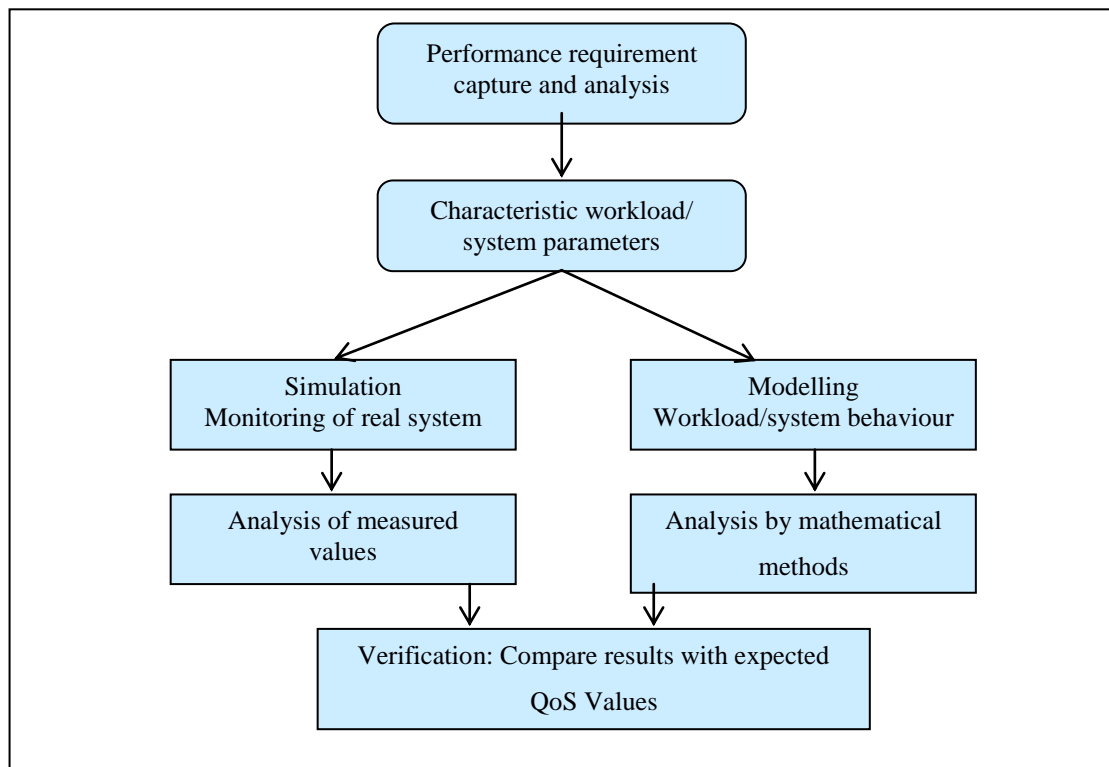


Figure 2.4: Performance requirements verification process

As was declared in the 2.2.1, currently there are limited representations and modelling notations for non-functional requirements, therefore, the automation of the verification process is still an open area. As we said earlier, the main goal of this thesis is to find a methodology that will allow the verification of performance non-functional requirement from the early stages of the system life cycle. This verification process tends to be automated by taking advantage of the architectural and behavioural models developed during the design of the software system. Figure 2.4 illustrates the process of performance requirements verification. At the first stage, the performance requirements mentioned in 2.2.1 are captured and analysed, and the required performance indices are specified. There are two ways of acquiring the performance characteristics of the suggested design of the system. These are *analytical modelling* and *simulation*; which will be discussed in the next chapter. The resulting performance indices from the performance study which represent the expected performance characterisation of the modelled system will be then analysed and validated against the initial performance requirements. The goal of this validation is to provide performance related design guidance during the system development. The performance requirements verification process will be further explained in Chapter 4 of this thesis.

2.3 UML System and Data Modelling

Since the introduction of the waterfall methodology, the alterations and enhancement methods have concentrated on increasing the visibility of the projected software system during each phase. As subsection 2.1.2 showed, one of the improvements provided by the UP method was the models produced after each phase which represent the result of that phase. There were several suggested modelling languages; some for modelling the process and others for modelling data. Data oriented modelling arrived first with the *relational oriented models*[34] and *entity relation models*[35]. The process modelling was followed by the arrival of flowcharts and structure chart diagrams (DFD)[36] and other models like Yourdon charts[37] and behaviour models[38]. Because data and structure must work together in software systems, integrated modelling languages were needed to provide a more accurate and comprehensive representation of the system. These models tried to represent both the static and dynamic aspects of the system. Examples of early versions of these languages are the JSD[15] and ACM/PCM[39].

The diversity of the modelling languages contradicted the main objective behind introducing them in the first place. This objective was to introduce a common language for all software engineers and the customers. This means that a consistent modelling language needs to be standardised for use in the software engineering community. UML (Unified Modelling Language)[40] was suggested by the OMG to provide a semi formal language for specifying, visualising and documenting software artefacts. UML provides graphical notations that will allow the user to describe multiple static and dynamic views of the system. Each model provided by UML provides a description of the system depending on the phase and functionality of this specific model (see Figure 2.2). UML provides a variety of modelling views of the system that comes in the form of a *Diagram*. These diagrams include:

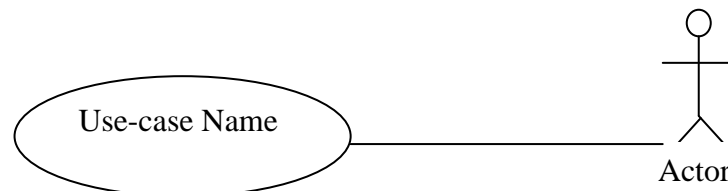
- *Use-case Diagrams*: These provide specification of the functional requirements as use-cases and users, participating systems as stakeholders and the association between them, to illustrate the relation between these entities.
- *Class Diagrams*: These provide a static representation of the classes composing the system and the association, multiplicity and inheritances relation between them.
- *Behaviour Diagrams*: These provide the dynamic interaction aspects of the system. There are two types of behaviour diagrams; *state-chart* and *activity* diagrams.

- *Interaction Diagrams*: These are used to describe the dynamic interaction of *objects* through the exchange of messages using service/function calls. There are two kinds of interaction diagrams; *sequence* and *collaboration* diagrams.
- *Deployment Diagrams*: These are used to model the configuration of the run-time processing elements of the software components.

In this section, we assume that the reader is familiar with UML; therefore we will only consider the relevant UML diagrams deployed in the methodologies explained in this thesis. These are use-case, sequence and deployment diagrams. We will describe these diagrams with a simple example. For further information about UML, the reader can refer to[40].

2.3.1 Use-case Diagram

Use-case diagrams describe the system's functional requirements relation with the *actors* using the system. The actors are the external users of the system. They can be human users or external systems communicating with the system being modelled. Use-case diagrams are used as a functional requirement specification document. Each use-case represents a function or a service provided by the system. Different scenarios might represent each of these use-cases. The graphical representation of a use-case is an oval with the name of the use-case inside it. The actors are represented graphically using a stick figure with the name of the actor beneath it, as shown below:



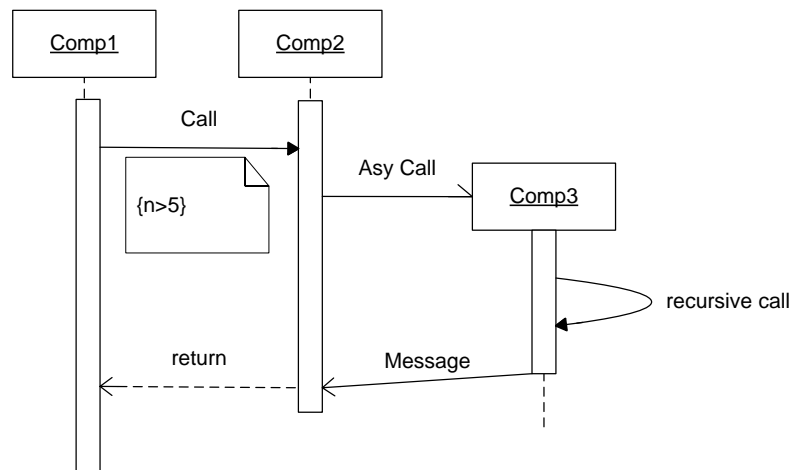
The specification of *who-is-using-what* in the system is represented by an association between the actor stick figure and the case. An actor can be associated with multiple use-cases and a use-case can also be connected with many actors. Other associations that define the relations between use-cases exist. These relations can be inclusion, extension and generalisation. We are not concerned with these associations in this thesis.

2.3.2 Sequence Diagram

Sequence diagrams are used to describe the internal behaviour of use-cases. This behaviour is specified by the interaction of the components (usually objects) involved in the implementation of this scenario. This interaction defines the scenarios representing the functionality of this use-case. The interaction is displayed as a set of ordered

messages and each of these messages is sent from one component to another. These messages represent calls for services provided by these components. Components taking part in the interactions are displayed horizontally, each in a box with the name of the component. Each box has a *lifeline* represented by a dotted line and a parallel solid line covering the time when the component is live in the interaction. Interaction messages (service calls) are represented by arrows originating from the caller component to the called component. Each of these messages has a name labelling the arrow which represents the message.

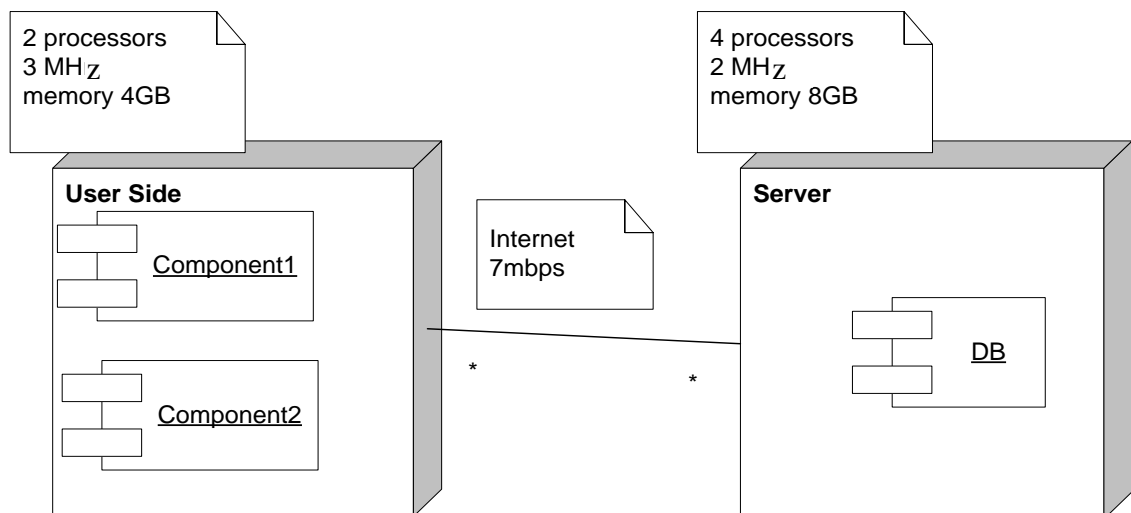
There are two types of messages; *synchronous* and *asynchronous* messages. The type of the message is denoted by the graphical representation of the arrowhead. Reply messages for synchronous messages are represented by an arrow with a dotted body. Messages originate from the position in the life of the calling component representing the time of the calling (i.e. the order of the calling), to the position on the lifeline where the function on the called component is invoked. A component may call a function on its own available functions list. The component can therefore be the sender and the receiver of a message, in what is called a *recursive call*.



Conditional calls can be represented in a sequence diagram by introducing a label on the message(s) controlled by this condition. This label will have the controlling condition written inside it. Iteration can be represented by including all the messages that are included in the loop, in a label that has the number of iterations, or the loop control condition. Concurrency can also be modelled in a sequence diagram by organising the messages to be called, one after the other, originating from the same calling component.

2.3.3 Deployment Diagram

A deployment diagram in UML illustrates the configuration of the runtime platform on which the software system runs. The system is shown as a set of nodes representing the different physical locations on which software components are located. These nodes have their own specification (processor speed, memory etc) and are interconnected by a communication media which has its own specification (i.e. transportation rate). Nodes are graphically represented in a deployment diagram by a box with the name of the node written in the top left hand corner. External components interacting with the system are represented as nodes with connections to the systems node (i.e. sensors, external database servers). Inside each node is a collection of *components* which reside in this physical node representation. Components residing in a node are drawn inside the box representing this node as rectangle with ports (as shown in the figure below) with the name of the component inside. The association connections between the nodes may be labelled with the specification of the type of connection.



2.3.4 Example: Video Search System

The example is for a video searching system that will allow the users to share and add video clips. This system will cache all clips previously stored, or of interest to the user (according to his/her profile) when the network usage is idle. Figure 2.5 shows the use-case diagram of the system.

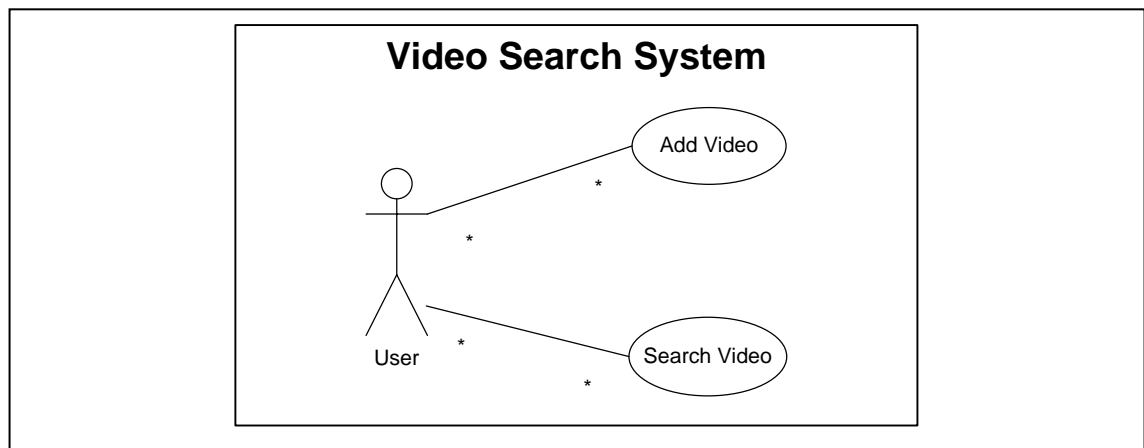


Figure 2.5: Use-case diagram for the video search system

As the above abstract description of the system explains, the system is used to add, search and view videos clips. The use-case diagram is surrounded by a system boundary with the name of the system on top. There is only a single actor named *user*. The two use-cases in this system are: *add video* and *search video*. There are two search operations; internal (in the local cache) and external (in a central database). Note that we have only one search use-case. This comes back to the requirement that the search operation is not transparent to the user. If the requirement insisted that the user chooses where to search, then we would have two use-cases defining the search operation. Figures 2.6 and 2.7 show the sequence diagrams defining the scenarios of the *add video* and *search* use-cases respectively. The system is constructed from three main components which are: *interface*, *internal DB* and the *VDB* (video database). In the *add-video* use-case, there is only one scenario accounted in this abstraction of the system, which is adding a video successfully.

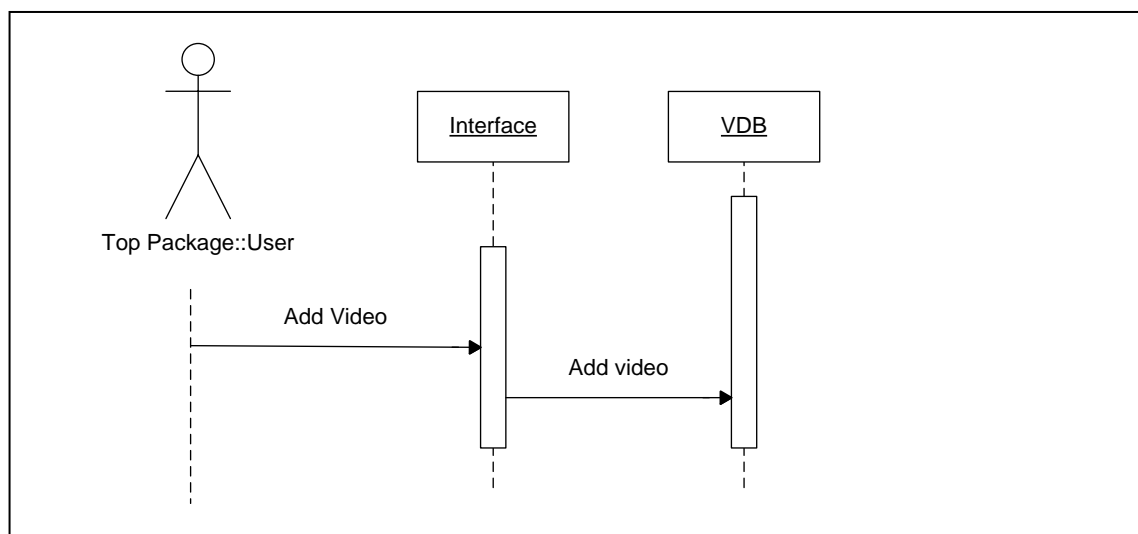


Figure 2.6: Sequence diagram of the “Add video” use-case scenario

This is done (as shown in 2.6) by the user requesting to add a video clip by calling the *add-video* function in the interface component. The interface component will process the video and send it to the VDB in an “*add video*” request. Figure 2.7 describes the scenarios for the search use-case. The two scenarios are for when the requested video is available in the system internal cache; then it will be played directly to the user. The second scenario describes when the requested video is not found locally; then the VDB will be searched, and references will be passed to the user. Figure 2.8 shows the deployment diagram of the video system. In this diagram, we have two nodes on which the components of the system will reside. At the user side, there will be the interface and the internal database, and at the video server side there will be the VDB. The two nodes will be connected through the internet.

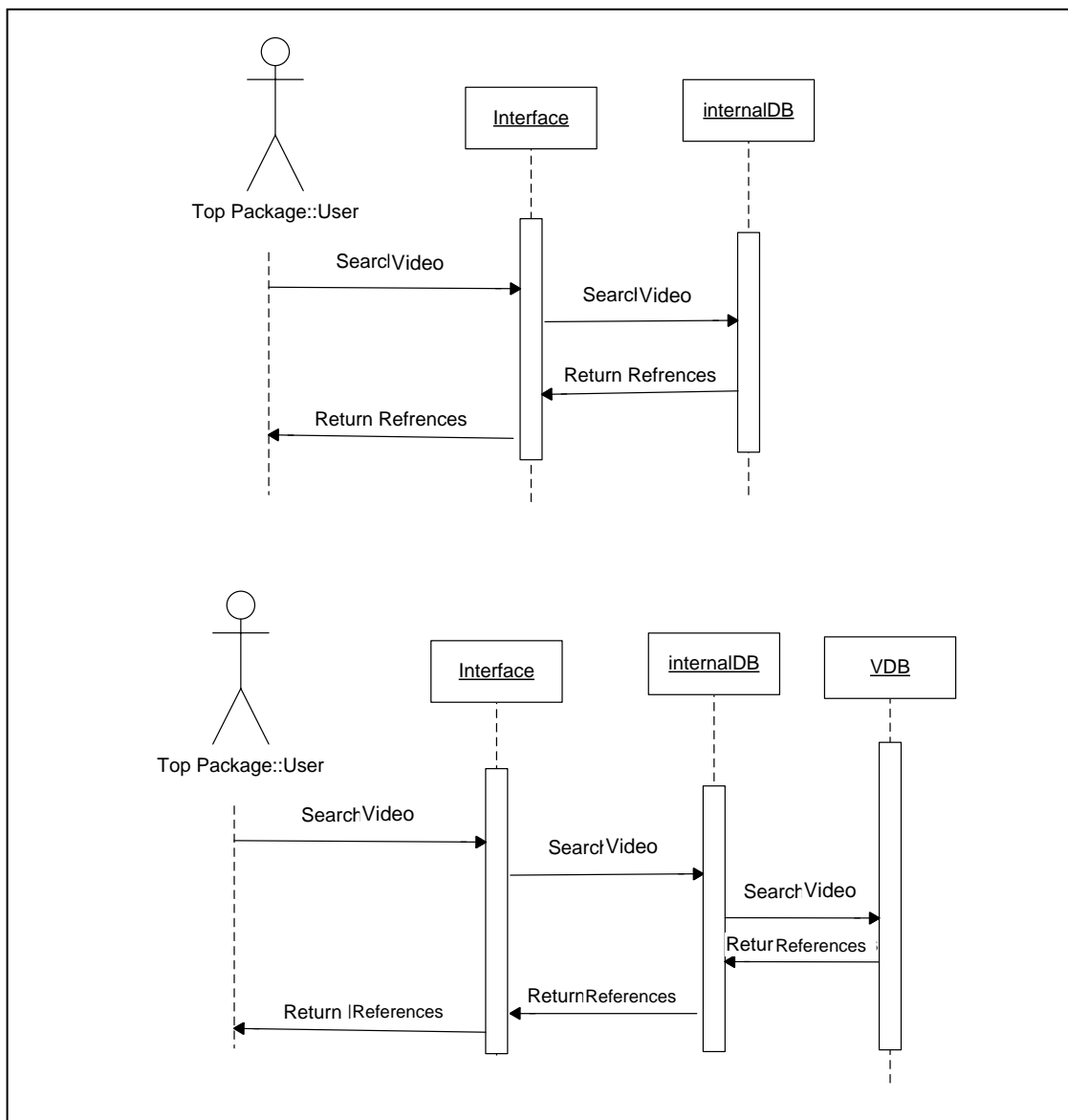


Figure 2.7: Sequence diagram of the Search video use-case scenarios; internal search and external search.

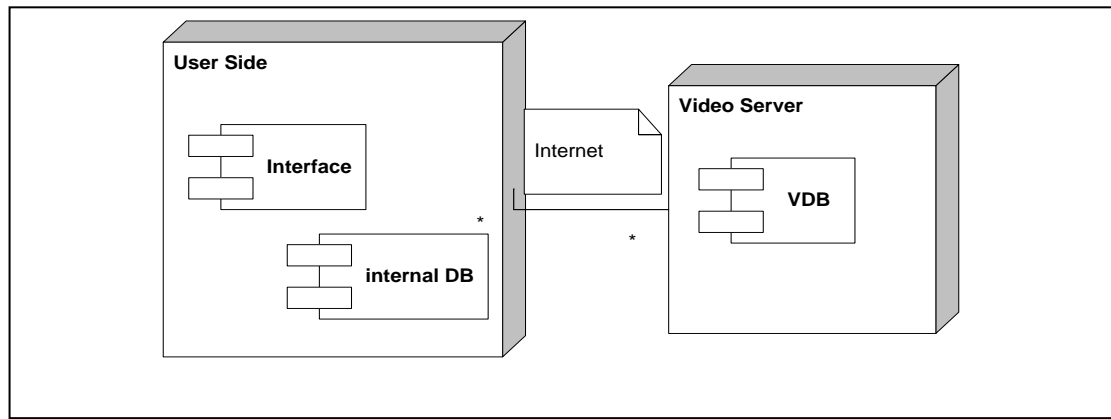


Figure 2.8: Deployment diagram for the video search system.

2.4 Software Engineering Modelling and CASE Tools

There are many types of tools that support software systems engineering which are available commercially or as open source. These tools provide diagrammatic modelling representations for data, flow-control, process, objects and structure of the software systems. The functionalities provided by these tools include model and consistency checkers, code generation, system simulators and even documentation generators[41]. These tools range from straightforward drawing tools that will allow the designer to represent the system in a specific modelling paradigm (i.e. UML) by providing basic representation functionality, to full computer aided software engineering (CASE) tools which provide more automated functionalities. This section will discuss these tools, and the functionalities that they provide to the software engineer. As a part of the methodology we are discussing in this thesis depends on drawing or CASE tools to represent the design model for the software system under study, we need to understand these tools and how they represent software systems.

2.4.1 Drawing Tools

The main goal of a drawing tool is to support the creation and management of graphical models of a software system [42]. A drawing tool usually supports only one or a few static or dynamic modelling paradigms. A drawing tool consists essentially of two main components. The graphical support system is responsible for creating the drawings of the model. Usually this model is represented as a graph of nodes and links. The second component is the information repository. This component is used to store and retrieve the model. It will also organise the information of the model to provide some functionalities like version management, consistency checking and documentation generation[42]. Newer versions of drawing tools may include import and export agents that allow the transformation of the model from one modelling or CASE tool, to

another. Drawing tools provide easy to use model generation tools, but they do not provide an aid to the software engineer in deploying a software engineering methodology. Also, they do not provide functionalities that will provide support in the management of the project or its phases.

2.4.2 CASE and iCASE Tools

CASE tools combine the graphical support of the drawing tools with more efficient functionalities like code generation, formal model verification, prototype generation and model animation, plus all the features usually provided by the drawing tools [41]. An example of a simple CASE tool is ArgoUML[43]. This tool is a widely used open source UML modelling tool that covers all the UML 1.4 standard diagrams. The ArgoUML provide a variety of functionalities that include forward engineering (code generation for java, C# and PHP), reverse engineering (for java Class/jar files), documentation generation, model checking using simulation and UML model Exporting and Importing using XMI.

Although CASE tools provide a wider range of functionalities and services than drawing tools, the problem of project management and phase distinguishing is that these could cause confusion in the state of the project and in the deployment of the development methodology. This inspired the development of the integrated CASE (iCASE) tools. iCASE provide a multiple CASE tools environment. These CASE tools cover every phase of the development methodology. The transformation between these case tools is hidden from the user side as all the integrated CASE tools have a common graphical interface and a common repository. The iCASE tools provide support for multiple modelling paradigms, with links between these models in the context of the project. For example, a project is modelled in UML and ER (entity relation) has models for the system and the database. An iCASE tool could provide the functionalities to model, verify and generate code for the goal system from these two different modelling paradigms.

2.5 CASE Tool Model Representation

The representation of the model in the CASE tool depends entirely on the implementation of that tool. In some cases, an engineer would need to utilise the functionalities of a different CASE tool other than the one he started the project on originally. The model transformation from one CASE tool to another was classified as a difficult task in the past. OMG has issued a standard model exchange language that is

used to export and import design models in CASE tools. XMI(XML Metadata Interchange)[44] is used to represent any MOF(Meta-Object Facility) Model to be exchanged between CASE tools. This section will discuss in detail, the XMI representation of the UML models. We will need this information in Chapter 7 where we will implement an XMI parser for the tool discussed in this thesis.

2.5.1 UML XMI Representation

As we saw in the previous section, majority of the later CASE and drawing tools offer the functionality to export UML models. The exported model is presented in a standard exportation schema (i.e. XMI documents). XMI documents are actually XML schemas structured in a standard defined by the OMG, this standard is usually reviewed and updated regularly to a newer versions. The XMI specification contains a complete pattern for syntax and encoding needed to export and import models, with complete DTDs for UML and other MOFs. The XMI standard we are explaining in this section is version 1.4 which was the latest version, the ArgoUML [43] (the UML modelling tool that we have adopted) can also export at the time of the writing of the UML-JMT tool. The latest version of XMI specification, at the time of writing this thesis, is version 2.1.1[45]. This version supports additional enhancements and repository-based configuration management for model-driven, team-based software development[46].

The XMI document explained in this section is taken from a file extracted using the ArgoUML tool. In this section, we will try to explain what an XMI document is, how it represents a UML model and how we can retrieve this model from the XMI document. The XMI standard explains how different UML diagrams and notations are represented. We will only concentrate here on the UML models used in the methodologies discussed in this thesis; these are Use-case, Sequence and Deployment diagrams. We will discuss the different parsing strategies that can be used to retrieve the UML notations from the XMI model. Throughout this section, we will use the same simple video search example explained in 2.3.4 as an example of a UML model.

```

<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp = 'Wed Jul 08
15:56:49 BST 2009'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>ArgoUML (using Netbeans XMI Writer version
1.0)</XMI.exporter>
      <XMI.exporterVersion>
        0.26.2(6) revised on $Date: 2007-05-12 08:08:08 +0200 (Sat, 12 May 2007) $
      </XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="UML" xmi.version="1.4"/>
  </XMI.header>
  <XMI.content>
    ... The UML Model Elements ...
  </XMI.content>
</XMI>

```

Figure 2.8: XMI File Structure

XMI File Format

A Sample for an XMI document defined for a UML model extracted by ArgoUML modelling tool is shown in Figure 2.8. The document is an XML file, which indicates that the XML version and encoding processing instruction must be shown at the beginning of the document, as the XMI schema declare the encoding is optional. The *XMI* root element, which indicates that this XML document is actually an XMI document, has two main nested sub-elements. The attributes for the XMI element include the *XMI schema version*, and the *name space* for the MOF model represented in this document (in our example it is a UML model), and the *date* of creation of the document. The first internal element of the XMI root node is the *header* element which contains documentations and declarations of the document. The documentation part includes naming the *exporter program* and its *version*. In the example from 2.3.4, it is ArgoUML with an exporter v0.26.2. The declaration includes the *type of model* represented in the document and the *version* of the XMI conversion specification. The second element nested in the XMI root element is the content element. Inside this element are the elements that represent the UML model and encapsulate all the sub-elements which represent this model's diagrams and notations. Each component, attribute or association is represented as an element. Each diagram has its specification nested inside it as elements[45]. Next, we will describe the representation of the use-case, sequence and deployment diagrams.

Use-case Diagram Representation

The use-case diagram consists of *actors* and *use-cases*, and *associations* between them[40]. These are represented in the XMI document inside the *model element name space* nested inside the *content* element. The actor elements are defined as follows:

```
<UML:Actor xmi.id = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EA6'
  name = 'user' isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
  isAbstract = 'false'/>
```

The tag *UML:Actor* defines that this is an actor of namespace UML. Each element in the XMI document that represents a UML notation is given a unique *xmi.id* which is used to define associations. This actor represents the only actor in the video example that symbolises the user of the system. The name of the actor is defined in the *name* attribute. The use-cases are defined inside the model element in the same way, but with a different name tag as *UML:UseCase*. In our example one of the use-cases defined in Figure 2.5 is the Search use-case and it will be represented in the XMI document as follows:

```
<UML:UseCase xmi.id = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EA8'
  name = 'Search' isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
  isAbstract = 'false'/>
```

The association is represented in the use-case diagram by introducing an *association element* tagged with *UML:Association* which holds an “xmi.id” and a “name” elements. Inside it, *connections* are defined, each connection having two *association end* elements that contain the *xmi.id* of the participating elements in the connection[45]. In our example the association between the actor ‘user’ and the use-case ‘search’ is defined as follows:

```
<UML:Association xmi.id = '...' name = 'SearchVideo' ...>
  <UML:Association.connection>
    <UML:AssociationEnd...>
      <UML:AssociationEnd.participant>
<UML:UseCase xmi.idref = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EA8'/>
      </UML:AssociationEnd.participant>
    </UML:AssociationEnd>
  <UML:AssociationEnd ...>
    <UML:AssociationEnd.participant>
<UML:Actor xmi.idref = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EA6'/>
    </UML:AssociationEnd.participant>
  </UML:AssociationEnd>
</UML:Association.connection>
</UML:Association>
```

The association tag is similar in all the association definitions for the different UML diagrams. The definition of the association for a specific diagram is defined inside the element representing this diagram.

Sequence Diagram Representation

Sequence diagrams are represented by a group of *components* (objects) with *messages* between them which define a processing scenario[40]. A sequence diagram is presented in XMI as an element tagged with the name ‘*UML:Collaboration*’. In its attributes are the scenario’s name and XMI id. The opening and closing tags for the sequence diagram defining the ‘Add video’ scenario shown in Figure 2.6 are as follows:

```
<UML:Collaboration xmi.id = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EC4'
name = 'AddVideo' isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
  == Sequence Diagram elements ==
</UML:Collaboration >
```

Inside the collaboration element there are elements representing the *components* participating in this scenario, *association* between these components and the *collaboration interaction* element defining the messages between the components. The components are defined as elements with the “*UML:ClassifierRole*” tag. The attributes for this element include the component name and XMI id. Inside this component is the multiplicity role for this component (we are not concerned with it in our tool). An example of the VDB (video database) in the sequence diagram shown in 2.6 is as follows:

```
<UML:ClassifierRole xmi.id = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EC8'
name = 'VDB' isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
...
</UML:ClassifierRole>
```

The association rules are defined for each interacting component using an association element similar to the one defined for the use-case diagram. The collaboration interaction part of the sequence diagram representation contains a group of *message elements* that define each message call interaction between the components. Each message element has attributes of name and xmi.id, and has three main child elements which are the *sender*, *receiver* and the *communication* connection. The sender and receiver elements contain the id of the sending and receiving component respectively. The communication connection element identifies the id of the association rule defining the connection between these two components. One of the messages in the ‘add video’ scenario is from the interface to the video database and is represented as follows:

```
<UML:Message xmi.id = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000ED0'
name = 'Add video' isSpecification = 'false'>
  <UML:Message.sender>
  <UML:ClassifierRole xmi.idref = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EC5'/>
  </UML:Message.sender>
  <UML:Message.receiver>
  <UML:ClassifierRole xmi.idref = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EC8'/>
  </UML:Message.receiver>
  <UML:Message.communicationConnection>
```

```

<UML:AssociationRole xmi.idref = '-119--61-27-76--3ca482dc:1222ca7a717:-
8000:000000000000ECC'/>
</UML:Message.communicationConnection>
</UML:Message>

```

Deployment Diagram Representation

Deployment Diagram is a collection of *nodes* grouping *components* in the same location or the platform [40]. In an XMI document, deployment diagram is defined as a group of elements representing the nodes with association elements, representing the connectivity between these nodes. The association components are the same format as the ones described previously for the use-case representation. The elements representing the nodes are tagged with the name “*UML: Node*” and as with the other UML notations has the attributes *name* and *xmi.id*. Each of the node elements contains a set of child elements representing the components in this node. In Figure 2.8, the deployment diagram contains two nodes representing the user side and the server side. The server node is represented in the XMI document as follows:

```

<UML:Node xmi.id = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EB8' name =
'Server' isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
<UML:Component xmi.id = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EBC'
name = 'VDB' isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
<UML:Component.deploymentLocation>
<UML:Node xmi.idref = '-119--61-27-76--3ca482dc:1222ca7a717:-8000:000000000000EB8'/>
</UML:Component.deploymentLocation>
</UML:Component>
<UML:Node.deployedComponent>
<UML:Component xmi.idref = '-119--61-27-76--3ca482dc:1222ca7a717: 8000:000000000000EBC'/>
</UML:Node.deployedComponent>
</UML:Node>

```

2.5.2 Working with an XMI Document

Although the use of XMI to express the object model of software systems and generate implementation classes from design models is a hot topic in research and development, the existence of tools and libraries to support the extraction and management of UML models, other than class diagrams, is limited. There have been some attempts to construct a library that reads an XMI file and arranges all the model diagrams in the form of objects that can be used and analysed, but most of these attempts are in their early stages or even prototypes. Most of the programming community in the programming forums advise each other to build their own parser that will fulfil the programmer’s specific needs using the available XML parsers libraries, given that XMI document is actually an XML document. This will cause an overhead in the development as the developer will need to know the XMI schema for UML. Even still, this was the method we used in the development of the parser for the UML-JMT tool.

When trying to parse an XML document, a programmer has the option of using one of two kinds of parsers which differ from each other in the way that they deliver the XML elements, either as event driven, as in SAX(Simple API for XML)[47], or a one that provides an entire structure document, as in DOM(Document Object Model)[48]. We will discuss them briefly next.

SAX

SAX is an event based API that allows the serial parsing of an XML document. The user will define a set of event handlers that will execute when the parser encounters one of the events (i.e. finding an element node, text node, XML instruction or comment). The event is fired at the beginning and the end of the encounter (i.e. opening and closing tags)[47]. As SAX does not have an internal structure to represent the XML document, SAX parser does not require large space of memory, therefore, a SAX parser will not face any difficulty parsing large XML documents. The streaming nature of SAX and the fact that it does not require a structure makes it run faster than DOM. On the other hand, the fact that an overall picture of the document cannot be given by SAX makes it harder to implement some programs that require a complete access to the document, like some types of validation and XSLT and XPath which require to have access to any node in the tree all times [49].

DOM

DOM is a defined standard for accessing and analysing XML documents. The DOM parser works by loading the entire XML document into a tree structure. The root of the tree represents the root element in the document object and the internal elements, attributes and text, as the child nodes. DOM parser provides APIs that allow the programmer to traverse the tree in all directions. It also allows the user to check the type of node or retrieve all the elements of a specific type. Figure 2.9 demonstrates a partial view of a DOM tree structure of an XMI file with the XMI tag as the root and elements representing the different UML notations, and diagrams as child nodes. Although only the elements were displayed in the figure, attributes are also represented as child nodes. The DOM parser offers an easy to navigate, whole document approach to the user. On the down side, the footprint of the DOM tree on the memory may cause difficulties in parsing large-scale documents. The java DOM library was used in the light weight XMI parser that was implemented for the UML-JMT tool. We chose to use DOM because the

parser passes the document object from one model extractor to the other. We will explain this parser later in Section 7.1.

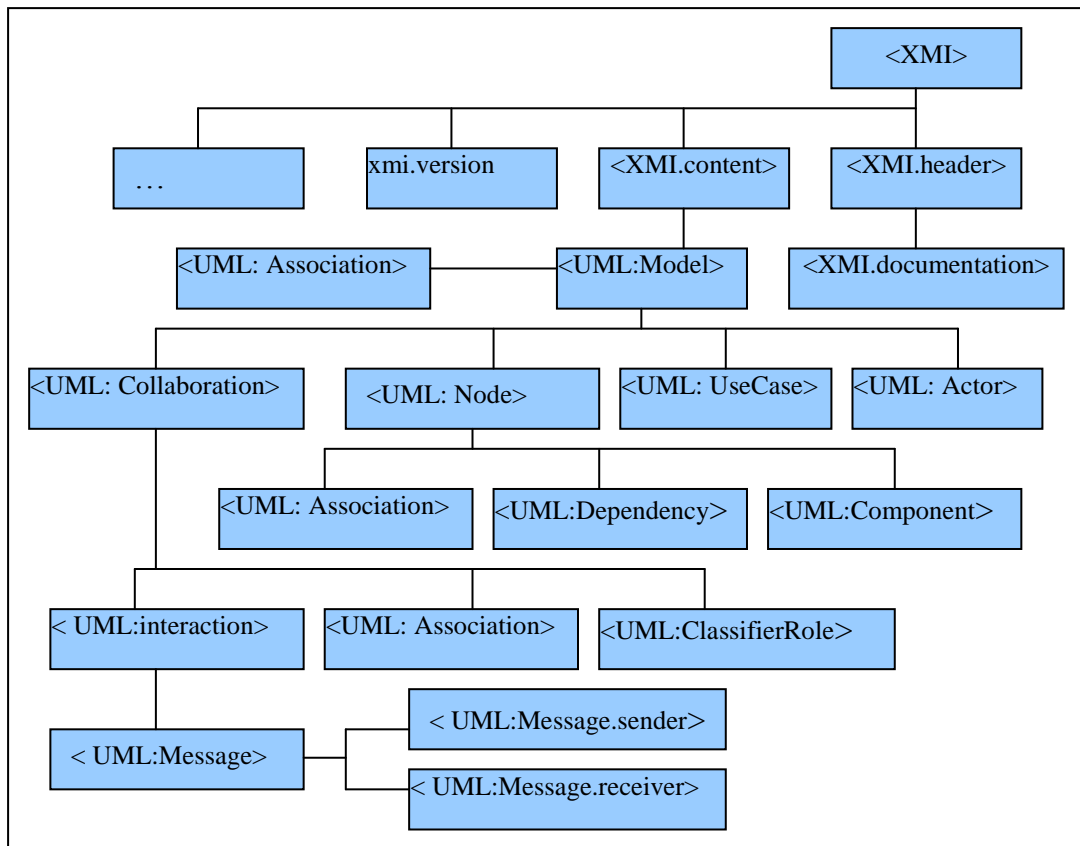


Figure 2.9: Partial DOM tree for an XMI document with use-case sequence and deployment diagrams.

2.6 Summary

The main objective of the thesis is to provide a performance evaluation methodology in the context of software engineering terminology. As performance is one of system's characteristics that are affected by the whole system, the integration of the performance engineering into software engineering will depend on the availability of the requirements. This is why we distinguished the software engineering paradigms according to the availability of the requirements to conventional (i.e. waterfall development paradigm) and agile. This Chapter discussed the relevant terminology related to the software engineering domain. In this chapter, we defined software systems engineering, discussed some of the software engineering schools of thought and the different development paradigms available. Then we discussed requirement engineering and validation (as a task of software engineering) and explained the importance of validation of performance non-functional requirements. As this thesis discusses the UML model transformation approach; we have defined UML modelling as it is the standard modelling paradigm used for representing the architectural aspects of a

software system. This chapter also provided background knowledge about CASE and modelling tools as well as background about XMI model representation.

Performance of a computer system is a behavioural aspect of the system which is concerned with resources in the system's environment. These resources include time and usage of the system's physical artefacts. *Performance evaluation* is the process of assessing the performance of the software system. The performance aspects of a computer system are evaluated by calculating performance related measurements called the *performance indices*. These indices relate to the speed of response, usage of resources, and the usage of the system in the context of the organisation. The performance evaluation task can be carried out by direct measurement of the existing systems or by *modelling* the projected systems. The importance of system performance evaluation arises from the need for methods for analysing and optimising existing software systems to improve its performance aspects. Furthermore, performance evaluation could be used to assess the design of projected systems, by validating that a suggested design would provide the expected performance measures. This chapter will present the process of software systems' performance evaluation. Section 3.1 will describe the performance evaluation task as inputs, processes, and outputs theme. Sections 3.2, 3.3 and 3.4 provide an in-depth description of some of the key performance evaluation paradigms used in software performance evaluation. As this thesis is oriented toward non-deterministic performance evaluation in the early stages of software development, Section 3.5 will discuss the suitability of the previously discussed performance evaluation paradigms for this task. Section 3.6 will discuss the tools used to evaluate a software system's performance, by setting criteria for evaluating them.

3.1 Software Performance Study: Modelling and Evaluation

A software system performance study involves analysing the performance characteristics of a system in response to changes in the system's environment variables (i.e. number of users, number of servers etc). The goal of a software performance study is to compare the actual system performance indices to the anticipated ones, and eventually, tune the system to achieve the *best* performance that can be gained from the

system. Figure 3.1 shows a diagram illustrating the process of a performance study. A performance study starts by identifying a set of performance objectives; these objectives are set to be the expected performance indices clarified in the system specification document (i.e. non-functional requirements specification). The next step involves constructing an abstract representation of the system, called a *Model*. This model will only concentrate on the aspects of the system that affect the performance indices under study. This model can be viewed as a function representing the system, with variables representing the change in the system's environment. The next step involves evaluating the system performance model and generating the real performance indices. This depends mainly on the performance modelling paradigm used, as we will see in the next section. The next step involves analysing the resulting performance indices and comparing them to the ones defined as objectives, and constructing plans to achieve these objectives. These plans are translated to tunings and alterations on the design or specifications, which in return, will require a new performance study to inspect how these changes affected the system's performance. In this section, we will further explain the modelling, evaluation and analysis steps.

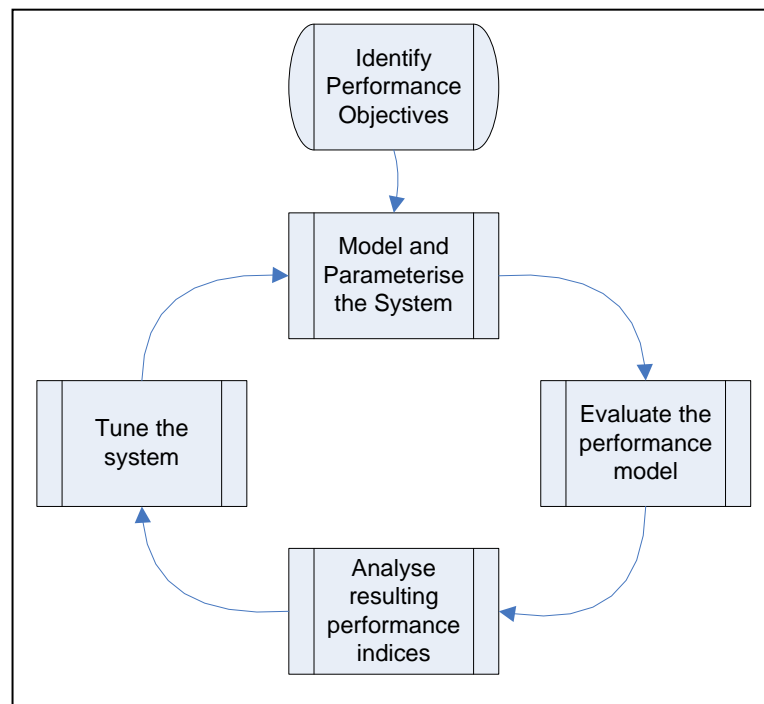


Figure 3.1: Steps of a software system performance study

3.1.1 System Abstraction and Performance Models

A *performance model* of a software system can be defined as an abstract view of that system which focuses on the artefacts that define the performance characteristics of the

system under study. As mentioned earlier, performance models work as functions representing the system's *behaviour* (and/or) *structure*, and providing a relation between change in the performance defined by the system characteristics and performance indices defining the system. The behaviour of the system is characterised by the events and actions that define the system. *Performance characteristics* include performance related state variables such as job arrival rate for the system or the service time for one of the system components. *Performance indices* are the measurements used to indicate the performance of the system (i.e. throughput, response time, utilisation ... etc). The process of abstracting a system to a performance model depends mainly on the nature of the performance study. The main factors that control the abstraction process are the performance measure required and the controlling performance variable.

There are a variety of performance modelling techniques, each of which has its own uses and limitations. Performance modelling methodologies can be distinguished as three main trends. These are as follows:

- *Simulation*: Where a prototype of the system is abstracted, programmed and executed with different control variables and performance indices are measured from the different simulation runs. We will discuss simulation in Section 3.2.
- *Analytical Modelling*: Where the systems' architecture or state space are modelled visually or symbolically and then transformed to mathematical equations that can be solved analytically or by simulation, to calculate estimates of the performance indices. Examples of analytical models are *Markov Chains*, *Queuing networks* and *Petri-nets*. We will discuss them in more detail in Section 3.3.
- *Formal Modelling*: Where the structure and behaviour of the software system is translated to algebraic equations that can be translated to analytical models, which can be solved to provide meaningful performance indices. Examples of the formal modelling techniques are *Process algebra*, and *PEPA* models. We will discuss them in more detail in Section 3.4.

Choosing between these performance modelling methodologies depends mainly on the system type and the stage in which the performance study is conducted. As performance evaluation studies are necessary throughout the system life span, the different performance evaluation methodologies can be seen as complementary to each other. Different performance evaluation methodologies can be used in different stages of the system development and run. As an example, in the design phase, the amount of

information about the system is limited and a performance study is needed to justify a design or to choose between different designs alternatives. In this stage, analytical modelling appears to be the “best” tool to conduct such a study. As the system progresses and goes through the maintenance phase, more definite and exact results are needed where measurements of the system can be undertaken to aid tuning. We will discuss some of the main performance modelling paradigms later in this chapter.

3.1.2 Performance Model Evaluation

Performance model evaluation can be defined as extracting performance characteristics of the system represented by the performance model. These performance characteristics represent the performance indices required by the performance study. The variety of performance indices that can be extracted depends mainly on the performance evaluation paradigm being used (as simulation provides no limit to the extracted indices). The common and most notable performance indices studied in most software systems are:

- *Throughput (X)*: This measurement represents the rate of completed ‘jobs’ over a period of time (T).
- *Utilisation (U)*: This measurement represents the rate of *usage* of the system’s resources.
- *Service Time (S)*: Represents the average time required by a resource to accomplish a job.
- *Response Time (R)*: Time interval from issuing a request to when a response is returned.

The process of extracting these performance indices from a performance model depends utterly on the performance evaluation paradigm used, as mentioned in 3.1.1. However, the equations used to drive these values are known as the *operational laws*. Operational laws are a set of fundamental laws and their derivations, which are used to calculate the performance indices from basic measured performance quantities. These quantities are T (the time in which the system was monitored), A (the number of jobs arrived in time T), C (number of jobs completed at time T) and B (the length of time that the system was busy). The operational laws state:

$$\text{(Arrival Rate) } \lambda = A/T$$

$$\text{(Throughput) } X = C/T$$

$$\text{(Utilisation) } U = B/T$$

$$\text{(Service Time) } S = B/C$$

These fundamental equations are extended to other laws, such as utilisation law, Little's law, and general response time law ...etc. With the help of making general assumptions about the system, the above performance indices can be derived. Later in this chapter, we will discuss in more detail how we can *solve* or *execute* a performance model.

As discussed earlier, the degree of correctness of the resulting performance indices depends mainly on the performance modelling paradigm being used. Simulation models tend to provide a high degree of accuracy and model details, regardless of the type of system being used. On the other hand, analytical and formal models require the model to satisfy some constraints in order to extract performance indices with a high degree of accuracy, therefore, the performance measures gained from an analytical or formal performance study are expressed as approximations.

3.1.3 Performance Analysis

Analysis of performance is required for one of the following tasks:

- *Design justification and experimentation*: This type of performance study is usually conducted in the early stages of the system development life cycle. If there are multiple candidate design alternatives, performance studies are used to choose the best design that will implement the non-functional requirements specified for that system. This can be done by studying the performance indices for the different design alternatives (if there are alternatives) and comparing them to the required specifications.
- *System tuning*: When a system is experiencing performance problems, a performance study is conducted to locate the source of this problem. The system is modelled, and the performance indices are calculated. Alterations are made on the model to locate the problematic parts of the system. Changes are then suggested according to change to the performance indices.
- *Specifying systems limitations*: In any system, it is necessary to discover its limitations in order to prevent unexpected crashes. Using performance studies, a system could be tested to find its breaking point.

3.2 Performance Evaluation: Simulation

Simulation is defined as an imitation of the operations of a process or system, monitored over a period of time[50]. Simulation performance study involves the construction, implementation and execution of what is called a *simulation model*. This model is based on the structure and behaviour of the system, and the performance characteristics

of the system. Simulation performance studies provide a high degree of accuracy and detail, which can provide performance indices, which can express more accurate values. Evaluation of a simulation model involves implementing and executing a simulation program. This will provide no limitation on the types of system architectures and behaviours being modelled, as in other modelling paradigms that we will see later. This section will discuss briefly, simulation performance study methodology. In the next subsection, we will discuss the process of conducting a simulation study. In 3.2.2, we will discuss the advantages and disadvantages of simulation.

3.2.1 Simulation Study Steps

A simulation study can be described as the process of monitoring the system *state* over a period of time. The system state is defined by a collection of variables describing the system in a given point of time. These variables are chosen according to the nature and goal of the performance study. State variables can have a *discrete* or *continuous* nature. The change of the system state is denoted as an *event*; these events can occur within the system itself or in the surrounding environment. Events may occur at a discrete point of time or continuously. Accordingly, simulation can be distinguished into three main types:

- *Continuous time simulation*: In this type of simulation, the system state will be monitored and changes recorded continually over time. This type of simulation behaviour is described using differential equations. Continuous time simulation is usually used in scientific analysis software.
- *Discrete time simulation*: In this type of simulation, the state of the system is captured in each time cycle. Note that the state may not change for multiple clock cycles.
- *Discrete event simulation*: This type of simulation is used with systems that have a state that does not change continually with time. The discrete nature of computer systems makes this kind of simulation more suitable to use.

J. Banks *et. al.* has illustrated in their book “*Discrete-event system simulation*”[50], the steps of conducting a simulation study. It starts by formalising the problem and understanding the system to be simulated, and then objectives of the study are set. These objectives are formulated in the form of questions that need to be answered at the end of the study. The nature of the objectives will determine whether simulation is the best paradigm to accomplish the study objectives. The plan of the study will specify its

stages and the resources needed to execute it. In the next phase, the simulation model will be abstracted in a form that will allow all the required performance indices to be obtained with minimal complexity. The next phase involves collecting the data required to perform the simulation study. This data include traces of an existing system, or information about the work load. The next phase involves model translation. At this stage, the simulation model is implemented into a simulation program. This can be done using special purpose simulation languages (e.g. SIMULA[51]) or using conventional programming language, equipped with simulation libraries (e.g. SimPack[52]). After the simulation program is verified for errors and validated for representing an accurate representation of the system under study, the experiments implementing the simulation study are decided. This includes variables defining the length of the simulation run and the number of replications. At this stage, the simulation program is ready for execution. At the execution time, the monitored variables defining the system state are analysed and performance indices calculated. At the end of each run, a documentation of all the outcomes is produced. After the analysis phase, a document containing a description of the study and documentation of the simulation program and simulation results is formed[50].

3.2.2 Simulation: For and Against

As we stated above, simulation performance studies are seen as the most flexible approach to computer performance modelling. This comes as a result of the degree of accuracy provided by the simulation results and the unlimited, unrestricted modelling spectrum allowed in simulation studies. There are almost no limits to the range of performance measures that can be monitored and calculated in a simulation performance study. Furthermore, there are no assumptions forced on the system that could restrict the use of simulation for specific system architecture. All of this comes at a cost, which is translated in the large computational cost and resource requirements of a simulation study. The costs arise from the enormous effort required to conduct a simulation study. This effort is spent in the analysis and development of the simulation program. In addition, the results of a simulation program need further efforts to interpret them into useful performance measures.

3.3 Performance Evaluation: Analytical Modelling

Analytical performance modelling involves building notational or formal models which represent the modelled systems' structural or state space behaviour. Analytical modelling is regarded as one of the cost efficient performance prediction techniques.

The degree of accuracy of the performance measures gained from an analytical performance study depends mainly on the complacence of the system to the assumptions specified in each of the analytical modelling paradigms. In a study for the accuracy of the throughput, utilisation and response time for analytical models [53], analytical results were compared to numerical and simulation results and the error margin was 10% for throughput and utilisation, and around 30% for response time. Analytical performance analysis will provide a low cost, sufficient solution for tasks like capacity planning and design aid. This thesis is concentrating on the use of analytical models as a design aid in the early system development stages.

There are a number of analytical modelling approaches in literature, we are only concentrating on Markov Chains and Queuing Networks, as these are the basis for the two methods discussed in this thesis. This section will discuss three of the most well known paradigms in analytical performance modelling. Section 3.3.1 will discuss Markov chains, Section 3.3.2 will briefly discuss Petri-Nets, and finally Section 3.3.3 will discuss queuing networks.

3.3.1 Markov Chains

Markov chains form the basis for model-based analytical performance evaluation in many areas of science and engineering. It can be described as the low level language for modelling. The Markov chain is named after the pioneer mathematician Andrei Markov who introduced the finite-state Markov chains. The use of stochastic Markov models in performance evaluation tasks can be described in two main activities. The first use is the evaluation of the probability of an observed behaviour, for example, the probability for the occurrence of that behaviour (i.e. the buffer is full). The second activity is to find the best design in terms of performance; this is done by observing the different behaviours that a system can take and adjusting the system's design and parameters so that the design can deliver the best performance possible. By solving the Markov model, a series of performance indices and observations can be obtained from the model. Haverkort [54] had two categories for the outcomes of such a performance study. They are; *system oriented* (i.e. utilisation) and *user oriented* (i.e. waiting time, throughput).

A Markov chain is a *stochastic process*, with all the random variables constricting this stochastic process have the *Markov property*. A stochastic process is a set of random variables $\{X_k, k \in K\}$ where K is known as the *index set* which is the controlling index

for the change of the random variables. Markov chain models come as discrete and continuous time models. For continuous time models, K will represent time. The Markov property states that the future value of a random variable X_k depends only on its current value and not on any previous values. This is called the *memory less* property:

$$\Pr\{X_k=i | X_{k-1}, X_{k-2}, \dots, X_1\} = \Pr\{X_k=i | X_{k-1}\}$$

A Markov model is a finite automaton containing a set of distinct states that a system can take, known as the *state space* S . Starting from an initial state, the model represents the state transitions from the current state, to another state according to a set of probabilities associated with the states, known as *Transition Probability*.

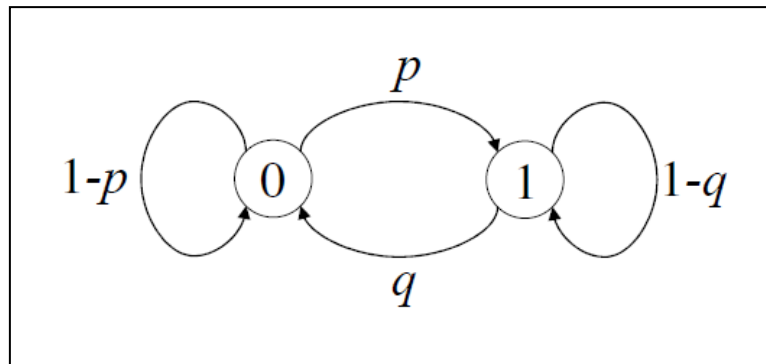


Figure 3.2: state transition diagrams of a Markov Model example with two states.

Figure 3.2 shows an example of a Markov model with two states. The states are labelled with numbers. At the time of state change, the decision of the next state will depend only on the current state and is controlled by the transition probability:

$$p_{ij} = \Pr\{X_k=j | X_{k-1}=i\} \quad \forall i, j \in S.$$

The transition probability has to comply with the following rule: The total transition probability from state i to all possible states must be equal to 1:

$$\sum_j p_{ij} = 1$$

Software system is modelled as a Markov chain by abstracting the system as a set of states (S) that represent all the states that would have an effect on the system's performance characterisation. As we said earlier, the process of choosing an abstraction of the state depends mainly on the objective of the performance study.

Deriving performance indices from a Markov model depends on calculating the *Steady state probability distribution*. This represents the probability distribution of the transition from one state to another when the system enters into a regular pattern behaviour. The study state theorem states that, for every finite, time homogeneous, irreducible Markov process there will be a steady state probability distribution. This

distribution will not change as the model changes or states progress in time. Gaining the stationary distribution in a Markov model involves the solving of the *Global balance equation*. The global balance equation is an equation extracted from the *probability flux* for a specific state (probability of transition from one state in time to another). The study state assumption states that the performance study will take place when the system enters in the equilibrium stage. This means that the system is in a state where its behaviour is regular and predictable. Using this assumption, we can declare that the total flux out of a state is equal to the total flux into a state [55]. First we define $\pi_k(i)=\Pr\{X_k=i\}$ for all $i \in S$; Where π is a vector containing the probability distributions of each state. The global balance equation will be:

$$\sum p_{ij} = 1$$

$$\pi(i) = \sum_{\forall j} \pi(j)p_{ji}$$

Probability distribution will be used to calculate the performance indices. For example, utilisation of a device can be calculated as the total probabilities that the system is in a state where the device is being used. *Rate-based* measures (i.e. throughput) are related to measures in which some event occurs. This will be the product of the rate of the event and the probability that this event has taken place [55]. Operational laws are used as will to calculate other performance indices[56]. Next we will describe the stationary distributions for both discrete and continuous time Markov chains, but first we will clarify the assumption that must exist in a Markov chain for it to be solvable.

Assumptions of Markov Chains

For a Markov chain to be solvable by global balance equations, there are some properties that Markov models have to satisfy. These are as follows:

- A Markov chain is *irreducible*: This means that all the stats can be reached from all other states. For any states $i, j \in S$, state i is said to be *reachable* by state j iff:

$$P\{X_n=j | X_0=i\} > 0 \text{ for any } n \geq 0 \text{ where } n \in K$$

States i, j are said to be *commute* if these states are reachable to each other. A Markov chain is said to be irreducible if all of its states are commute.

- A Markov chain is *positive recurrent*: This means that a state visited must have some probability that it will be visited again. A state is recurrent if it has a finite hitting time of that which is:

$$\Pr\{X_k=i \text{ for some } k > 1 | X_1=i\} = 1$$

Positive recurrent state is a recurrent state with a *finite expectation*, e.g. if T_i is the time between visits to state i , then i is positive recurrent if $E(T_i) < \infty$.

Discrete Time Markov Chain

In Discrete time Markov chains; state change is carried out after *fixed time slots*. The system modelling depends on the behavioural modelling of the system represented by state change. Markov chains are represented as either *state transition diagrams* (as in Figure 3.2) or as *probability matrix*. The probability matrix P of a Markov model with n states is a $n \times n$ matrix with the transition probability for state i to j is in the i^{th} row and j^{th} column of P . The probability matrix for the Markov chain in Figure 3.2 is:

$$P = \begin{bmatrix} 1-p & p \\ q & 1-q \end{bmatrix}$$

From the global balance equation defined above, we can conclude that:

$$\pi_{k+1} = \pi_k P$$

π can be gained by solving the above equation. For the example, in Figure 3.2 we can find that solving the equation by linear algebra, the values for vector π are:

$$\begin{aligned} \pi(0) + \pi(1) &= 1 \\ \pi(0) &= \frac{q}{p+q}; \pi(1) = \frac{p}{p+q} \end{aligned}$$

Continuous Time Markov Chain

Continuous time Markov chain is a Markov chain with an index set represented by time (T). We say that the set of random variables $\{X(t): t \geq 0\}$ is a continuous time Markov chain if:

$$P\{X(s+t)=j \mid X(u); u \leq s\} = P\{X(s+t)=j \mid X(s)\}$$

This is the Markov property for a continuous time Markov chain. In a continuous time Markov chain model, the dynamic behaviour of the system is modelled by the transitions between the states, and the time spent in each state (*sojourn time*) which usually represents the processing time. From the memoryless property of Markov chain and the property transition that does not change over time, we can conclude that the distribution of time between the changes of states does not depend on previous states. This means that the sojourn time is memoryless[56] and therefore, the only probability distribution to represent time distribution between changes of states, is the exponential probability distribution function. If T_i is the sojourn time for state i and q_i is the total transition rates for any state $i \rightarrow j$:

$$q_i = \sum_{i \neq j} q_{ij}$$

Then we can say that:

$$T_i \approx \text{Exp}(q_i)$$

And further, the *transition probability* from i to j can be calculated as q_{ij}/q_i .

In continuous time Markov chain, the transition matrix has a special form and is called the *generator matrix* Q . In Q the entry of the i^{th} row and j^{th} column is q_{ij} where $i \neq j$. the diagonal elements are chosen to make the sum of all rows equal to zero:

$$q_{ii} = -q_i$$

Calculating the steady state probability distribution π for continuous time, Markov chain depends on the global balance equation. The global balance equation for continuous time Markov chain is:

$$\pi_i \sum_{i \neq j} q_{ij} = \sum_{j \neq i} \pi_j q_{ji}$$

After normalizing the global balance equation, the general form of the equation will be:

$$\pi Q = 0$$

Solving this equation using linear algebra with the equation:

$$\sum_i \pi(i) = 1$$

Will extract the values of vector π , which will be used to calculate the performance indices of the system.

Markov chains models provide flexibility in modelling any system type and representing any behaviour. The only downside of Markov chains comes in what is known as *state explosion*. In large and complex systems, the number of states could make the model difficult to solve. As a result, the complexity of solving the global balance equation increases as the state space grows.

3.3.2 Stochastic Petri-Nets

Petri nets were introduced in 1964 by Karl Petri [57] as a graphical description language used to model large and complex systems' concurrency and synchronisation. The first Petri Nets were concerned with the test of systems for functional correctness (i.e. deadlock, liveness...). The need to study quantitative properties of systems led to the addition of a time element to the models, which introduced *Stochastic Petri Nets* (SPN)[58] and *Generalised Stochastic Petri Nets* (GSPN)[59]. SPN came as a solution for the Markov chain state explosion problem. SPN can be seen as a higher level language that uses a performance analysis technique based on Markov theory. The solution of a SPN corresponds to the solution of an underlying Markov chain which can be gained by modelling the SPN states, as we will see shortly.

SPN models represent systems as a set of *Places* and *Transactions*, and a set of *Arcs* connecting places and transactions to each other. In the graphical representation, the places are drawn as open circles, transitions as bars, and arcs as arrows. Figure 3.3 shows an example of a SPN model for a system that has two parts that might fail and need to be repaired. The status of the system is modelled by the whole graph, unlike Markov models where each node in the graph represents a status of the system. SPN uses what is known as the *token game* to describe the behaviour of the system. The system is modelled in multiple states and in each time step, the *tokens* (modelled as solid circles located inside the places) will move (*fire*) from one place to the next state(s) according to set rules:

- A transaction is *enabled* if it has tokens placed that it is connected to as output.
- Only enabled transactions can fire.

The *Marking* of SPN models represents the distribution of tokens inside the parts of the model; it is represented by the number of tokens in each place. The markings are used as model status records. The *reachability set* represent all the reachable markings for the model from the initial marking. The solving of a SPN depends on building an equivalent Markov chain with the state space represented by the reachability set and the transition rates between the states of the Markov model presented by the transition rate between the markings. By solving this Markov model, we can extract the required performance indices of the model.

We stated earlier that the motivation for introducing Stochastic Petri-Nets is to overcome the state explosion problem found in Markov chains. This is demonstrated in Figure 3.3 where we have a model of a system that has two replicated parts that provide that same service. The model is to represent the availability of the system. In a Markov chain the number of states depends on the number of parts modelled in the system, that is, if the system has another part, we will need another state. For the SPN model we will only need to add a new token in the initial marking, without changing the model. Although SPN solution is based on a Markov chain, the problem of state explosion is lessened as there are normalising algorithms and simulation tools that would help in the solution of SPN models. The drawback of SPN is its lack of generality advantage that was available in Markov models, as it is difficult for it to model certain types of system architectures, such as systems with specific scheduling schemes for sharing resources[60].

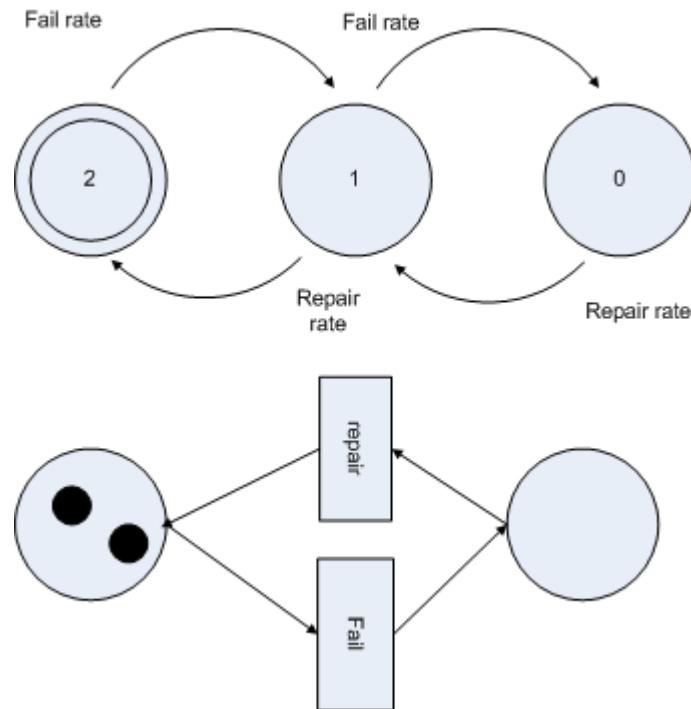


Figure 3.3: Comparing Markov Chain Model to an equivalent SPN model for system frailer status of a system with two parts.

3.3.3 Queuing Networks

One of the main drawbacks of analytical performance modelling techniques was in the lack of intelligibility in the abstracted models. Usually that abstracted model represents a system's status behaviour in a form only understood by the modellers themselves. The use of queuing theory in computer performance studies started in the 1960s, and although this use was in its simplest form, it was obvious from that time that queuing models were the future of computer performance evaluation. The first use of queuing theory in computer based performance evaluation was to model time sharing systems[61]. The study was to evaluate different CPU scheduling and disk management strategies. At first, queues were used as a unit that represent the entire system but later, queuing networks were used to get more realistic models representing the components of a system. Computer systems can be viewed as a set of loosely coupled components (software or hardware) which interact with each other by executing *jobs* or *transactions*. This view of computer systems made queuing networks more instinctive to use as a modelling technique for evaluating performance.

A queuing network is a representation of the system as a set of *service centres* which are connected to each other in a topology that represent the systems' architecture. A service centre is a queuing system that consists of a *queue* and a *server*. The parameters that define a queue are the *queuing discipline* and its *capacity*. Queuing discipline defines the algorithm used to control the order of jobs in the queue. There are some known queuing disciplines which are considered in most of the queuing networks solutions, such as FIFO (first come first out) and LIFO (last in first out). The capacity of the queue defines the size of the buffer that can hold waiting jobs. Another parameter for the queue which can exist in some simulation solutions of queuing networks is the *drop strategy* which defines the strategy used to reject incoming jobs to the service centre after the queue buffer becomes full. The parameters that define a server are the *service time distribution* and the *number of servers*. Workload is defined in a queuing network by an *arrival rate* or the *number of users* depending on the type of the queuing network. Queuing networks can be *open* or *closed* depending on the behaviour of the job inside the network. In open networks, jobs tend to *leave* the network after they are completed, whereas in closed networks, they will return in another round. Queuing networks can have multiple *classes* of jobs, each with its own workload and *routing strategy*.

The A/B/X/Y/Z notation is used to describe a queuing system which was suggested by D. Kendall. It defines the type of a queuing system by describing the properties and parameters that define it. The notation A/B/X/Y/Z stands for:

A - inter-arrival time distribution

B - service time distribution

X - number of servers

Y - system capacity (in the queue and in service)

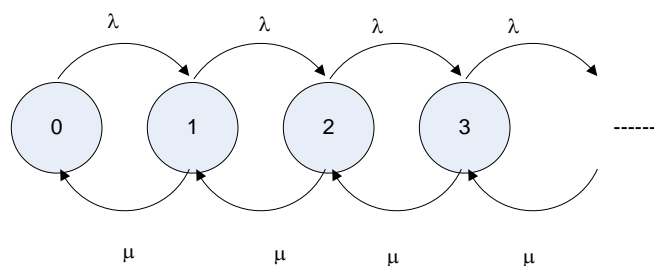
Z - queuing discipline

The default value for Y is ∞ (i.e. there is no limit to the buffer) and for Z is FIFO, if the queue have Y and Z as default the type can be written as A/B/X. the inter arrival distribution and service distribution can be of type M (Markov exponential distribution), D (deterministic), G (general) ... etc. An M/M/1 queue is a queue with an exponential arrival rate, an exponential service rate and a single service centre, unlimited queue and FIFO queuing strategy.

Computer systems modelling using queuing network models can be employed on different levels of abstraction. The queuing network may represent the underlying

hardware or components (software/hardware) architecture of the system. In this thesis, we are concentrating on the component view of the system. The process of modelling a system as a queuing network starts by defining all the service providing components. These components will be represented in the system as service centres. The characterisation of these components (workload, service time ... etc) can be gained from the specification of this component (i.e. if the component is a DBMS, the specifications of the DBMS will include the performance characteristics of this component). The classes of the jobs can be defined from the type of processes or scenarios running in the system. The topology on which the service centres are connected depends on the architecture of the system. *Delays* can be added to a queuing network model, to add overheads like thinking time and network latency. Delays are a special kind of service centres, where the queues are infinite and the jobs remain for a time, defined by a *wait time distribution*. Figure 6.9 shows a queuing network for a video search system.

A queuing network can be solved either analytically or by simulation. Simulation provides a general technique where a variety of system architectures and queuing discipline can be modelled. Moreover, simulation provides more accurate results. Simulation is used usually with *non-product form* queuing networks. These are queuing networks that do not apply the assumptions insisted on by the algorithms defined for analytical queuing networks solution. The analytical solution of a single M/M/1 queuing system relies on defining a continuous time Markov chain with a state depending on the systems population, as follows:



Where jobs arrival rate is defined as exponentially with parameter λ , and the service time is also defined exponentially with parameter μ . For some classes of queuing networks with *general* arrival and service distribution, they can be modelled with a particular discrete time Markov chain named *birth-death Markov process*. A queuing network solution is based on defining a Markov chain with a state space defined by the number of customers in each service centre queue. The computational complexity of the

analytical solution limited its generality. *Product-form* queuing networks are a distinct family of queuing networks that have simple and efficient solutions. In the next two sub-sections, we will discuss the analytical and simulation solutions of queuing networks. First we will define product form queuing networks and their analytical solutions and limitations. Then we will discuss an alternative approach for solving queuing networks, this approach involves using operational laws previously discussed. This approach is called *Mean-value Analysis* (MVA). After that, we will describe the *Extended Queuing Network* EQN by discussing its properties and solution. Note that we will be using EQN in the methodology discussed in Chapter 6.

Product-Form Queuing Networks

Product form queuing networks are defined as a class of queuing networks that satisfy a set of assumptions. These assumptions qualify this class of queuing networks to be solved analytically, using *product-form equations*. The importance of the product form solutions for queuing networks lies in the reduced complexity that these solutions provide, as the complexity of these solutions grows linearly with the number of service centres, compared to exponential growth observed in Markov chains. This will provide balance between the accuracy of the performance results gained and the efficiency of the model evaluation and analysis[62]. Product-form networks have some properties that will help in producing models with different levels of abstractions for a system. One of these properties is the *aggregation theorem* described in [63]. The aggregation theorem allows the replacement of a portion of the queuing network with a single queuing system that has the same performance characteristics of the replaced sub network, without change in the resulting performance indices[63]. This will aid the design evaluation of software systems as the model is extended and more information is known about it.

We stated earlier in this subsection that product-form queuing network solution requires the network to satisfy a set of assumptions. Some of these assumptions are related to the assumptions defined by the underlying Markov process representing a queuing network. Examples of such assumptions are:

- *Service centre flow balance*: This implies that the number of arrival jobs is equal to the number of departure (finished) jobs in the observed time period.
- *One step behaviour*: Only a single customer may arrive or depart from a service centre.

One of the main assumptions for a product form queuing network is quasi-reversibility. Quasi-reversibility of a service centre implies that the current state, and past departures and future arrivals, are independent[62]. The quasi-reversibility property was distinguished in [64] as a set of assumptions which are as follows:

- *Routing homogeneity*: This means that the routing patterns for different job classes between service centres, does not depend on the state of a queuing network.
- *Device homogeneity*: The service rate for a specific class of jobs depends only on the number of jobs and classes in this service centre.
- *Homogenous external arrival times*: This implies that arrival rates for new jobs do not depend on the status of the system.

The solution of the product form queuing networks progressed in several stages and in each stage new distributions and disciplines were added. The solution of product form queuing networks depends on providing normalised equations that will solve queuing networks of a specific type and discipline. At first Jackson[65] introduced a solution of exponential, open queuing networks. His solution was based on the *Burke's theorem*[66] which implies that each service centre in a chain of exponential Poisson driven service centres can be analysed independently according to the following equation:

$$p(k_1, k_2, \dots, k_n) = p_1(k_1) p_2(k_2) \dots p_n(k_n)$$

where $p(k_1, k_2, \dots, k_n)$ is the probability of finding k_1 jobs in service centre 1 and k_n jobs at service centre n and $p_i(k)$ is the solution of the corresponding service centre. Gordon and Newell[67] generalised Jackson's solution by including closed queuing networks where the job's arrival is not defined by a Poisson process, but as a fixed population. The product form equation that they produced was:

$$p(k_1, k_2, \dots, k_N) = \frac{1}{G(K)} \prod_{i=1}^N \frac{x_i^{k_i}}{\beta_i(k_i)}$$

Where $G(K)$ is a *normalisation constant* given by:

$$G(K) = \sum \prod_{i=1}^N \frac{x_i^{k_i}}{\beta_i(k_i)}$$

Normalisation constants can also be calculated using the *convolution* method which was provided by Buzen[68]. The convolution algorithm provides a method to derive average performance indices from model solution and the normalisation constants. This algorithm has a polynomial calculation complexity in terms of queuing network

number of centres and jobs. The BCMP[69] solution integrated several early results in a single framework for queuing networks with:

- Multiple customer classes
- Different queuing disciplines (FIFO, LIFO)
- Open, closed and mixed queuing networks
- Fixed probability job class change
- Different service time distributions

Currently most of the product-form queuing networks evaluation packages provide BCMP solutions or extensions of that solution. Examples of these packages are RESQ[70], QNAP2[71] and HIT[72].

Mean-value Analysis

The mean value analysis[73] provides an alternative approach to extracting performance indices from product-form queuing networks models. MVA algorithm provides an approach for calculating the *mean* values for the main performance indices, avoiding direct evaluation of the normalisation constants. MVA algorithm provides a basis for the approximation algorithms used to solve large product form QN and non-product form queuing networks. This algorithm gets its popularity from its dependence on operational laws basis. MVA provides an operational (non-stochastic) analysis where the variables defining service centres and queuing networks are exact measurements rather than stochastic variables. This means that the treatment of these variables will be exact rather than probabilistic. We talked earlier about the operational laws in 3.1.2 but here we will discuss them in more detail. Operational laws were originally described by Buzen [74] and later extended by Denning and Buzen [75]. If we consider a queuing network with N service centres, if we observe this model for a finite time T, and calculated the performance characteristics of each service centre *i* and found that the number of arrival jobs is A_i and the number of completed jobs is C_i , and at time T the device *i* was busy for B_i time. The basic operational law for calculating performance indices for service centre *i* is as follows:

Arrival rate	$\lambda_i = \frac{A_i}{T}$
Throughput	$X_i = \frac{C_i}{T}$
Utilisation	$U_i = \frac{B_i}{T}$

Mean service time	$S_i = \frac{B_i}{C_i}$
-------------------	-------------------------

If we calculate the total number of jobs completed in the system to be C , we can calculate the visit ratio V_i for each service centre, and the throughput of the queuing network X with the following equations:

$$V_i = \frac{C_i}{C}$$

$$X = \frac{C}{T}$$

From the relations above we can prove the *utilisation law*:

$$U_i = X_i S$$

When the number of incoming jobs is equal to the number of completed jobs, the device is said to be *flow balanced*, which is a requirement for PFQN. From this, the *force flow law* is concluded as follows:

$$X_i = X V_i$$

Little's law is one of most fundamental laws in calculating results of a queuing network. It defines the relation between queue length Q and the *resident time* R (time spent on a job in a service centre). Little's law states that the average number of jobs in a service centre is equal to the average resident time, multiplied by the jobs arrival rate. And by considering flow balance devices, the formula for Little's law will be:

$$Q_i = X_i R_i$$

Little's law can be applied, not only to a single service centre, but also to the whole network (as the network satisfies the flow balance requirement). The general formula for Little's law for a queuing network is:

$$Q = X R$$

Q is the total number of jobs residing in the network, and can be calculated by adding the jobs residing in each service centre representing the network:

$$Q = \sum_{i=1}^N Q_i$$

From Little's law we can conclude that:

$$X R = \sum_{i=1}^N X_i R_i$$

If we divide the equation by the throughput X and use the force flow law equation, we will have the *general response time law*:

$$R = \sum_{i=1}^N V_i R_i$$

The operational laws previously stated are sufficient for open queuing networks. This is due to the fact that, when applying the flow balance assumption to the whole system, these equations will provide a steady state for the system under study[62]. This is not true for closed and mixed queuing networks. This is due to a circular dependency in such networks between the throughput for customer classes and the service centre queue length. A solution for this problem was suggested in [73] where the *expected queue length* notation was introduced. The expected queue length for a specific job class arriving in a service centre is equal to the average queue length of that service centre, after removing one job of the same class from the system. MVA solution is used in the JMT queuing network solver and simulator, which is the one used in the tool UML-JMT, which implements the UML-EQN methodology discussed in this thesis.

Extended Queuing Networks

Product-form queuing networks provide a balanced trade-off between accuracy extracted performance indices and complexity of evaluating the performance model. The main drawback of product form queuing networks arises from the restricted class of queuing networks it represents. An accurate representation of the properties of a large spectrum of computer systems could not be gained due to these restrictions. *Extended queuing network* can be defined as a generalised product form queuing network with added properties. Some of the generalised properties in EQN include:

- *Extended queuing scheduling*: A new queuing scheduling discipline was allowed (i.e. priority). This was necessary to model systems, allowing these types of scheduling discipline.
- *Marking Jobs*: This means adding information about the job, such as the message length in networks modelling communication networks.
- *Network status routing*: New options for the routing procedure were added. These routing procedures depend on the status of the queuing network. An example of such routing procedure is *load dependent routing* where a job is routed to the (longest, shortest or fastest) queue.
- *Jobs holding multiple resources*: The main drawback of conventional product form queuing networks is its limitation in modelling jobs, in that it is capable of holding multiple resources, although this is a common activity in computer systems. EQN allows modelling such activity by introducing *passive* and *active* queues. Active queues are the conventional queues. A

job is allowed in EQN to hold resources in several passive queues and a single active queue.

- *Extended arrival and service process distributions*: Extended variety of distributions are added to represent the arrival and service process distribution.
- *Concurrency*: A new notation that was added to EQN is *fork/join service centres*. Fork/join notation is used to model jobs served in parallel. This is usually used in modelling parallel and grid systems. A fork station receiving a job will split this job into a number of identical jobs scattered to the service centres, connected by the fork/join stations. The join station works as a synchronization centre to collect all the completed jobs.

The extensions added to product form networks to reach EQN, contradicts with most of the assumptions made by the solution algorithms for product-form queuing networks. This will make a generalisation of the product form analytical solution include all EQN properties, unfeasible. Currently most of the modern queuing network evaluation tools provide analytical functionalities for solving product-form queuing networks and simulation tools for providing solutions tools for non-product form queuing networks. With the massive computation power of current computer systems, simulation is no longer causing a problem of computational resource requirements.

3.4 Other Performance Evaluation Techniques

Even though Analytical modelling and simulation can be described as being two of the key performance evaluation paradigms, there are other practical performance evaluation techniques. In this section, we will briefly discuss some of these techniques. First we will discuss a formal modelling paradigm known as process algebra modelling and its extensions. Then we will discuss an alternative approach to a performance study which is workload analysis.

3.4.1 Process Algebra

A *process algebra* model is a semantic model describing the behaviour of a system. Process algebra started as an aid to study the behaviour of concurrent systems. Originally, process algebra was found to offer algebraic means for the verification of the functional characteristics of a system. To study performance measures of software systems, time information has to be added to the behaviour model presented by process algebra. Two types of time annotated process algebra emerged; *stochastic process*

algebras, used to describe and investigate the behaviour of resource-sharing systems, and *timed process algebras* for real-time systems[76]. As this thesis is concentrating on software systems, which are classified as resource sharing systems, we will only discuss stochastic process algebra (SPA) in this section. The time information is added in the form of random variables representing the time in which they occur and duration of each activity. The evaluation of the semantic model represented by SPA provides means for the investigation of both functional, and non-functional aspects of the system. The functional aspects include (as we recall) functionalities presented by the system and the absence of deadlocks. The non-functional aspects include performance, reliability and availability[76]. SPA models are evaluated by solving an underlying continuous time Markov chain that can be driven from the semantic of the SPA.

The idea of SPA originated from the original time annotated process algebra, where time segment was associated with actions to define time duration before this action occurs. The need was to represent this time as stochastic. The first SPA extension dedicated for studying the performance characteristics, was TIPP[77]. Another extension of the SPA for performance was PEPA [78]. These extensions of SPA (and others) provided languages used to represent the semantic representation of the modelled system and the associated stochastic variables. Most of these extensions provided tools that will transform the semantic form to the equivalent Markov chain which will be evaluated (usually by simulation[76]) to deliver the performance results of the model. Examples of these tools are TIPPTool[79] and PEPA workbench[80].

Process algebra offers attractive features which gives it the ability to represent both structural and behavioural aspects of the modelled system. A PEPA model extends traditional process algebra by associating the actions with a random variable that represents that duration of that action which is assumed to be exponentially distributed. PEPA models are a formal description of the components composing a system and the behavioural interactions between these components. The interaction is described as a series of actions represented as a pair (α, r) where α is the action type and r is the parameter of the exponential distribution governing the duration of that action. The structure and behaviour of the system is demonstrated by demonstrating relation between the components and the behaviour of each component using a set of *combinators* defined in the PEPA syntax. An example of a PEPA model representing a web based system has two components: a *Browser* and a *Server*. A *Browser* either

display data from its cache or retrieve it from the *Server*, in which it will have to *send* a request, *process* the request and *download* the result. The PEPA model for that system is as following:

$$\begin{aligned} \text{Server} &::= (\text{send}, T).(\text{process}, \mu).(\text{download}, T).\text{Server} \\ \text{Browser} &::= (\text{display}, p_1\alpha)(\text{cache}, m). \text{Browser} + (\text{display}, p_2\alpha). (\text{send}, T).(\\ &\text{process}, \mu).(\text{download}, T). \text{Browser} \\ \text{WebSystem} &::= \text{Browser} \langle \{ \text{send}, \text{process}, \text{download} \} \rangle \text{Server} \end{aligned}$$

As the *Server* require a request sent by an acquirer, the action for the send is distinguished by a *Top* duration, which means that the rate of the action is outside the control of this action. The behaviour of the Server component is demonstrated by a sequence of actions in which the request will be sent, processed, downloaded then the server will be released. This is distinguished by the *prefix(.)* combinator. The Browse component has two options, either to display content from the cache, or to obtain this content from the server according to the probability p_1 and p_2 . The *choice (+)* combinator defines the different scenarios a component may have. The web system is composed by the cooperation of the two components Server and Browser, this is indicated by the *cooperation* combinator which include the *cooperation set* that contain the action types involved in that cooperation.

Stochastic process algebras provide a formal method for investigating functional and non-functional aspects of a software system. As we recall from the previous chapter, the process of validating functional and non-functional requirement are placed in separate stages. This comes back to the lack of a methodology that could include both validations in a single study. SPA provides a means to complete such a task and although the SPA extensions discussed above concentrate on performance, the research area is still relatively new. An SPA model could be the input for a CASE tool where verification and code generation are part of the automated tasks. On the down side, formal representation and lack of visualisation will make the *modelling* and understanding of these models much harder.

3.4.2 Workload Modelling

In computer systems, users generate requests in the form of commands, data, invocations ... etc; these inputs are collectively called *workloads*. Workload modelling involves studying a system's performance by analysing the *real* or *synthetic* workload characteristics of the system. From the definition, we can ascertain that workload

modelling involves two main procedures named *workload characterisation* and *workload analysis*. Workload characterisation involves collecting workload measures of the system under study according to the specification of that study. Workload analysis depends on the raw data collected during the previous phase into meaningful performance indices, and load distributions that will aid the decisions made on the system status. Workload modelling is usually a task involved in evaluating performance characteristics of existing systems. Such studies are essential for system tuning, to check the best alteration and the effect of this alteration. Workload modelling is also beneficial in component based systems to verify the compatibility between components by means of *benchmark* studies.

There are two types of workload characteristics (real and synthetic), which can be distinguished by where and how these characteristics are obtained. Real workload characteristics are taken from *live* runs of a system by logging performance measurements required by the performance study. The kinds of measurements obtained from such runs are neither controlled, nor repeatable. Furthermore, it does not usually cover the whole system's functionalities, which weakens their role in the process of gaining an overall performance study that covers all aspects of the system. The importance of real workload characterisation arises from the information they provide regarding frequency of usage, which is an essential performance measurement used in any performance study. Synthetic workload characterisation is obtained by conducting controlled and parameterised experiments using test or real data, which are called benchmark tests. Benchmark tests provide a means to gather measurements of the system that covers all possible behaviour and load scenarios. The main drawback of benchmarking is the lack of realism and the overhead costs.

Workload analysis involves using the measures collected from the system to obtain a clear view about the system that will assist in decision making. This involves selecting the components and parameters on which the study will be built and normalising the collected data to calculate performance indices. Workload analysis can be done by manually studying the logs of the experiments, or by using workload analysis tools. An example of a workload analysis tools is the JWAT, which is one of the tools provided by the JMT suite. This tool provides a means for analysing log files containing characteristics of resource utilisation or traffic requests, using a set of known statistical techniques (i.e. k-Means)[81]. As we stated earlier, workload modelling provides easy

and flexible methods to achieve relatively realistic measures of software systems performance. There is, however, one condition which is that the systems have to *exist*.

3.5 Performance Modelling for System Design

As the title of this thesis suggests, we are concerned with performance evaluation in the design phase of a system's life cycle. We require a performance evaluation methodology that will provide a means to easily and flexibly study and characterise systems performance measurements with limited information about the system. At the design phase, the type of performance study usually conducted falls into one of these categories:

- *Choosing the "best" design:* At the design stage, there are usually multiple design alternatives. Choosing among these designs depends on different measures. One of the important criteria for selecting a design is performance. The design with the *best* performance readings will be selected, in accordance with other aspects (e.g. cost).
- *Validating a Design:* As we clarified in the previous chapter, the validation of a design against non-functional requirements was a task left until the testing phase, and this can lead to catastrophic problems in software projects. Performance non-functional requirements can be verified against the suggested design in the design phase, where errors are still easy and cheap to fix. This can be done by conducting a performance study with the objective of comparing the required performance non-functional requirements to the actual measures of the suggested design.

The challenge in studying performance in early stages of system life arises from the limited amount of performance related data available at that stage. The performance data usually ranges from previously measured performance characteristics of hardware/software (off the shelf) components, which can be gained from the components specification document, to estimates for the expected workload for the different functionalities of the system. This can be calculated on the basis of predictions or historical data gained from previous or similar systems.

In the previous sections, we have introduced the main trends of performance evaluation technologies. Each of these paradigms has its strengths and weaknesses and the domain in which they become the best available technology. The requirements for a

performance evaluation technology to conduct a performance study in the design phase can be summarised as the following:

- Can report performance description of the system to an acceptable degree of accuracy with respect to cost, with minimal information about the system
- Reflect both the structural and behavioural aspects of the system
- Simple enough for modelling, inspecting and solving
- Cost efficient

We can directly exclude the workload analysis technique from our candidates list. This is due to the fact that this type of performance evaluation practice requires the system to exist. Simulation, on the other hand, can be used to evaluate the system performance at an early stage. Simulation programs can be built to a degree of abstraction related to the amount of information available about the system. The problem arises from the cost requirement. The amount of time and programming resources required to conduct such a study exceeds the potentials of that study.

From the previous requirement, it is obvious that the best performance evaluation paradigm is either analytical (Petri-nets, Queuing networks) or formal modelling. We noticed earlier that all of these performance evaluation paradigms provide cost efficient, flexible, and acceptably accurate performance evaluation of a system. Queuing networks models reflect both the structural and behavioural aspects of the system. The strength of the queuing network arises from the structure oriented nature of this modelling paradigm. On the other hand, Petri-nets and process algebra tend to be more behaviour oriented. For the sake of our requirements, from a design point of view, structural aspects are more salient, making queuing network more suitable for the scope of our work. Cortellessa *et.al.*[82] evaluated these three performance modelling paradigms from the perspective of the software designer. In an experiment using product-form queuing network, GSPN and TIPP process algebra to model an XML translator, the study objective was to find the most acceptable paradigm for software design. The study focused on two main dimensions which are relevant in the design level, which are the *adequacy* (use the paradigm to conduct a performance study at the design level), and *ease* of conducting a comparative experiment using this paradigm. The conclusion of the study stated that although product form queuing networks lacked representation of behavioural aspects, QN seemed to behave better with respect to the adequacy and easiness dimensions. We chose to adopt EQN in this thesis, as it

compensates for the drawbacks of limited representation of behaviour with the different extensions discussed earlier.

3.6 Performance Model Evaluation Tools

As we saw in the previous sections, the process of evaluating a performance model to derive performance indices requires intense mathematical and statistical background, along with deep knowledge of the modelling paradigm itself. This encouraged the development of automated tools for the solving and evaluation of performance models. These tools ranged from single tools to evaluate a particular modelling paradigm, to sophisticated capacity management and planning environments. In the earlier versions of these tools, a model evaluation algorithm would be implemented, which can work only on a single class of models. These tools were used primarily in research areas. An example of such tools is the PEPS Markov model solver[83]. This solver was dedicated to solving complex Markov models. Later, more advanced tools were developed that provided a complete solution for a specific modelling paradigm. These tools accompanied multiple algorithms for solving different classes of a specific modelling paradigm, as well as simulation of the model fall-out of the scope of the solving algorithms. An example of such tools is QNAP2[71]. QNAP2 tools are a queuing network evaluation tool that provides the user with multiple options for solving the network analytically (e.g. conventional, MVA, ITERATIV..., depending on the class of queuing network) or by simulation. QNAP2 requires the user to represent the queuing network in a PASCAL-like language. The notations written in the queuing network description code, as well as the execution code, will instruct the solver with the type of solution and which queuing network solving algorithm. Another example of such tools is PEPA workbench[80], which provides solutions to the PEPA formal stochastic algebra.

Currently most commercial performance evaluation tools come as a part of capacity management and planning environments. These environments provide the user with a wide range of performance evaluation techniques which include modelling, simulation and workload analysis. These environments usually incorporate multiple performance evaluation tools. An example, the BEST/1 queuing network tool (which was one of the first commercial queuing networks packages) is now a part of the BMC capacity management environment. Some of these environments provide multiple performance evaluation paradigms that can model the system with different abstraction notation and at different levels of hierarchy depending on decisions taken from the specification of

the system and the performance study objectives. Examples of these tools are the IMSE[84] and the HIT[72] environment. The IMSE provides functionalities to study a system performance by multiple performance evaluation paradigms. Modelling is available through both queuing networks and Petri-nets. In IMSE, the performance indices of the system under study is predicted by experimenter tools that work on a special model of the system called PrM notation.

Table 3.1: Comparison between some of the performance evaluation tools according to generality, simplicity and extendibility criteria.

		PEPS	QNAP2	PEPA	BMC	IMSE	HIT	JMT
Extendibility	Ability to extend the tool through a structured interface	0	1	1	1	1	1	1
	The interface is a standard interface and the model generation can be done easily	0	0	0	0	0	0	1
	The ability to provide both model and experimentation data through this interface	0	1	1		0	0	1
Simplicity	The representation of the performance model in a graphical form	1	0	0	1	1	1	1
	The experimentation process is easy, and the results are clearly displayed	1	1	1	1	1	1	1
	Alterations to the performance model on the tool are easy.	0	1	1		0	0	1
Generality	Support multiple performance models and performance evaluation paradigms	0	1	0	1	1	1	0
	Support multiple performance evaluation paradigms	0	0	0	1	1	0	1
	Support EQN	0	1	0	1	1	1	1

In this thesis, we were looking for a queuing network evaluation tool which we will use as the basis for the tool implementing the UML-EQN methodology. With the variety of performance evaluation tools available, we have to set some criteria for choosing the best one for our needs. The first criterion was *extendibility* which means that the tools need to be able to interact with our tool. This interaction can be by embedding any of the systems in the other or by allowing a common interaction language between the tools. Another decisive criterion is *simplicity*, which covers both the use of the tool for

modelling and visualising the performance indices. One of the main simplicity rules was that the model needs to be defined in the same way that the methodology specifies. As an example, in queuing networks, the network model needs to be modelled visually with graphical notations. This will prove the clarity of the model. Using textual notations to represent graphical models will undoubtedly increase the effort the user has to make in order to improve or inspect the model. One key criterion, which we partly considered (as the methodology is for EQN), is *generality* which denotes the coverage of the tool of performance evaluation methodologies. This includes the range of performance study paradigms and the variety of algorithms adopted. Table 3.1 shows a survey composed by the writer of this thesis for comparing a number of performance model evaluation tools according to the criteria discussed above. For each criterion we specified three properties which can be classified as being important in the performance model evaluation tool we are seeking. We gave scores for each tool according to these articulated properties as shown in Table 3.1. Figure 3.4 shows a comparison between the total scores for each of the criteria, the graph clearly shows that the tool most suitable to be the performance evaluation tool is the JMT suite.

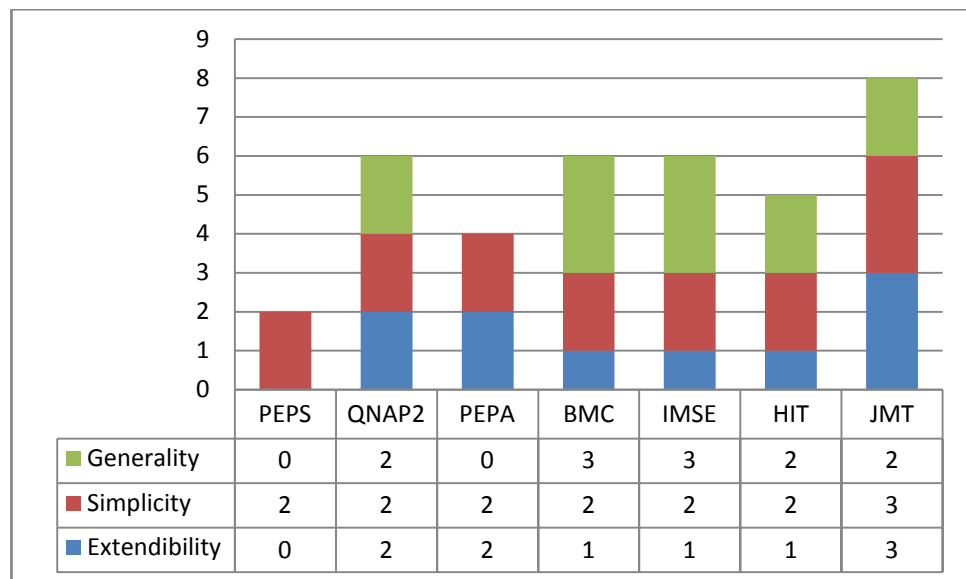


Figure 3.4: Comparison between some of the performance evaluation tools according to generality, simplicity and extendibility.

The queuing network evaluation tool that we are using in this thesis is the JSIMgraph which is a part of the JMT suite[81]. The Java modelling tools suite is a collection of performance evaluation tools that provide modelling and capacity planning, and analysis functionalities. The reason for choosing JMT is that it matched all the criteria we had set for the required model evaluation tool. From a generality point of view, JMT

provides the means to evaluate queuing networks models analytically and by simulation. It also provides workload characterisation and analysis tools which can provide characteristics that can be used as inputs for performance models in the other tools provided by the suite. As for simplicity, JMT provides full graphical modelling and analysis capabilities that will allow the user to build, inspect and update queuing networks models easily. The tools also provide user-friendly analysis tools that will help the user in studying the performance indices generated by the solving tools. Expanding the JMT tool to include performance models created by our UML-JMT tool was possible as the JMT main design goal was extendibility. Next we will discuss the queuing network solution tools and analysis tools available in JMT. In Section 7.2, we will discuss the technical aspects of the JMT suite which allow the extendibility of the tool.

3.6.1 JMT Queuing Network Solution Tools

The JMT suite provides two main methods for solving a queuing network. Queuing networks can be solved analytically or through simulation. The analytical solution provided by the *JMVA* tool provides the exact analysis of product-form queuing networks through a stabilised version of the MVA algorithm[81]. The simulation solution is provided by a discrete event simulator for the analysis of queuing networks called *JSIM*[85]. The *JSIM* supports several probability distributions for characterising service and inter-arrival times, as well as different routing strategies[86]. JMT suite provides simulation solution through two tools, the *JSIMwiz*, which is a wizard interface for the *JSIM* simulator, and the *JSIMgraph*. The *JSIMgraph* is a graphical user interface tool that allows the user to design and amend a queuing network model as a workbench. As the model generated by the UML-EQN methodology may include non-product-form aspects (i.e. fork and join), we chose to use the *JSIM* simulator as the main queuing network solver. The model produced by the UML-JMT tool is configured to be opened by the *JSIMgraph*, where the user can amend the model and conduct the performance analysis experiment. A great feature of the *JSIMgraph* tool is that it can open a model designed inside it in the *JMVA* tool, provided that this queuing network is a product-form queuing network. This means that if the model produced by the UML-JMT tool does not have any non-product-form aspects, this model can be solved either analytically or by simulation.

3.6.2 JMT Queuing Network Analysis Tools

The JMT queuing network analysis tools provide a set of analysis functionality that will help the user of the tool to study the performance indices of the system. These functionalities can be working on a fixed set of input parameters or on a variable control parameter. An analysis tool available in the JMT suite called the *What-if* analysis tool, allows the user to set one or more control parameters (can be the number of users, workload ... etc) ,and this tool will evaluate the performance model for the performance indices that the user selected along the ranges selected for this control variable. This will allow the user to observe the change in the system behaviour as the conditions around the system change. The JMT suite provides different performance indices for the user, such as throughput, utilisation and respond time. These can be for the whole system or for a specific work station or job class. Other performance indices that describe the performance of specific stations include, queue length, queue time, residence time, response time and utilisation. The reader can refer to the JMT suite user manual[87] for more information about the tools analysis functionalities.

3.7 Summary

Performance evaluation is the process of assessing the performance of the software system. The performance of a computer system is defined by performance related measurements called the performance indices. These indices relate to the speed of response, utilisation of resources, and the usage of the system in the context of the organisation. This Chapter provided background information related to software performance evaluation technologies. This included defining the software performance evaluation process and its importance. It also explained the process of software performance evaluation, describing the fundamental terminologies used in the process, and detailing the main techniques used to produce a performance study. The objective of this chapter is to investigate the different performance evaluation paradigms in order to determine the “best” modelling paradigm that would be most appropriate to represent the architectural specifications of a software system. To do this, we provided an in-depth description of some of the key performance evaluation paradigms used in software performance evaluation. The evaluation of these paradigms was in terms of the cost efficiency, the generality of the model, ability to maintain architectural aspects and the cleanness in representing these architectural aspects. At the end of this comparison, we found that the EQN would provide the “best” modelling paradigm. The choice of this modelling paradigm is also related to the availability of analysis tools that would

solve this performance model for the required performance indices. We found that JMT matched all the criteria we had set for the desired model evaluation tool. Therefore, JMT was chosen to be extended in the performance evaluation tool developed in this research.

Integrating Performance Evaluation in Software Engineering

When the software crisis hit in 1960, a great deal of attention was placed on the verification of functional requirements which were considered to be of crucial importance [5]. Most of the methods that were used in verification, such as prototyping, only focused on functional requirements, and even the modelling languages used to model these requirements focused on representing functional specification. Over the last two decades, researchers have addressed the importance of integrating qualitative requirement into the development process. One of the principal qualitative requirements for software is performance. The process of verifying the performance requirements of a software system is one of the tasks defined by *software performance engineering*. Software performance engineering is a means of integrating performance evaluation techniques into software development processes. Performance engineering processes include techniques that will assist software engineers with performance evaluation related tasks, either during the development or maintenance of a software system[88].

Performance evaluation task can be carried out in any phase of the system development life cycle. However, it is rare for a system to be fully designed and functionally tested before any attempt is made to determine its performance characteristics. This is due to the fact that redesign of both hardware and software is costly, especially in the late stages of the development cycle, and may cause delayed system delivery. Also, it is possible that the system in hand cannot be tested by direct experiment for a reason related to its nature (i.e. dangerous, disruptive ...), or because the system does not yet exist[55]. Despite its importance, the modelling process requires highly trained modellers; this is because the modelling process is said to be an art which requires experts who have significant experience in performance modelling terminology. The supplementary budget required to achieve performance evaluation often causes the exclusion of this task from software project plans. This event has inspired researchers to find methodologies that will allow system architects to complete the performance analysis task without any of the additional costs listed above. One methodology

investigated involves the generation of the performance model from the system architecture model (SA), represented in UML. The UML model represents how the system components interact with each other. Using this information plus statistical data about the system and QoS requirements, a performance model can be generated.

This chapter will discuss software performance engineering processes for both conventional and agile software development methodologies, and the performance evaluation techniques used in the performance evaluation tasks defined by software performance engineering. Section 4.1 will discuss the terminology of software performance engineering. As agile development was recently deployed, literature did not report any attempts to define a performance engineering method for system architecture assessment. Therefore, in 4.2 the author has suggested a method for assessing the performance of a projected design in agile development. Section 4.3 will provide a survey of some of the work done in the field of generating performance models from design models.

4.1 Software Performance Engineering

Software Performance Engineering refers to the performance related analysis and activities incorporated in the software engineering process[88]. The importance of software performance engineering arises from the need for methods that will provide assurance of the quality of software systems during development and maintenance. This is essential as the size and complexity of modern software systems are continually increasing. As Smith *et al.* explained, the earlier the performance validation process is undertaken, the more confident we are of finding any design faults that may affect the quality of the final software product[8]. In general, performance engineering tasks are incorporated into the development's design and test phases as a means of ensuring that the suggested design meets the QoS requirements. Moreover, performance engineering can be used in the maintenance phase for identifying and solving possible performance related problems. This thesis is concerned with the first role of software performance engineering, that is, the performance non-functional requirements verification role during the development of software systems.

Literature reports a number of performance engineering methodologies used to verify the software architecture against performance non-functional requirements. Work on this includes The Software Architecture Analysis Method (SAAM)[89], the Architecture Trade-off Analysis Method(ATAM)[90] and Performance Assessment of

Software Architecture (PASA) [13]. All of these methods provide an approach for evaluating the performance of software architecture. These approaches do not only concentrate on the generation and analysis of the performance model, but also on the methods used in gathering the performance related information, assessing the system from a performance perspective and suggesting methods for solving any performance problems. This defines the main difference between software performance engineering methodologies and *performance evaluation* methodologies. We can identify the performance evaluation methodologies as the methodologies mainly concerned with the generation of the performance model used to evaluate a system performance. These methodologies can be deployed in the performance evaluation step of a performance engineering methodology. We will discuss some of these methodologies in Section 4.3. PASA is one of the complete performance engineering methodologies dedicated to information systems performance assessment. It provides a method for assessing and solving performance problems related to software architecture, with respect to technical and economic aspects. PASA will be discussed in more detail in the next sub-section.

As we explained earlier in the previous two chapters, the use of performance evaluation in non-functional requirement verification involves conducting a performance study on the architectural design of the projected system, by abstracting the critical behaviour that is expected to affect the performance. This performance study will evaluate the performance capabilities of the suggested design with respect to expected workloads. Despite its importance in the software development process, it is generally acknowledged that the lack of performance engineering deployment is mainly due to the knowledge gap between software engineers/architects and performance engineering experts, rather than to fundamental issues. In addition, most of the well known performance evaluation processes require an extra budget to fulfil the performance evaluation task. This budget will be invested in hiring professional system modellers or in programming simulation models for the system. This overhead in financial and time resources can cause the exclusion of this task from the software project plans. This has inspired researchers to find comprehensible, cost efficient technologies that will allow system architects to perform the performance analysis task without any of the extra costs listed above. One approach, which has been investigated widely, is to use the system architectural and behavioural characteristics represented in software architecture modelling language (e.g. UML) as the source to generate an equivalent performance model for a system. These methodologies utilise the structural and behavioural aspects

of the system represented in different notations, in addition to expected workload characterisation of the projected system, to generate a performance evaluation model that can be solved or simulated to assess the expected QoS specifications of an architectural design.

4.1.1 PASA

The PASA method is a performance evaluation process for software architecture. It was introduced by Williams and Smith as the nectar of their experience in software performance engineering. The idea behind the introduction of PASA is stated in [13] “Our experience is that performance problems are most often due to inappropriate architecture choices rather than inefficient coding”. PASA provides techniques and strategies for identifying and solving potential performance risks in suggested architectural designs. PASA is scenario based, which means that the performance analysis of the software architecture will concentrate on a set of critical scenarios with respect to performance. The criticality of these scenarios arises from the workload characterisation or service demand expected in these scenarios.

The PASA method is specified to be deployed in the design and test phases of a conventional development process. It starts with the suggested architectural design in hand and a set of potential key scenarios with large workloads or service demands. The next step is to identify the objective quantitative performance acceptance measures. These are usually gained from the non-functional requirements specification. The next step includes building a performance model from the architectural design and the critical scenarios. This model will be solved and analysed to gain the expected performance indices of the suggested architectural design. As part of the analysis, the calculated performance indices will be compared to the expected performance characterisation to evaluate the performance of the architectural design. If any potential risks arise from the results of the analysis, PASA suggested three main strategies for eliminating these risks. These structures include deviation from the architectural style, alternative interaction between components, and refactoring. The last step of the PASA method includes economically analysing the suggested architecture. This analysis includes studying the cost/benefits trade-offs of the suggested architectures and the potential alterations on that architecture.

Williams and Smith summarise the activities of the PASA methodology in ten steps quoted from [13] which are:

1. *Process Overview*: The first step of the assessment involves an orientation process for the project staff on the importance and steps of the performance study.
2. *Architecture Overview*: The suggested architecture(s) are presented by the system architects.
3. *Identification of Critical Use-Cases*: The most important functionalities expected to affect the system performance are identified.
4. *Selection of Key Performance Scenarios*: The scenarios of the critical use-cases are identified.
5. *Identification of Performance Objectives*: The intercepted QoS measurements are identified.
6. *Architecture clarification and discussion*: Participants conduct a more detailed discussion of the architecture and the specific features that support the key performance scenarios. Problem areas are explored in more depth.
7. *Architectural Analysis*: The architecture is analysed to determine whether it will support the performance objectives.
8. *Identification of Alternatives*: If a problem is found, alternatives for meeting performance objectives are identified.
9. *Presentation of Results*: Results and recommendations are presented to managers and developers.
10. *Economic Analysis*: The costs and benefits of the study and the resulting improvements.

The PASA method suggests the use of the SPE [6](Software Performance Engineering) methodology and the SPEED tool [91] for performance evaluation tasks identified in the PASA method. We will discuss them further in the next subsection.

4.1.2 SPE Methodology

One of the first complete methodologies to integrate performance analysis into the software development process, was the SPE (*Software Performance Engineering*) methodology by Williams and Smith [6; 92]. The SPE adopted a new trend in system performance evaluation by considering both the architecture and behaviour of a system rather than considering only one of them. Prior to the SPE methodology, the system performance was measured by evaluating either the hardware configuration or the

behaviour of this system by a suitable modelling paradigm. SPE introduced the combination of these two fundamental aspects in two Meta models named *Software* model and *Machine* model. The software model is represented in SPE by a notation for modelling the behaviour and resource usage called the *Execution Graph* (EG). EG is a graphical representation of the functional and resource demand characterisation of a system. The machine model is represented in SPE as a QN, as discussed earlier in Chapter 3. The separation of the structural and behavioural characterisation will provide the modeller with flexibility which will allow him/her to experiment with different hardware/software configurations.

The SPE uses a top-down approach in the specification of the EG. The EG is a directed graph with nodes representing the functional components composing the system. These components can be detailed or abstract, depending on the stage of the performance study and criticality of the role of this component in the overall functionality of the system. The SPE adopts the 80/20 rule, which states that 20% of the total functionalities of a software system will determine 80% of the performance. Therefore, when initially constructing the EG, only the most common functionalities will be explained in detail. Between the nodes of EG are arcs that define the execution paths in an EG. These arcs are parameterised with probabilities depending on the frequency of the execution path they represent. The EG is also annotated with the attributes representing the resource requirement for each of the functionalities' execution paths. These annotations are called *demand vectors*. The analysis of the EG provides information about the performance of each of the execution paths. This is done in the *basic analysis* of the EG, where the EG is analysed in a bottom-up manner. In the basic analysis, the best, worst and expected delays are calculated for each of the execution paths. This is done starting from the leaf nodes and continuing upward. The information gained from this analysis, along with the hardware and software components model represented by the machine model, are added to a system execution model which will provide performance measures for the projected system. The SPE methodology was automated by SPE.ED [91], a performance modelling tool specifically designed to support the SPE methodology. In SPE.ED the user must identify the scenarios he/she wants to inspect, in terms of a modelling language that represents the EG. SPE.ED will then construct an EQN for the system; this EQN will be solved to provide the requested performance measures.

Although SPE provides simple, effective and cost efficient methodology for software performance evaluation, its dependability on non-standard software behaviour modelling notations (EG) is one of its drawbacks. This was solved in [13] and [93] where UML behaviour models were suggested as a starting point for building the EG. These methods provided algorithms for using the system's UML sequence diagrams to construct the EG and its UML deployment, and class diagrams to create the EQN of the system. This method of extracting the performance model from the software's UML architectural and behaviour model is the latest trend in software performance engineering. Literature reports a number of methodologies for transforming specific UML diagrams to different types of performance models [2-6]. Although these methodologies can help in capturing the performance aspects of the designs that they represent, the simplicity of these methodologies and the degree of automation of the performance evaluation test provided by these methodologies will affect the ability to merge these methodologies in the non-functional requirements verification task in any of the software development processes. These methodologies will be discussed further in Section 4.3.

4.2 Performance Engineering in Agile Development

As previously discussed in Chapter two, changes in the overall environment of the IT and business worlds require different techniques that can adapt to the current requirements of the business. Requirements such as increasing the business value of developed software and decreasing the costs caused by developed, unused functionalities, inspired developers to adopt alternative routes in software development project management. The terms agile and incremental became keywords when talking about software development techniques. These techniques are based on iterative development, where requirements, designs and developed programmes evolve continually. These paradigms depend on continuous automated testing for the purpose of verifying the implementation of the current release against the current set of requirements. At present, the majority of literature discussing the role of requirements engineering in agile development processes [29; 94; 95] seems to indicate that non-functional requirements verification is an uncharted territory. This was originally true for conventional development methodologies, such as waterfall or RUP, until frameworks were introduced to incorporate performance assessment as part of the development process.

As we discussed in the previous section, PASA is a framework for studying the performance aspects of a software design. PASA concentrates on systems developed in a conventional software engineering approach, where a full and finalised requirements specification is ready for the designers at the start of the design phase. This is not true in agile development processes, where requirements evolve and can only be finalised during an iteration of the development of a component where its requirements are specified. In response to this, CPASA (Continuous Performance Assessment of Software Architecture) is an extension to the PASA framework, which is adjusted to work in software projects where the performance can be affected by changes in the requirements. CPASA is, in fact, designed with agile and incremental software development processes in mind. It was developed on the basis that, since continuous change in the requirements will eventually have an effect on design, checks should be made in each cycle to ensure that the performance characteristics of this design are not adversely affected. To achieve this, the CPASA framework matches the main practices of agile modelling and development. This section will discuss the CPASA formwork. We will first discuss the extension of the original PASA framework (CPASA) to allow it to be deployed in an agile development methodology, and then we will discuss the deployment of the CPASA in the development, using the performance evaluation tool developed by the writer of this thesis, which is discussed in detail in Chapter 7.

4.2.1 CPASA

As previously mentioned, CPASA is an extension of the PASA method. The motivation behind this extension was to customise the steps of the PASA method to be deployed in the expansively adopted agile development process. PASA was designed on the basis of the conventional development processes, where the full set of requirement and design specifications are decided in the early stages of the development cycle. PASA adopt a method for assessing the performance characteristics of the architecture design. The cost of maintaining performance problems arising from this stage is undoubtedly lower than if these problems are found later. As discussed in Chapter 2, in agile development, the requirements specification for a component is only available at the time of development of this component. In several cases, these requirements will cause a change in the instant architectural design of the overall system. Since continuous change in the requirements will eventually have an effect on design, checks should be undertaken in each cycle to ensure that the performance characteristics of this design are not adversely affected.

The CPASA method was designed, from the outset, to be integrated into an agile development process without affecting the overall agility of the development process. The requirements of the CPASA method to evaluate the performance aspects of a system's architecture are all utilised from the information and artifices generated and used in any agile development process. The philosophy of the agile development methodology can be summarised in the following points:

- Continuous requirements elicitation and design
- Continuous test driven implementation
- Automated testing
- Continuous integration
- Continuous feedback

The CPASA method was designed to maintain these points in the development process while providing the performance verification required during system development. In this section, we will discuss the effect of the CPASA method of performance assessment on the agile development process, by discussing the deployment of the CPASA method in the development of an agile project.

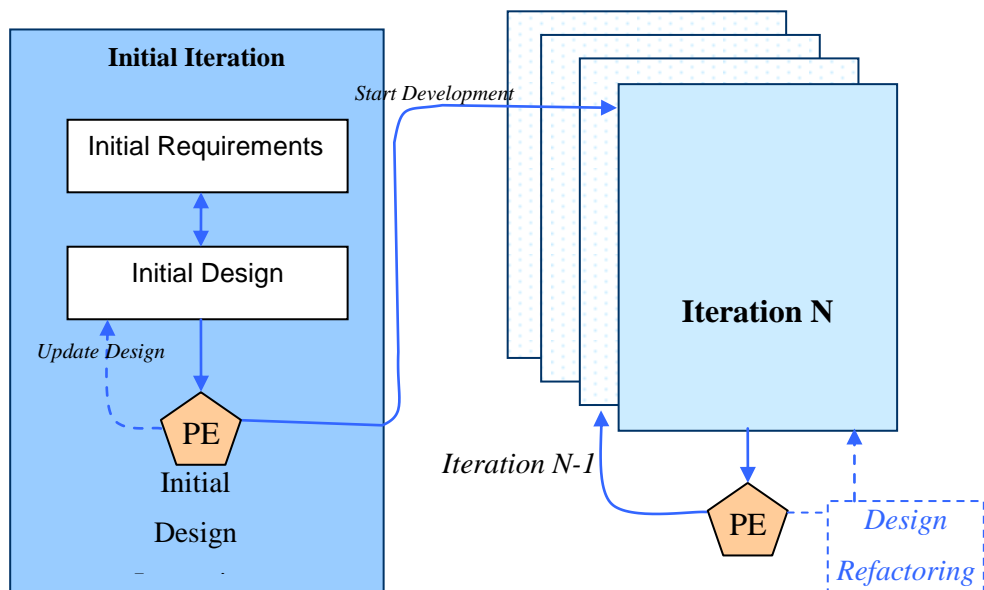


Figure 4.1: Outline of the CPASA method in agile development process.

The main extension of the PASA methodology suggested by CPASA can be summarised by extending the scope of deployment of the PASA method from simply the design phase, to the whole development process. This comes back to the fact that the agile development process relies on continuous requirement/design iterations. Another alteration of the original PASA method is by concentrating and minimising the

performance assessment steps to allow the CPASA to be included in the short/fast iterations of the agile development process.

The CPASA method consists of continuous PE (Performance Evaluation) tests (shown in Figure 3 as pentagon shapes). Each PE consists of the following steps:

- *Construct Performance Model*: As in PASA, the architectural design and the key scenarios will be used to automatically construct a performance model. This model will be used to study the expected performance capabilities of the instant design. The scenarios used to construct this model depend on the stage in which the model is built (which iteration). This will depend on the amount of information about the scenario gained by the stakeholder.
- *Solving/Analysis of the Performance Model*: The solving and analysis of the performance model built in the previous step should be automated to implement the automatic-continuous-testing tenet of agile development. There are several ways for implementing similar automatic tests as we saw in Section 2. In the next section, we will discuss the UML-JMT tool which was developed to provide an automatic tool for verifying performance of non-functional requirements during agile development.
- *Tuning or Refactoring (if needed)*: If potential risks arise, the agile development process adopts refactoring as a procedure for correcting any problems in relation to architecture. We propose two types of changes in the architecture in regard to performance. If the performance can be solved by simple tuning on the components configurations (number of service threads), we call it a *minor* refactor. If a full refactor is needed, we call it a major refactor.

We will further explain the PE steps in the next sub-section where we will discuss the PE test using the UML-JMT tool developed to deploy the CPASA method.

The main outline of the CPASA method deployment in an agile development process is shown in Figure 4.1. The initial architectural design of the projected system is verified in the initial iteration, where the system requirements and structure are envisioned. In this stage, the main components of the system are distinguished and the structure in which these components interact is specified. Furthermore, the behaviour in which these components interact is outlined. These components will be developed during each iteration. The *initial design inspection* is concerned with assessing the performance aspects of the initial architectural design. This will help in validating the suggested

design and comparing different design alternatives. The process of comparing these alternatives will be based on both the technical and economical aspects of the suggested architectures. As discussed in Section 3, by the end of the initial iteration, the initial architecture and initial plan will be determined. During the implementation iterations, more information about the implemented component will be released. This returns to detailed requirements gained during each of the iterations. These requirements will include new performance characteristics and new potential critical scenarios. This could affect overall performance on the instant architecture. These emerging requirements will potentially have an impact on the design; therefore, a re-inspection study of the performance capabilities of the instant architecture is required. After each iteration, a performance study is conducted to *re-inspect the design*. If the performance was adversely affected by these changes, a design refactoring is indicated which can alter the design so that it can provide the required performance measures.

4.2.2 CPASA at Work

As a part of the work in this thesis, a performance assessment tool (UML-JMT [11]) was developed. This tool was designed to deploy the CPASA Performance evaluation tests. The UML-JMT tool is an interactive system that provides the software designer with the means to automatically assess the performance characteristics of an architectural model represented in UML, by converting this architectural model and some key scenarios to an equivalent EQN (Extended Queuing Network) performance model. This will help in the process of requirement verification for performance non-functional requirement. UML-JMT adopts a component oriented view of software systems. That is, it models a system as a set of components. These components reside in the system according to a specific structure and interact according to specified behaviours. The component based representation will allow the conduct of performance studies with different degrees of abstraction. This is essential in agile development as the level of detail about the system specifications increases as the development iterations proceed. At the initial iteration, the UML-JMT tool can be used to verify that the configuration of the initial architecture will meet the anticipated performance requirements. As the iterations progress, the UML-JMT tool can be used to build more detailed performance models which can be used to obtain more accurate performance indices representing the system. As requirements may change, the UML-JMT can be used to study the effect of these changes in the overall system performance characteristics. The performance requirements verification process provided by the

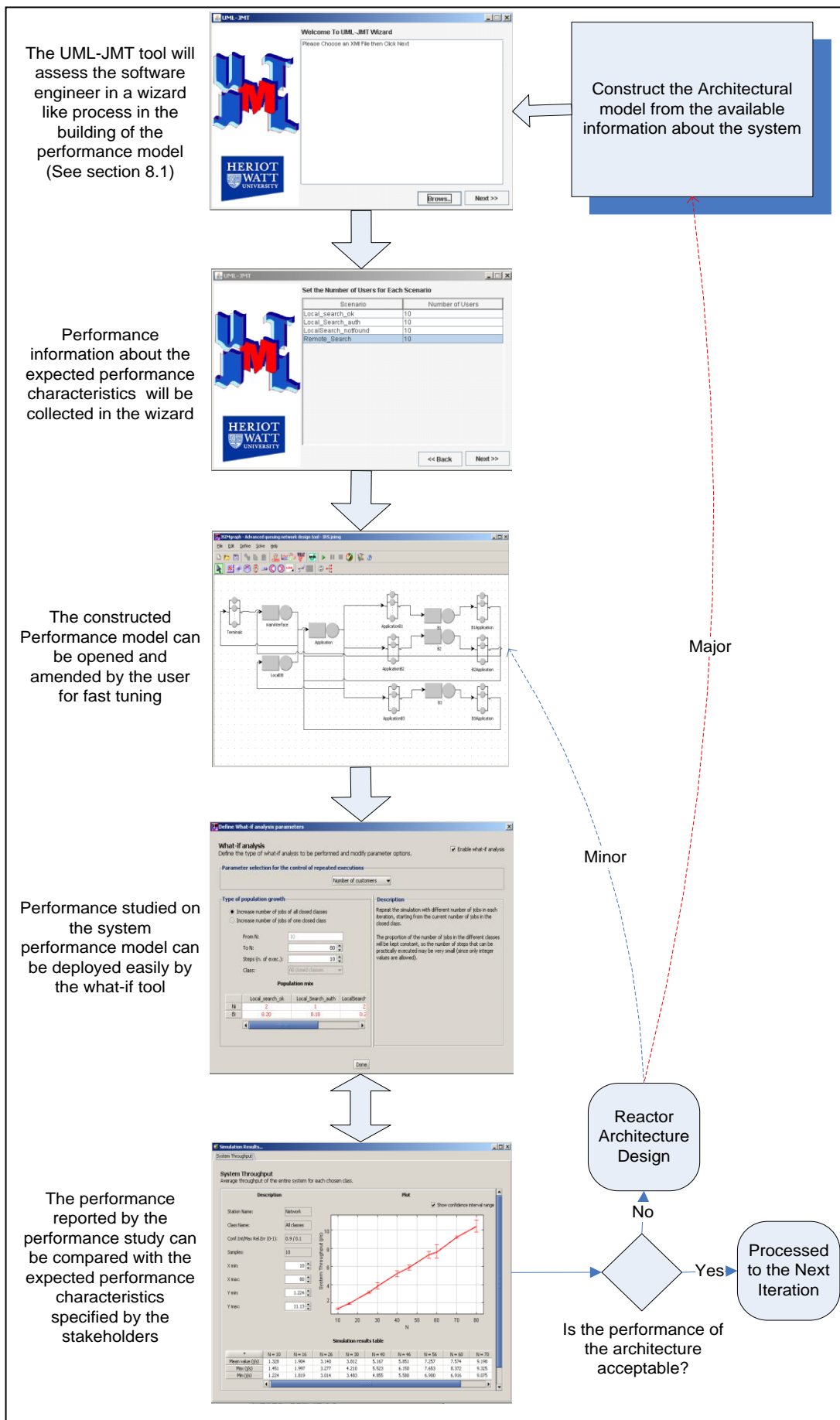


Figure 4.2. Using the UML-JMT as PE tool for CPASA assessment

UML-JMT tool can assist the user with the building of a relatively accurate performance model from the design specifications of the system under study, which can be simulated to provide the performance indices of the design. These indices can be compared to the required performance aspects of the projected system. The main advantage of the UML-JMT is that it provides a highly accurate, cost-efficient means of evaluating the performance characteristics of a software design.

Figure 4.2 illustrates the process of using the UML-JMT tool for conducting the PE tests specified in the CPASA method. The UML-JMT will accept UML models representing the architectural design and the key scenarios deployed in that architecture. UML-JMT will receive these as an XMI document containing the use-case, sequence and deployment UML diagrams modelling the structure and behaviour. The UML-JMT will analyse these diagrams then it will query the user about the performance characteristics of the required performance model (i.e. workload intensity, service time, average delay in networks ... etc). The EQN performance model representing the studied architecture will be opened to the user in the JMT performance evaluation suite [81]. This model can be tested using the model evaluation tools available in the JMT suite (i.e. what-if tool [96]). The results will be available for the user to compare with the anticipated performance indices. If the results do not meet the required performance measures, the user can make a major or minor modification to the architecture. A major refactoring decision will require the PE test to be repeated, as the performance model of the new architecture design will differ from the old design. Minor tuning can be carried out on the performance model directly. Such minor tuning includes increasing the number of service threads, using a faster network, using faster hardware ... etc.

4.3 Performance Evaluation of System Architecture

The previous two sections have discussed the software performance engineering methodologies. We explained earlier that software engineering methodology employs *performance evaluation techniques* for the process of assessing the performance aspects of system architecture. We have already discussed one of these methodologies (SPE) in 4.1.1. As we previously explained, one of the drawbacks of this methodology was in its dependability on a non-standard system architecture and behaviour notation. This drawback was caused by the absence of a similar standard until the introduction of the UML modelling notations (see 2.3). The introduction of the UML notation led the research of performance evaluation to utilise the UML architecture and behaviour

artefacts in the process of building the performance model used in the performance study of the performance evaluation procedure.

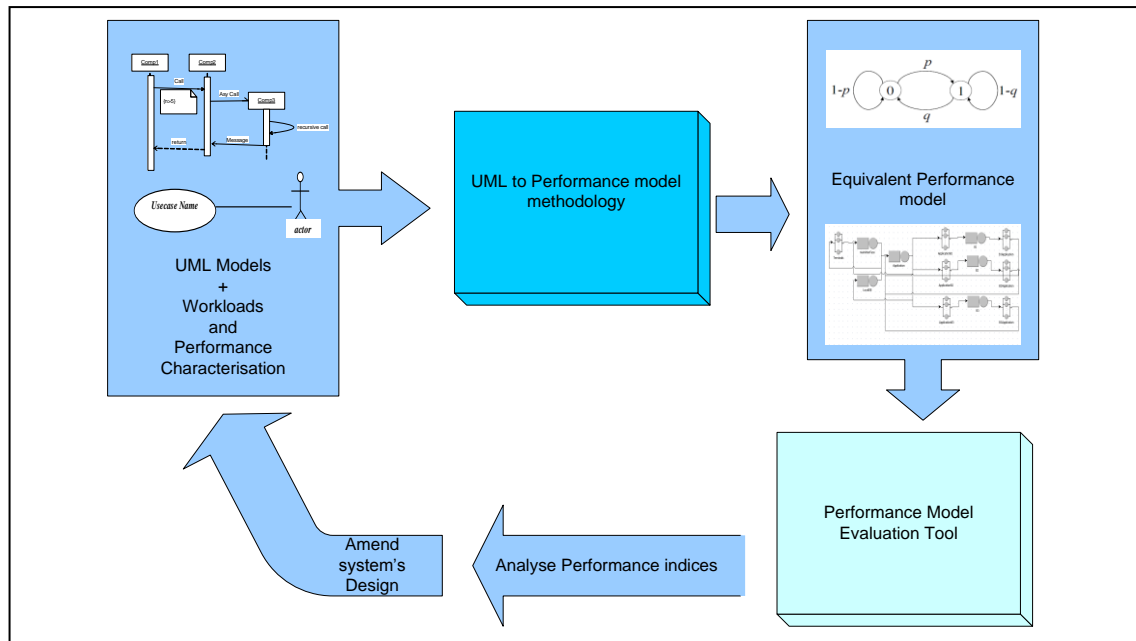


Figure 4.3: The steps of a system performance study using the UML to performance models methodologies

Literature reports a number of methodologies for extracting a variety of performance models from different architectural and behavioural UML models. King and Pooley suggested the *state marking methodology* to derive GSPN performance models from UML diagrams [97]. Also Grao *et al.* [98] suggested a methodology to translate a UML activity diagram - associated with performance annotations from UML proposed profile - to GSPN. Ping and Petriu suggested an algorithm that will transform UML to Layered Queuing Networks (LQN) [99]. More recent papers have suggested algorithms to translate UML to Stochastic process algebra; examples are a method by Canevet *et al.* which describes an algorithm that will automate the extraction of PEPA models from UML state chart and collaboration models [100], and a paper by Bennett *et al.* which describes their methodology for extracting FSP models from UML models [101]. In this thesis we will explain two methodologies for extracting Markov models and EQN from UML models, and we will discuss this in Chapters 5 and 6 respectively. In this section we will discuss some of the methodologies suggested for performance evaluation of a system's architecture by evaluating these methodologies against a set of criteria which we defined. These criteria define the fundamental aspects that should be available in a performance evaluation methodology in order for it to be effectively used in a

performance engineering process. We will discuss and justify these criteria in Sub-section 4.3.1. Then, in Sub-section 4.3.2, we will make a survey of a selected set of performance model extraction methodologies in the context of these criteria.

Figure 4.3 illustrates a template of the role played by the methodologies discussed in this section, in the process of conducting a performance study. The methodologies discussed here start by collecting the information required to build the performance model from the UML architectural and/or behavioural models. The UML models used differ from one methodology to another; this will depend mainly on the type of the resulting performance model. Along with the UML models, the system workloads and resource demands are also required in the process of building the performance model. The process of building the performance model differs from one methodology to the next. The result of these methodologies is a performance model that can be solved by a performance evaluation tool. The solution can be analytical or by simulation, depending on the type of the performance model. Some of the methodologies provide tools that will include the two blocks in Figure 4.3 which represent the building process and the evaluation process. The performance indices gained from the evaluation of the performance model can be analysed in order to suggest plans for amending the system's design.

4.3.1 Evaluating the Methodologies

The main goal of these performance model building methodologies is to decrease the costs of conducting the performance study tasks deployed in any performance engineering framework. These costs arise from the resources and the time needed for programming simulation models. These methodologies intend to reduce the knowledge gap between software engineering and performance engineering, by providing a black box approach that will help in conducting the performance study, which is important in the performance requirement verification process. To achieve this goal, the methodology has to comply with four main criteria which are:

Time Efficiency: The methodology has to be cost efficient in the sense that it will generate performance models with an acceptable degree of accuracy and with minimal cost. One of the important resources in software development, apart from financial resources, is time. This is essential in the context of software performance engineering as one of the main causes for the exclusion of performance engineering from project

plans is, once again, the complexity and time consumption of this procedure. The methodology has to provide results that do not require extra time to learn or deploy. The efficiency criterion can be measured by the following factors:

- *The deployment of the methodology* does not require the use of non-standard models or meta-models for the system or the performance. The time required to learn these new models will affect the simplicity of the methodology, and further still, will conflict with the goal of bridging the knowledge gap between software and performance engineering.
- *The resulting performance model* has to be easy to evaluate (solve) and analyse. The performance model produced needs to be supported by efficient and easy-to-use evaluation and analysis tools.
- *Availability of tools that deploy the methodology* is one of the key factors that defines the time efficiency of a methodology. This will help in providing software engineers with minimal knowledge in performance engineering terminology, with the means to conduct performance assessment studies, without the need to fully understand the steps or the theory of the methodology. This factor is also related to the automation criterion.

Generality and Transparency: The generality and transparency criteria are mainly related to the performance model produced by the methodology. The ability of a methodology to represent a performance model capable of representing all classes of system architectures is an essential factor. As we discussed previously in Chapter 3, some analytical performance models are limited in their representation of some architectural and behavioural aspects. This limitation will affect the generality of the methodology. We have already discussed the requirements of studying a system architectural design in Section 3.5 and we explained that analytical modelling provided the “best” performance models from the cost-time perspective. And we further explained that queuing networks provided the means to combine architectural and behavioural aspects of a software design. The limitation of the analytical solution algorithms for product-form queuing networks led us to adopt simulation based solutions for non-product form queuing networks. Transparency criterion means the ability to reflect the architecture and behaviour from the performance model. This is essential in reverse engineering process. In some cases, design tuning is made directly to the performance model. The ability to trace these changes to the design model is a useful feature.

Automation: automation means that the methodology needs to be systematic in a way which allows it to be automated in the tool that will deploy it. The availability of a tool for deploying the methodology is essential in automating the non-functional requirements verification task, which is a beneficial practice in some software engineering methodologies (e.g. Agile). Also, it is essential in the time efficiency criterion, as we discussed above.

4.3.2 UML to Performance Model Methodologies

Literature reports a range of methodologies dedicated for transforming UML models to equivalent performance models. These methodologies differ from each other in the deployment of UML as a representation of the system architecture or as structural language used to represent a specific system structure or behaviour. The deployment and utilisation of these methodologies depend on the stage of development, on which these methodologies can be deployed, the objective and nature of the performance study and the level of detail of the information available during the deployment of the performance study. The complacence of these methodologies with the criteria discussed in the previous section depend largely on the type of performance model produced, the nature of the transformation method (syntactic or semantic) and degree of the effort invested in realising these methodologies.

For the methodologies that embrace the architectural prospective of the UML models, these methodologies consult UML models that represent both the structural and behavioural aspects of a system. The resulting performance model is a model capable of representing both of these aspects (i.e. queuing networks, stochastic process algebra and simulation models). Most of these methodologies have a high degree of generality as the performance models produced are capable of representing a broad range of systems structural and behavioural characteristics, except for these methodologies that produce analytical models with limitations on the assumptions made by the solution algorithms (e.g. PFQN such as [102]). The leading methodology for this class of model transformation methodology is the SPE [6; 92] discussed earlier and the methodologies adopting the same concept of separation the software and machine models (e.g. [93; 102; 103; 104; 105]). These methodologies depend on the queuing network as the core output performance model, and therefore, the degree of efficiency from the prospective of the simplicity of solving the performance model will be high. The dependability of

some of these methodologies on representing the software model as an execution graph will limit the efficiency as the notation used is a non-standard modelling notation. Most of the methodologies adopting the SPE method create an extended version of the queuing networks modelling paradigm (i.e. EQN (e.g. [93]) and LQN (e.g. [102; 103; 104])) which will provide them with a high level of generality; on the other hand, there are methodologies that generate forms of PFQN which will affect their generality. These methodologies adopt syntactic algorithms that map UML models to equivalent performance models. This will improve the transparency of the methodology as this will enable the preservation of the architectural aspects of the system, and will maintain a notational linkage between the performance model and the UML models. In some of the methodologies not adopting the SPE method, the performance model produced will affect the levels of efficiency and transparency. This returns to the relation between the performance model generated by the methodology and the nature of the algorithms needed to build them. For example, the generation of a PEPA model (e.g. [100]) will require a semantic algorithm that may affect the transparency and further will lose the link between performance and architectural models. Also, the generation of simulation models (e.g. [106]) may affect the efficiency as the analysis of simulation results are time consuming.

Another type of model transformation methodologies adopt the UML as a notation for representing a particular structural or behavioural aspect of the system that need to be represented as a performance model. The majority of these methodologies are dedicated for generating a behavioural performance model for a system (e.g. GSPN [97; 107; 108]). This may affect the generality of a methodology, as these performance models are restricted in the sense of the types of system they can represent. Also, although these methodologies adopt both semantic and syntactic transformation algorithms, the behaviour dependent performance models produced by these methodologies will eventually affect the transparency of the methodology.

We have composed a survey for some of the work done in this field. We have discussed the methodologies in the context of the criteria we discussed in the previous section. For each of the methodologies discussed, we provided the input UML models, the output performance model, and summarized the performance model generation process. Then we classified these methodologies' compliance with the criteria defined in 4.3.1 as High, Mid (medium) or Low. High compliance reflects that the methodologies comply

with the entire factor defining a criterion or have extra features that cover the missing factors. Medium compliance refers to the methodology complying with some of the factors and low refers to the methodology not complying with any of the factors which define the criterion. For simplicity, the survey is formatted in a table form. Table 1 in appendix D contains a survey of some of the methodologies for performance evaluation. We have constricted it to methodologies that adopt the SPE method of separating structural and behavioural aspects. This is a return to the use of this method in the key performance model building methodology, as discussed in Chapter 6 of this thesis. For the same reason, we also concentrated on methodologies generating QN models. We have included in the survey selected methodologies that produce other types of performance models (i.e. Petri Nets, Process algebra and simulation).

4.4 Summary

Software Performance Engineering refers to the performance related analysis and activities incorporated in the software engineering process. The importance of software performance engineering arises from the need for methods that will provide assurance of the quality of software systems during development and maintenance. In general, performance engineering tasks are incorporated into the development's design and test phases as a means of ensuring that the suggested design meets the QoS requirements. The integration of the performance engineering into software engineering will depend on the availability of the requirements, as performance is one of system's characterises that are affected by the whole system. This is why we distinguished the software engineering paradigms according to the availability of the requirements to conventional (i.e. waterfall development paradigm) and agile. For the conventional, we have described the PASA framework, whereas for the agile development paradigm, we introduced the CPASA framework. This chapter defined the role of performance evaluation in software development and described how to integrate performance evaluation into the software engineering process for these two main software engineering paradigms.

Software engineering methodology employs performance evaluation techniques for the process of assessing the performance aspects of system architecture. We have discussed the role played by the model transformation methodologies in the process of conducting a performance study. And we have composed a set of criteria for comparing these methodologies according to the system types they service and the nature of the algorithms used for the conversion process. This chapter included a literature review of

these methodologies based on these criteria. This comparison was essential for the design of a methodology that will avoid any drawbacks this thesis domain.

An Application of the State Marking Methodology

This chapter explains a methodology which was the result of the author's early studies in the field of performance evaluation automation techniques. This methodology was published in [33; 109]. The work involved when developing this methodology consisted of automating the extraction of a generic performance model. The development of the methodology concentrated on three fundamental criteria:

- Firstly, the method has to be *simple* which will allow the user to deploy it easily without the need for learning new notations or out of context operations.
- Secondly, the method has to be general in terms of its ability to model any expected system.
- Thirdly, the method has to be systematic in a way that will allow the methodology to be automated; this will allow the development of a tool to deploy the methodology.

The methodology extended in this chapter is based on the *state marking* methodology, originally developed by King and Pooley[14]. The state marking methodology concentrates on capturing the behavioural aspects of the modelled system in a behaviour oriented performance model. The original state marking methodology proposed a method for extracting a GSPN performance model from a meta-model composed of collaboration and state-chart models. The limited generality of the GSPN and the non-standard input model used, motivated the extension of the state marking methodology. The extended methodology proposes a systematic approach for extracting Markov chain models from *performance annotated* sequence UML models[1]. Section 5.1 will summarise the original state-marking methodology developed by King and Pooley. Section 5.2 will discuss the extension suggested by the author of this thesis. As the extended methodology was not developed as a tool, Section 5.4 will discuss the technical requirements and possibilities for the development of a tool that will deploy this methodology.

5.1 The State Marking Methodology

The *State Marking Methodology* is a performance evaluation methodology that can be classified as one of the structural model based methodologies described in Section 4.3. The state marking methodology captures the behavioural aspects of a software system by building a performance model which represents the overall system states. The methodology was first introduced by King and Pooley in [14]. Their proposed methodology was to use a behavioural model composed from the UML collaboration and state chart diagrams for deriving GSPN performance models. The suggested methodology builds the GSPN with states representing change in the UML behaviour model. The resulting GSPN will represent a modelling of the overall behaviour of the modelled system. Although this method provides a simple approach to extract performance models from UML model, the use of GSPN as the target performance model affected the generality of the method, as it lacks the ability to model some kinds of system architectures (i.e. some systems with specific scheduling schemes for sharing resources). Pooley avoided this problem in [3] as he generalised the method to generate Markov chain performance models directly from UML collaboration-state chart diagrams. This gave the method the advantage of simplicity and generality. This section explains the state marking methodology and evaluates this methodology in order to justify the work carried out in modifying and automating this method.

5.1.1 System Representation

The state marking methodology utilises a number of UML models in the process of extracting the required performance model. The workload characterisation of the modelled system is represented in the use-case model. Different use-cases represent each functional request representing the main workloads affecting the system. The structural specification of the system is represented by the collaboration UML model.

The collaboration model of a system represents the objects composing the systems with associations representing the interaction between these objects. The state-chart diagram defines the internal behaviour of each of the components composing the system. It is composed of a set of states with transactions between them. The transactions are triggered by a set of actions. State marking methodology uses a meta-model known as a collaboration-state model, which is a combination of collaboration and state-chart models.

The model shown in figure 5.1[3] represents a producer consumer system with a buffer. The system consists of three objects; a producer, a consumer and a three spaces buffer. The collaboration-state diagram shows the interaction between the objects as the producer objects add to the buffer and the consumer takes from the buffer. Inside each of these objects there is a state chart that represents the behaviour of that object. This state chart will model the internal behaviour of the object itself as it performs its functions. This diagram represents the different states that a system can have and is used in the process of marking and registering the different states of the system behaviour in the performance model.

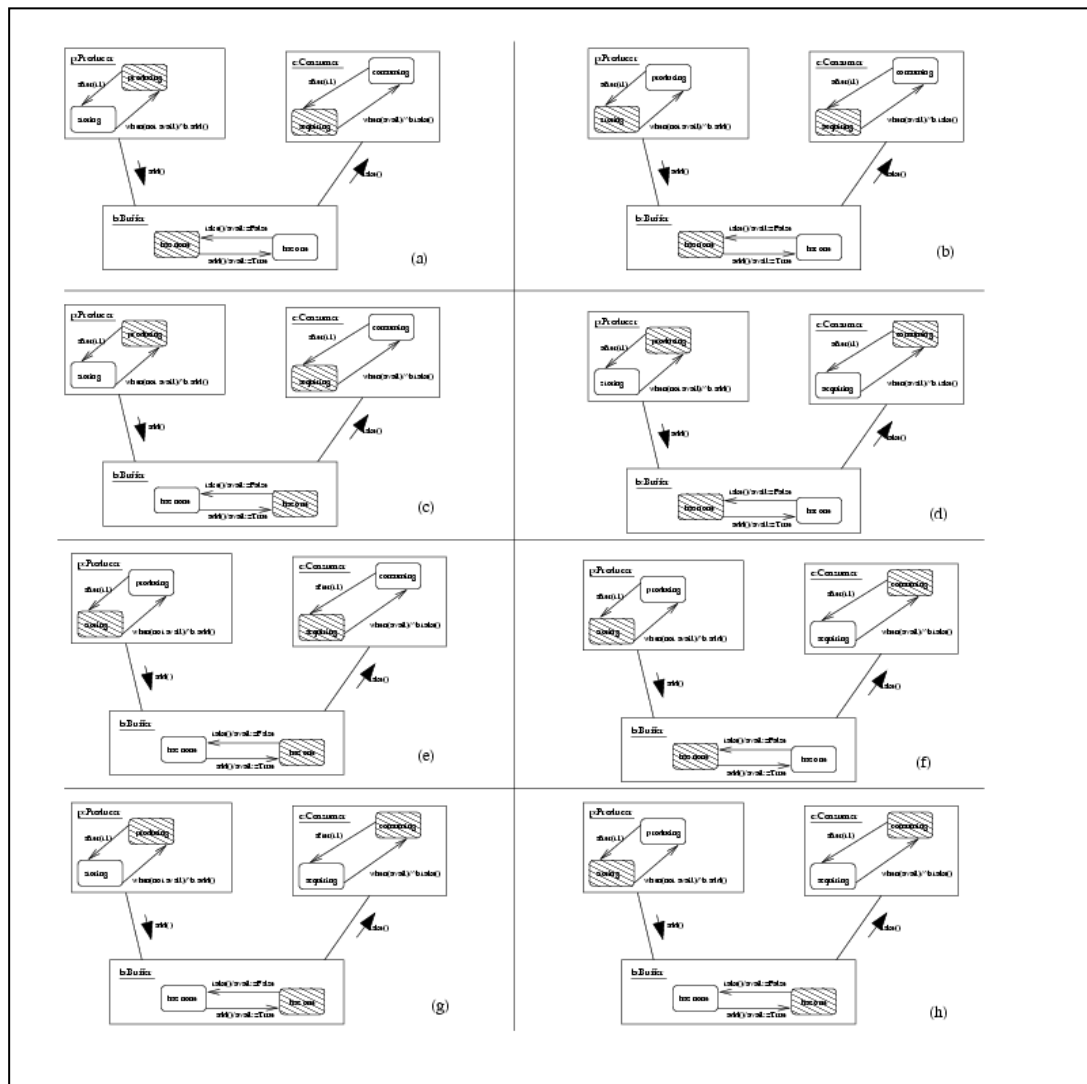


Figure 5.1: Registering the systems states for produced consumer system [3]

5.1.2 State Marking With GSPN

The process of building a GSPN from a collaboration-state model is as follows:

- **For each** object in the collaboration diagram, build a corresponding SPN model as follows:

- ⇒ Each state in the state-chart model representing the object has a place in the SPN model.
- ⇒ Each transition in the state-chart models represents transitions in the SPN model. The mapping of the transitions between the SPN model and the state-chart model depends on the states involved this transition taking into consideration that transitions in the SPN have only one input place and one output place.
- ⇒ The Token is placed in the state, representing the current state of the state chart model.
- *After* constructing the individual PN models for each state model, these individual PN will be connected together to compose a global PN that represent the entire system according to the association defined by the collaboration diagram.

5.1.3 State Marking With Markov Chains

The extension of the state-marking methodology suggested by Pooley in [3] used the same concept as the original state marking methodology. It involved transforming a UML collaboration/state model into a Markov chain model. The transformation approach involved a marking algorithm that will catch and register the state of the entire system for each step executed. Each of the registered states will represent a state of the overall Markov chain. The arcs between the states are represented with steps causing the move from one step to the other, as shown in Algorithm 5.1. Applying the algorithm on the produced consumer system shown in the collaboration-state model in Figure 5.1, the Markov chain model in Figure 5.2 can be gained. This extension of the methodology

```

current state = new_state
snapshot set  = current_state
while (there exist more actions)
{
    Take an action
    new_state = Execution of action on the current state
    if (new_state in snapshot set) // the state already exist
        add an edge from the current state to new state
    else
        {
            add new_state
            add an edge from current state to new state
        }
    Current state= new_state
}

```

Algorithm 5.1: State marking methodology for transforming UML (collaboration-state chart) diagram to Markov chain.

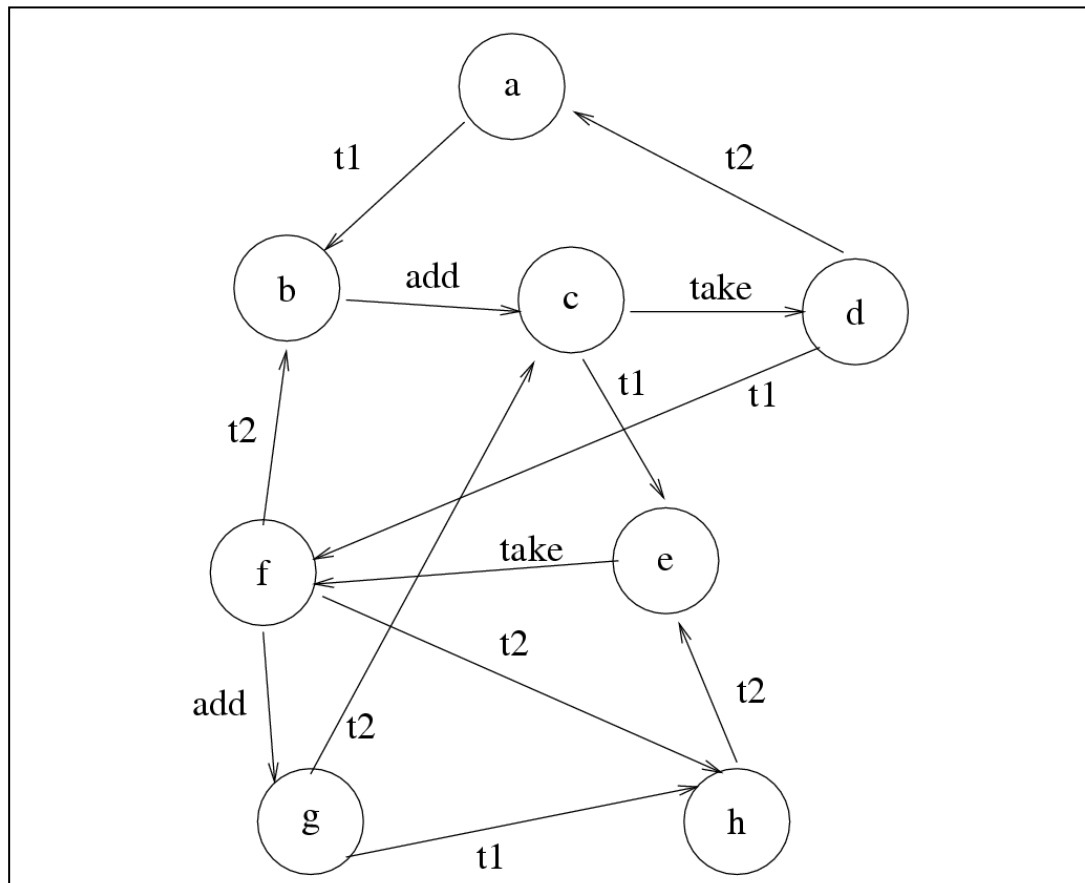


Figure 5.2: Markov Chain produced from the deployment of the State marking methodology on the produced consumer system in Fig 5.1 [3]

can be viewed as producing the reachability graph of the SPN. This would improve the methodology by making the methodology more systematic.

One of the reasons for extending the methodology to produce Markov chain as the performance model comes back to the useful performance indices and the availability of analysis information provided by a Markov model. Another advantage gained from using the Markov chain is its generality in the context that a Markov model is capable of representing all kinds of system behaviours. As explained in Chapter 3, the solution of a Markov chain model involves solving its *state transition matrix*. The representation of some large and complex systems, such as a Markov chain, can be extremely difficult. This is due to the number of states in such a system that can grow exponentially with the number of elements.

5.2 Extending the Methodology

The previous section explained the deployment of the state marking methodology to obtain different types of behavioural performance models from a meta-model composed from the collaboration and state-chart models. As we recall from 4.2.1, the usage of non-standard meta-models as the input models for the methodology might affect the

simplicity of the methodology, and as a result, will affect the time/efficiency factor for the deployment of the methodology. Although the deployment of the methodology does not require altering the originality of the two models, the non-standard coupling of the two models will affect the simplicity of deploying the methodology. The original methodology for extracting GSPN used a systematic algorithm for the performance model generation. This algorithm used a one-to-one approach for defining the GSPN network places and transactions from the state-chart model. The Markov model version of the transformation methodology required a simulation-based algorithm (Algorithm 1) for capturing the system's states snapshots. This algorithm has the disadvantage of its dependent on the state space, in addition to the complexity of systematically defining a generic algorithm for generating all the states in a system's state space. This inspired AlAbdullatif and Pooley[33] to extend the state marking methodology to avoid the disadvantages mentioned above, and to take the state marking methodology to the scenario level. The original state marking methodology aims to represent a comprehensive model representing the behavioural aspects of the whole system. In some performance studies, it is more convenient to consider the performance of specific critical scenarios than to study the performance of the system as whole.

The extended state marking methodology uses a standard UML behaviour model as its input system model. This model was chosen for its ability to represent the behavioural and collaboration aspects of the different scenarios representing the behaviour of the system. The UML model used in the extended methodology is the *performance-annotated* sequence diagram. This class of UML diagrams comply with the recently adopted UML Profile for Schedulability, Performance and Time[1]. The extended methodology defines a systematic algorithm for building a Markov performance model from a performance annotated sequence diagram. This algorithm builds a performance model based on the artefacts of the sequence diagram(s) used in the modelling process. This section will describe the extended version of the state marking methodology. First, we will explain the UML model used as the input for this methodology. Then we will discuss the Markov model used as the output and process of mapping the workload information annotated in the UML model on the performance model. Finally, we will discuss the steps of the methodology and further explain the methodology, using an example.

5.2.1 Input Model Representation

As explained in Chapter 2, In UML an *interaction diagram* is one that shows how a group of participants (objects and actors) in a system collaborate in some behaviours. Sequence diagrams as well as collaboration diagrams are members of the interaction diagrams family. They can be used to capture a specific scenario of the system by showing how the objects involved in that scenario collaborate by exchanging messages to perform a specific behaviour. There are four main types of message in a sequence diagram; synchronous, asynchronous, reply and found messages. Synchronous messages are the ones in which the sender will enter a wait state until a reply from the receiver arrives. Asynchronous messages are the ones in which the sender will continue with its work after sending the message to the receiver. Reply messages are sent in response to a synchronous message and found messages are the messages initiated from outside the sequence diagram, to start the scenario.

The participants in sequence diagrams have a life line that represents the flow of actions resulting from participation in the scenario by that member. An activation bar on top of the life line shows the time during which the participant is active in the interaction. Control logic can be modelled in sequence diagrams using interaction frames. The interaction frame labelled 'loop' is used to model iteration in a section of the sequence diagram, while the labels 'alt' and 'opt' are used to model conditional sections. Interaction frames can also be used to illustrate concurrency with the help of the label 'par', representing concurrent activities, and 'region', to mark a critical section [110].

Figure 5.9 in the example shows a sequence diagram: The participating members are drawn at the top of the diagram as boxes. Each of these boxes has a dotted line coming down from it, representing the life line. The thick grey line on top of a life line is the activation bar. The messages are denoted by arrows, each message type having a specific arrow style; a synchronous message is presented with black arrow head, whereas an asynchronous message is presented with an empty arrow head. Reply messages take the form of a dotted line arrow. Figure 5.9 shows a loop activation frame; this frame is presented as a box covering the area that it affects with the guard condition in the corner.

Annotated Sequence Diagrams

One of the main concerns in the performance model building methodologies is the representation of performance workload and resource usage characteristics of the

modelled system. In the early methodologies where UML was not a standard modelling notation, different techniques were suggested for representing these performance characteristics (e.g. SDL[111], LOTOS[112]). Pooley and King proposed in [97] a method to include performance data in UML diagrams in the form of performance tags (time labels). Chapter 7 of the "UML Profile for Schedulability, Performance and Time"- adopted by OMG in 2005 - is an extension of the UML standard to accommodate UML quantitative performance annotations. These annotations allow the association of performance related quality of service (QoS) characteristics with selected elements of a UML model [1]. The profile explains these extensions to the UML standard in the context of the standard itself. It defines stereotypes, tagged values and constraints that represent the performance requirements and resource allocation of the modelled system.

The main stereotypes used for performance modelling include *PAclosedLoad*, *PAopenLoad*, *PAhost*, *PAstep* and *PAresource*. The first two of these stereotypes represent the way the work is fed to the system or, as it is often described, the workload. The *PAclosedLoad* stereotype represents a closed workload; it has four tags: *PArespTime*, *PApriority*, *PApopulation*, *PAextDelay*. *PAopenLoad* models an open workload with the tags: *PArespTime*, *PApriority*, *PAoccurrence*. The objects or participants in the system are classified as either *PAhost*, modelling a processing resource with tags including: *PAutilisation*, *PAschedPolicy*, *PApreemptable*, *PAthroughput*, or *PAresource*, modelling a passive resource with tags including *PAutilisation*, *PArespTime*, *PAthroughput*. A *PAstep* models a scenario step with tags including *PAdemand* defining a step's execution time, *PArespTime* defining a step's response time, *PAprob* which represent probability to execute the step and *PAdelay* which shows the time before executing the step [1].

5.2.2 Output Performance Model

The output performance model from this methodology represents a CTMC (Continuous Time Markov Chain) discussed earlier in Chapter 3. The sequence diagram that we will use to define a performance model in this methodology will be annotated with the performance stereotypes defined above. Each of the performance stereotypes can be accompanied by an appropriate sequence diagram section. The workload tags can be added to the found message at the beginning of the diagram to define the nature, ratio of the initiating messages and the expected QoS characteristics of the scenario in hand.

Host and resource stereotypes can be used to describe the performance features of the participating members in the scenario that will help in the performance study of the system. The main stereotype that will be used is the PAstep, which will be used to describe the performance tags for every message defining the collaboration between the participants. The performance information that they define in their tags will be used to label the arcs in the Markov chain model. PAstep will also be used to define the performance information for the interaction frames in a diagram, such as defining the average number of iterations in a loop or the probability of a condition being true. This kind of performance information is a key factor in the performance model that we are trying to build.

5.2.3 Extracting the Performance Model

The method for extracting the performance model is as follows: First an initial state is defined, recording the initial state of the system before executing the first step. Then, for each of the steps defining the scenario in hand, the step is “executed” and the status of the system is marked, which may create a new state of the system or return to one passed through before. If this state is a new state, then a new node is created in the Markov chain with an edge from the previous state to the new state. This edge will be tagged with suitable a PAstep tag (usually PAprob) depending on the current model. This algorithm will continue until all the steps are executed. For simplicity, we will use the probabilities to mark Markov edges instead of rates, as probabilities can be computed from these rates as explained in Chapter 3. The resulting series of markings of the sequence diagram forms the Markov chain model of the system. The execution of a step will differ according to the type of diagram element being executed in the step. There are a number of elements in a sequence diagram, as described above, and each of these will have a different representation in the performance model.

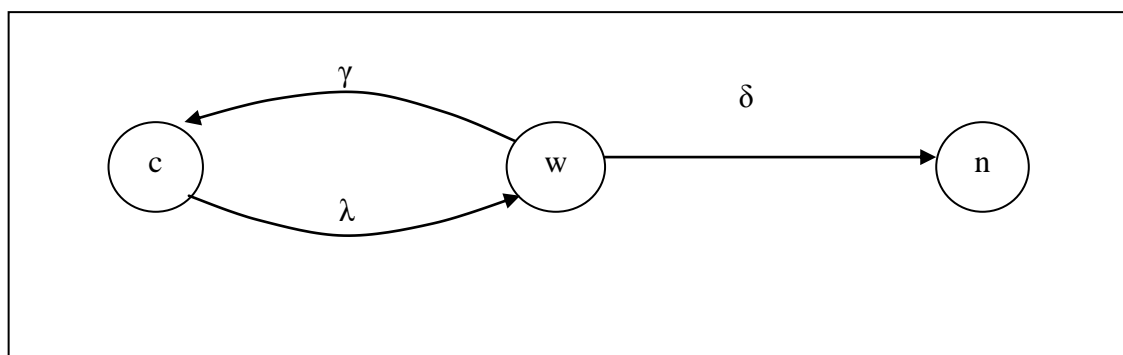


Figure 5.3: Markov model for a Synchronous message.

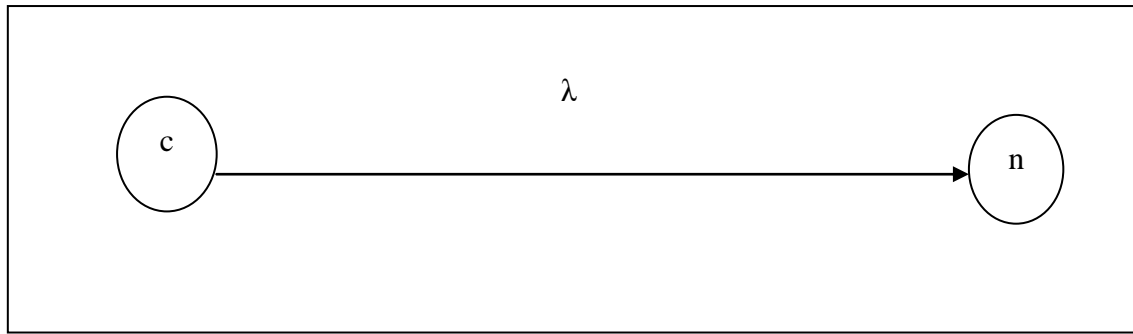


Figure 5.4: Markov model for asynchronous message.

Synchronous/Asynchronous Messages

We will now describe how a performance model can be constructed using a variety of readymade Markov Chains performance model components, these pre-cast performance model components will cover sequence diagram notations such as messages and interaction frames, composing a sequence diagram. For synchronous messages, the system will be in a specific state, noted in Figure 5.3 as state c . In the execution of the action (response to the message), the system will wait for a response from the receiver, which means that it will reach the wait state denoted by w . The rate for moving from state c to w is noted by λ which represents the PApob tag of the PAsstep stereotype for that specific message. When the message is in the waiting state w , it will either go to a new state, n , on the arrival of the response message, with a rate of δ representing the average time for the reply message to arrive, or, if no reply arrives, the system will wait for a time out and then re-send the message (there is no constraint on multiple messages) and return to state c . The rate for resending the message γ depends on the average lost message ratio.

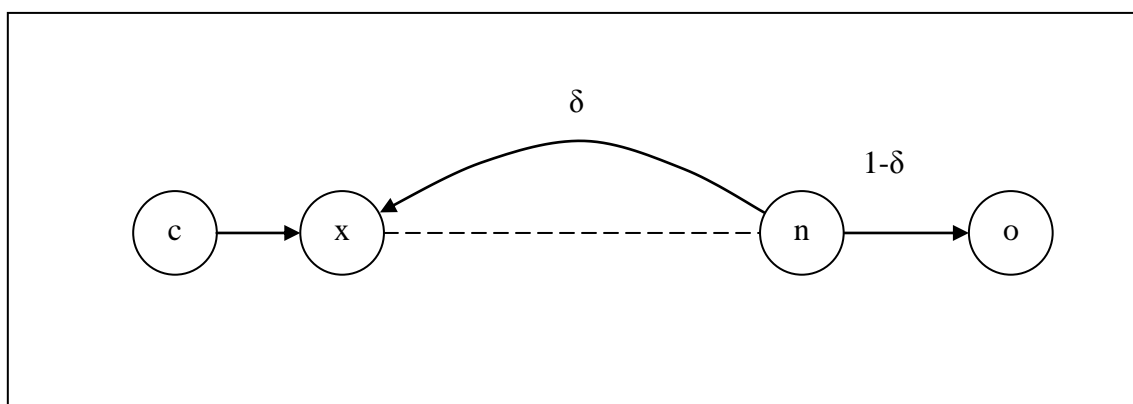


Figure 5.5: Markov model for a loop frame.

Figure 5.4 shows a Markov model of the system in the case where it receives an asynchronous message. As stated earlier, in the case of an asynchronous message the sender will send the message and then the system will continue with the next step without waiting. In this model, we have a current state c for the system, and when the message is processed the system will enter another state n , with a probability, λ . Synchronous and asynchronous messages represent the most common artefact used in sequence diagrams, this is true as the main goal of a sequence diagram is to illustrate the collaboration between the participating members of a scenario with messages.

Loop Interaction Frames

A looping interaction frame in a sequence diagram is shown as a Markov model in Figure 5.5. The content of the loop frame will be surrounded with a loop model. State x is the beginning of that loop, and state n is the inspection state where the loop condition will be checked. If it is true, an arc will return to x or another state (say o) indicating that we are out of the loop. The δ represents the probability that the looping condition evaluated as true.

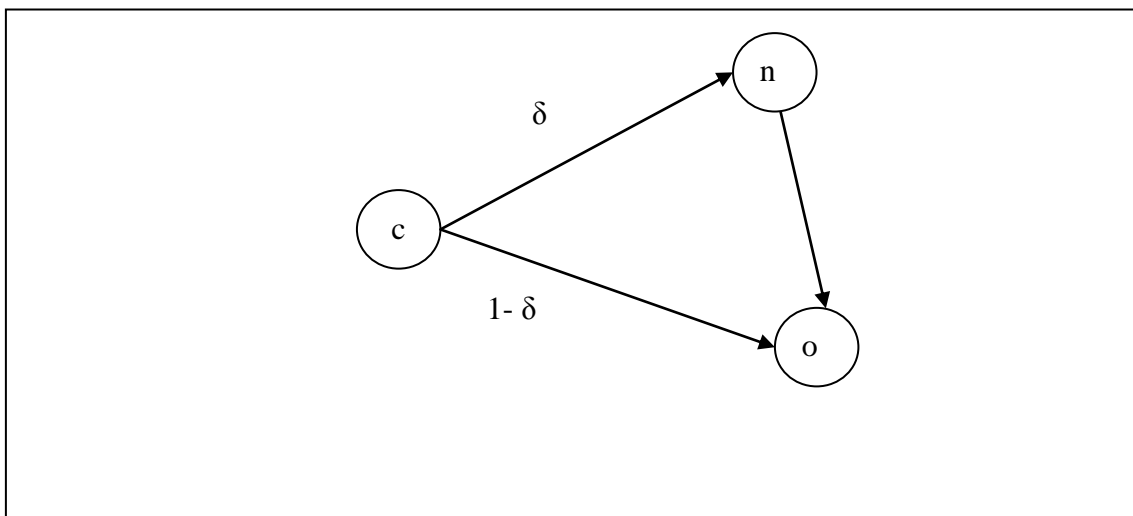


Figure 5.6: Markov model for an Alt frame.

Alt/opt Interaction Frames

In the case of a conditional interaction frame (alt and opt), the system will be in a new state if the condition evaluated is true, and either in the same state, or an alternative state, if the condition evaluated is false. This is represented in Figure 5.6 which shows how an alt frame can be modelled as a Markov chain. δ represents the probability that the condition evaluated is true. The system will be in a new state n if the condition evaluates to true and in another state o otherwise.

The model for an opt frame is similar to Figure 5.6 but has only a single outgoing arc from state c representing the system when the condition evaluates to true, when the system enters a new state n . It returns to the original state c otherwise.

Region Interaction Frames

The region interaction frame is used to identify a critical section that only a single process can enter at any time. The modelling of a critical section in a Markov model depends on the number of processes trying to access the critical section at a given time. Figure 5.8 shows how a critical section for a two process system is modelled; the states that a process can have are either to be out of a critical section (O), inside the critical section (i) or waiting to enter the critical section (w). In the figure the two processes start out of the critical section (state $[O, O']$) and either of them can enter the critical section (one of the states $[i, O']$ or $[O, i]$), usually with similar probability. In the case where one of them is inside the critical section and the other tries to enter, the latter will enter the waiting state ($[i, W']$ or $[W, i']$). From the wait state it will enter the “in” state when the process occupying the critical section leaves the critical section (one of the states $[i, O']$ or $[O, i]$).

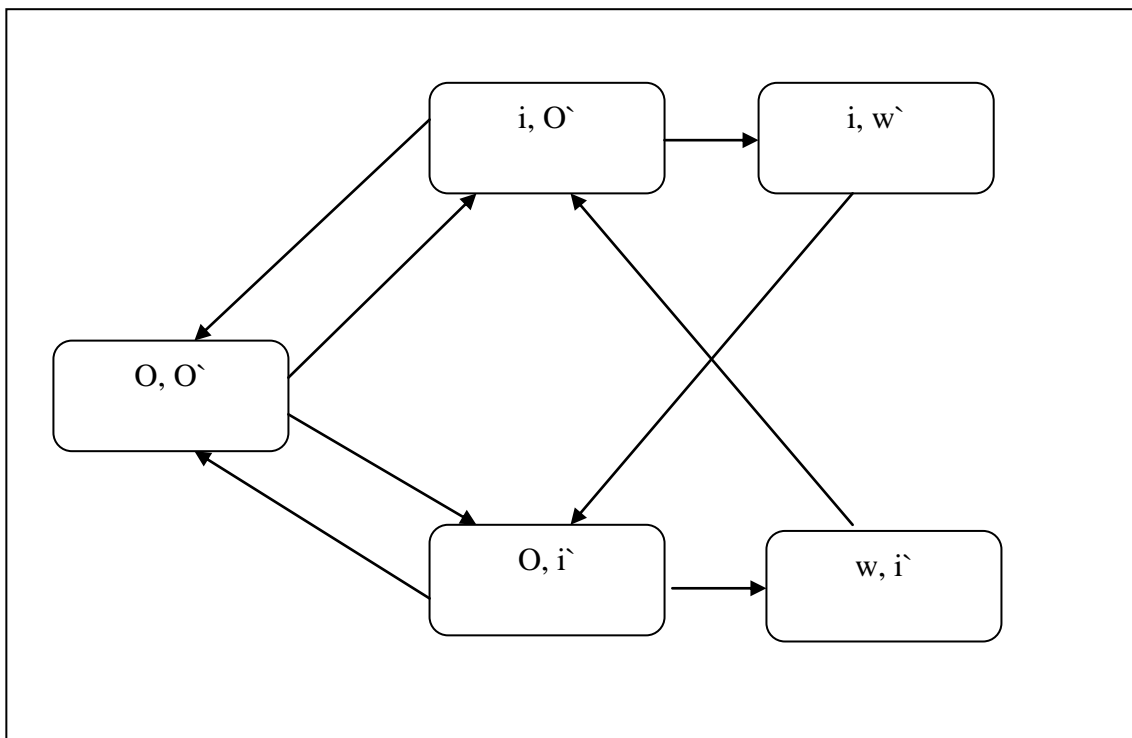


Figure 5.7: Markov model of a two process critical section.

The number of possible states for such a model will grow rapidly as the number of participating processes increases. Other modelling schemes like Petri Nets provide simpler notations to represent similar situations, but ultimately, the model underneath

will be as complex as the one in Figure 5.7. Our method states that for every critical section in the system, this critical section will be modelled according to the number of processes potentially trying to access it, and then be added to the complete model. This process, although a long one, is the only obvious way to model a critical section in a Markov model.

Parallel Interaction Frames

If we try to model parallel behaviour of a system, the main issue that may arise is that a parallel interaction frame represents the concurrent execution of actions (messages) in the system being modelled. In the case of encountering a parallel interaction frame, each of the execution branches will be modelled as a separate Markov model and these two models will fork at the beginning and join at the end of the parallel behaviour.

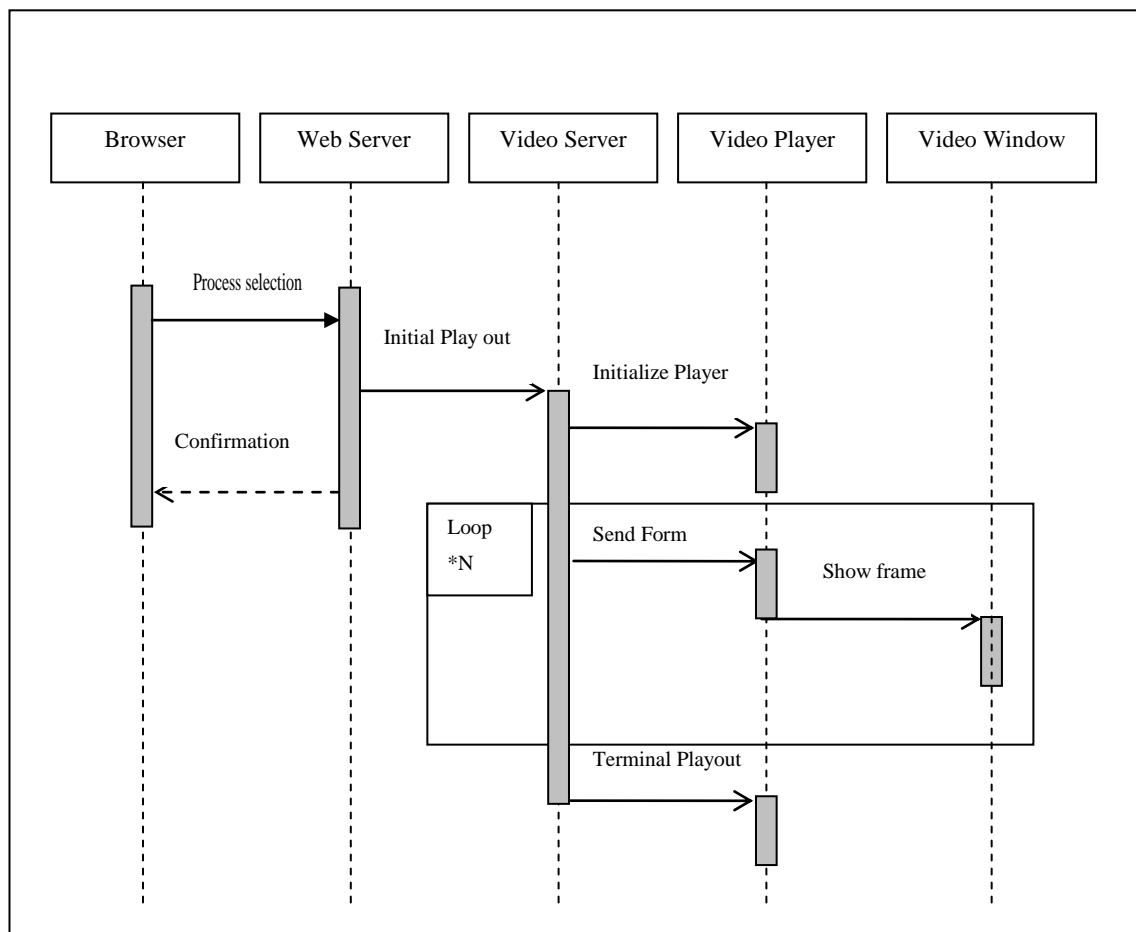


Figure 5.8: sequence diagram of a web video application, modified from [1]

5.2.4 Example: Web Video Application

The case study that will be used in this chapter is derived from the example provided in the Section 5.9 of the “UML Profile for Schedulability, Performance and Time” [1]. The system described is a web video application. This allows users to access video streams through their web browsers where they will be connected to a video server that contains

the streams. The video server then plays the requested stream on a video window that contains a video player. The components of the web video application and the relations between them are shown in Figure 5.8. In this case study we will choose one of the possible scenarios of the system, and then model this scenario as a sequence diagram. Using the QoS priorities described for this system in [1], we will compose an annotated sequence diagram and, finally, we will use our methodology to compose a Markov performance model from this diagram.

Figure 5.8 shows a sequence diagram for the scenario of a user accessing a video stream. First of all the user will choose a video to be played on their browser and the request for that video will be passed to the web server which will select the video server that has this specific stream. The web server will initialise a video player for the user and will start streaming frames to that player, to be shown in a video window. This process will continue until all frames of the video are sent to the user player, as described in the figure with a loop frame that will iterate for N times, where N is the number of frames of the stream.

The type of performance annotation to be added depends mainly on the context of the experiment. The performance requirement for this system is described in [1] in terms of response time for messages. In our case we require information about the probability of a message being sent. In the QoS there is a requirement for the confirmation response time stated as “the response time for the confirmation to the user that the request has been received”. This requirement is specified as a probability that the delay in receiving the confirmation will not last longer than half a second in 95% of the cases: Probability (Confirmation delay > 500 ms) < 0.05 or, expressed as a percentile measure: 95th percentile (Confirmation delay) < 500 ms” [1]. In this case, the time out constraint on the synchronous message process selection is to be less than half a second and the probability that the confirmation will arrive is 0.95, and 0.05 to resend. Another example of performance information that may be added to the system is on the video stream, as the frames fed back to the user should be displayed at regular intervals of 30ms, that the probability of a frame being displayed late is less than 1%: Probability (Interval between frame display instants < 30 ms) > 0.99. For our study we will use these two requirements and add them to the sequence diagram which will be consulted in the performance model building process.

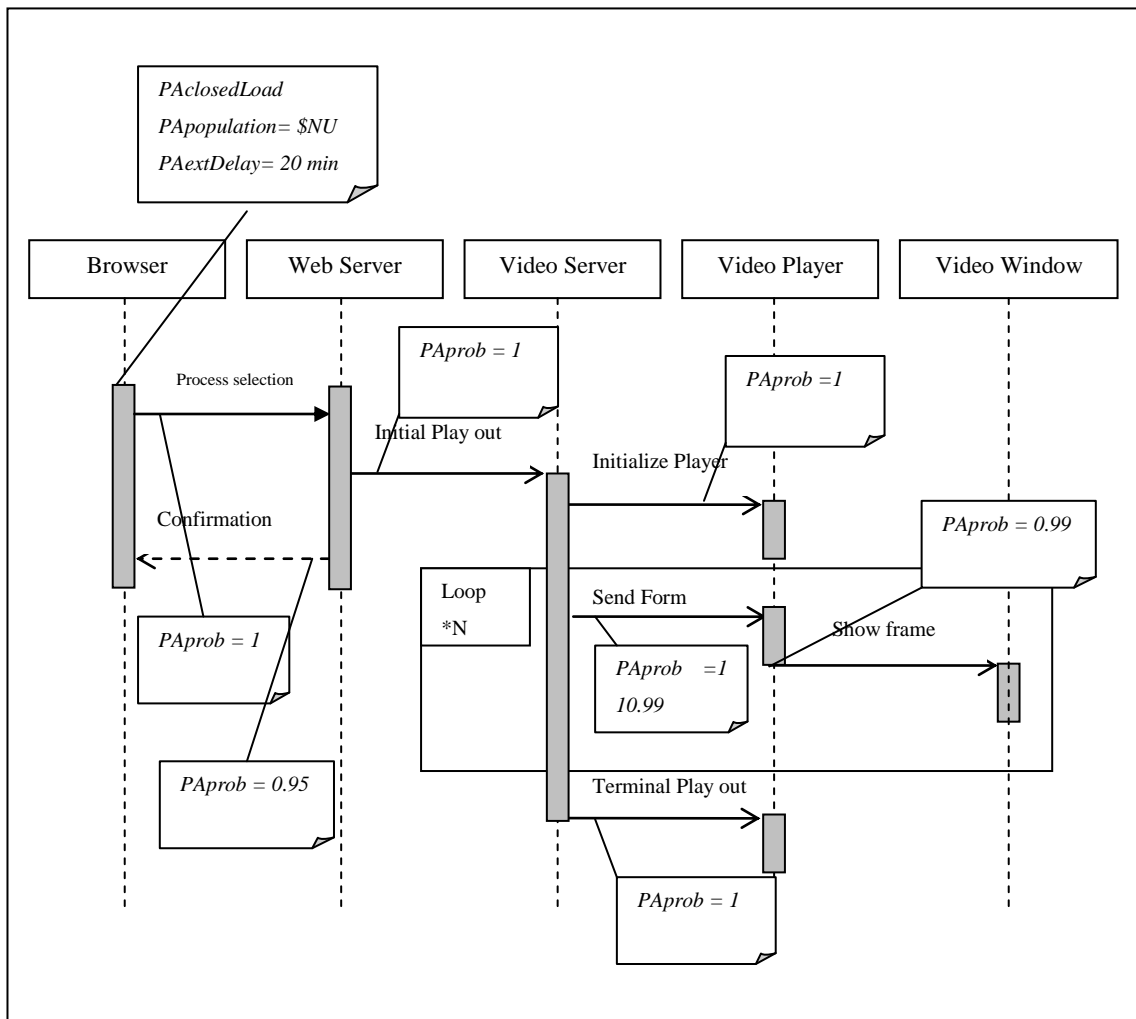


Figure 5.9: Annotated sequence diagram of a web video application, modified from [1]

Figure 5.9 shows the sequence diagram with added performance information. The labels added to the sequence diagram contain stereotypes for performance information and tags with their values. In this diagram we added two stereotypes: One of the load type as the system has a closed load with NU users where each user has an average delay between ending one session and beginning another of 20 minutes. The other is the $PAstep$ for labelling each of the messages in the diagram with a probability of occurrence according to the performance requirements that we describe in the QoS.

Figure 5.10 shows the Markov chain performance model extracted from the annotated sequence diagram. The state a , is the initial state and is a part of the representation of the first synchronous message in the sequence diagram, known as *process selection*. This representation includes also the states b and c as the waiting and response states respectively. The second message in our diagram is *initial payout*. This is an asynchronous message which will be modelled as in Figure 5.4 with state d . The same is true for the rest of the messages in the diagram, but, as we have a loop activity frame

surrounding the messages *send frame* and *show frame*, the representation of these two messages must be boxed in a loop model like the one in Figure 5.5. Here the x state in Figure 5.5 is represented with the f state in Figure 5.10 and the n state is represented by the i state. The ratios labelling the arcs are extracted from the PAProb tags in the diagram. The λ variable depends upon the average number of frames in the streams N .

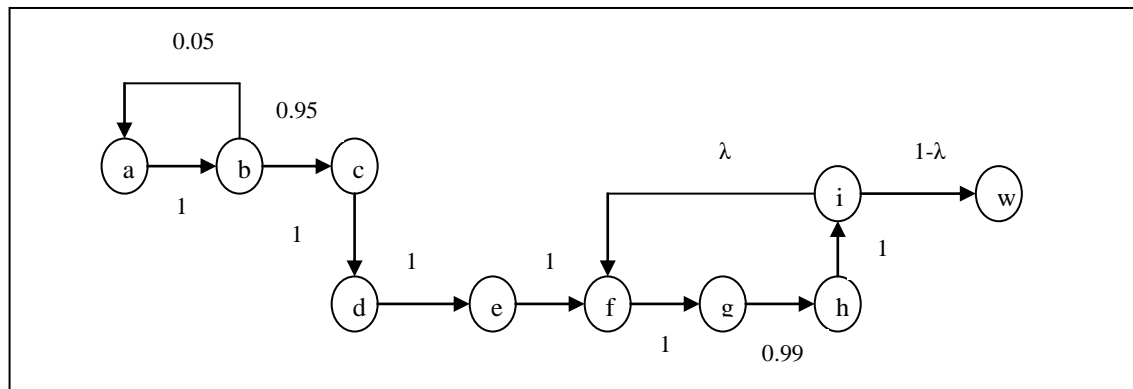


Figure 5.10: Markov chain of the video application sequence diagram

5.2.5 Evaluating the methodology

The performance evaluation of software systems is a highly valuable task, especially in the early stages of a software project. Many methods for integrating performance analysis into the software development process have been proposed. It is essential that these methodologies are simple, general and described systematically. We have evaluated the original state marking methodology in the beginning of this section and have noticed that the UML model used in this methodology affected the simplicity of the methodology, and the algorithm used in the Markov chain version of the methodology affected the automation of the original methodology. The version suggested by AlAbdullatif and Pooley focused on covering the disadvantages of the previous versions of the methodology. The input sequence diagram model chosen for this methodology represents a standard UML model suggested by the OMG in the UML Profile for Schedulability, Performance and Time. The output performance model represented by a Markov chain is general in the sense that all aspects of behaviour represented in a sequence diagram are covered by the methodology. The Algorithm provided by the methodology provides a systematic approach for building the output performance model, which will assist the automation of the methodology. The disadvantages of this methodology arise from the state-explosion problem reflected in the use of Markov chains as the output model. This problem is one of the main problems which limits the use of Markov chains in modelling complex systems.

In the context of the scope of this thesis, this methodology provides a straightforward and efficient way of representing the behavioural aspects of a software system in a performance model. As we recall from Chapter 3, the requirements of a performance model to study the performance aspect of a software design, required this performance model to include both the architectural and behavioural aspects of the system under study. Markov models lack the ability to represent architectural aspects of software systems. Furthermore, Markov models lack the model transparency criterion discussed in Chapter 4. This inspired the author to develop the performance evaluation methodology discussed in Chapter 6.

5.3 Realisation of the Methodology

As we recall from Image 4.3, the role of a UML to performance model transformation methodology can be summarised in preparing a performance model representation that can be evaluated by a performance model evaluation tool. The output performance model produced by this methodology is a CTMC model. Literature reports many tools for solving and evaluating Markov chains, either by numerical solutions (i.e. calculating the equilibrium probability distribution), or by simulation for evaluating semi-Markov models. Examples of these tools are *Computer-Aided Rate Modelling and Simulation* (CARMS) [113], *Markov Analysis Software* (MKV)[114], *Symbolic Hierarchical Automated Reliability and Performance Evaluator*(SHARPE) [115] and *Markov Chain Analyzer*(MARCA)[116]. Most of these tools are dedicated to solving Markov chains (apart from commercial availability and reliability tools) and concentrate on calculating the equilibrium probabilities and transition rates between the states of the chain. This comes back to the variety of uses that Markov chains have in different QoS applications (i.e. performance, availability and reliability). The translation of the outcomes of the Markov chain analysis to useful performance measures was explained in Chapter 3. The use of one of the previous tools as a performance model evaluation tool for the produced performance model will require the user to calculate the performance measures from the produced probability distributions, as discussed in Chapter 3. This conflicts with the main objective of automating the performance evaluation study, which is closing the knowledge gap between software and performance engineering. This section will discuss a possible implementation for the methodology previously discussed in Section 5.2. This implementation prepares a performance model to be solved using the MARCA tool.

5.3.1 MARCA Package

MARCA is a software package designed to generate and determine mathematical properties of large Markov chain models [116]. The mathematical properties include stationary probability, transient distributions and mean time. This tool was developed by W. Stewart in FORTRAN. MARCA provides different means for representing the analysed Markov chain, which include a graphical representation for drawing the Markov network and text based interface for providing the Markov chain in the form of a transition matrix. This allows us to use this tool as the evaluation tool for our methodology. This can be done by writing a tool that will use the methodology in 5.2 to generate a text file that will include the transition matrix of the output performance model. This tool stores the transition matrix in a compact form which permits very large state spaces to be analysed. MARCA provides a wide selection of numerical solution methods for computing the stationary behaviour (i.e. stable direct solvers based on Gaussian elimination ,LU decomposition and single vector iterations (power, Gauss-Seidel, SOR, preconditioned power)) the tool provides a variety of techniques for computing the transient behaviour such as:

- Randomization
- Runge - Kutta
- Adams-Bashforth/Moulton
- Matrix powering for small systems (< 120 states)[116]

5.3.2 Performance Model Building Tool

The tool that we are suggesting to implement the methodology in 5.2 will build a Markov model solvable by the MARCA tool. The tool suggested will build the output performance model according to the state marking algorithm discussed above, in the form of a transition matrix. This transition matrix will include the probabilities and demands annotated in the sequence diagram. The MARCA tool manual provides a full specification of the formant of the text file, representing the transition matrix and other required information used in the analysis of the Markov chain model. The tool can use an XMI representation of the sequence diagram(s) representing the behaviour of the system under study, as the input document. The XMI representation of the model is a standard model exchange format in most of the UML modelling packages. An XSLT parser can be used to generate a text file representing the output model. XSLT (extensible Style sheet Language Transformations)[117] was developed by the

W3C organisation especially for transformations of XML documents. XSLT is a core technology for processing XML documents. The XSLT parser will be used to query the sequence diagram XMI document for the artefacts described in 5.2.3 and write the appropriate Markov chain representing this artefact. Once all the artefacts are complete, a complete Markov model can be generated by combining all individual Markov representations into a single Markov chain.

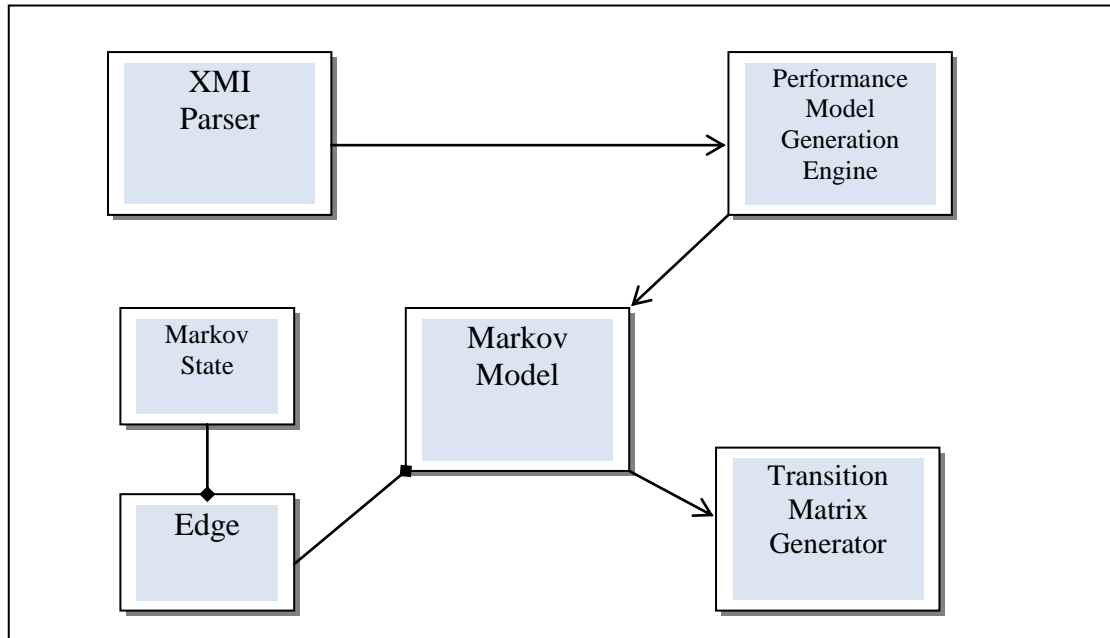


Figure 5.11: Components of a tool for the extended state marking methodology

A component diagram of the suggested tool is illustrated in Figure 5.11. The XMI parser will be used to extract the UML sequence diagram notations needed in the conversion process. This XMI parser can be any standard XML parser, either XSLT or a dedicated XMI parser written for this tool. The notations to be detected in the parsing process include:

- The Association messages, these messages can be detected by the message and association tags in the XMI document (See Chapter 2) and the type of message can be found in the attributes of the association.
- Interaction Frames can be detected by the fragment tag in the XMI document. The type of interaction will be declared in the type attribute.
- The Performance Annotations for the workload, probabilities and time demand.

The Performance Model Generation component is responsible for deploying the transformation method described in the extended methodology to generate a Markov model representation, stored in the form of states and edges between them. This Markov

model will be transformed to a transition matrix by the transition matrix generator component. This transition matrix will be formatted to be solved by the MARCA tool. This design of the suggested tool will give it the benefit of extendibility. The tool can be extended to another tool by developing a new *driver* for that tool. This driver is represented by the transition matrix generator component.

5.4 Summary

This chapter explained a methodology which was the result of the author's early studies in the field of performance evaluation automation techniques. The methodology explained in this chapter is based on the state marking methodology. The state marking methodology concentrates on capturing the behavioural aspects of the modelled system in a behaviour oriented performance model. The original state marking methodology proposed a method for extracting a GSPN performance model from a meta-model composed of collaboration and state-chart models. The limited generality of the GSPN and the non-standard input model used, motivated the extension of the state marking methodology.

The extended methodology proposes a systematic method for extracting Markov chain models from UML-SPT models. This chapter started by an explanation of the original state marking methodology, and how it was extended to increase its automation level to directly generate the reachability graph representing the GSPN. The methodology explain in this chapter was aiming to provide a syntactic algorithm that provide means for building a performance model from an annotated UML-diagram. Although this algorithm was systematic in a way that will allow the automation of this methodology, the simplicity of the deployment of this method and its tool is still an issue. This is caused by the employment of the UML-SPT as the input model. This model although provide a standard modelling notation, the representation of the performance data as tags and symbols would increase the ambiguity when conducting a performance experiment. This gave us an idea of changing the technique when collecting data for a performance study, which we will be explain in the next chapter.

In this chapter, we present a methodology dedicated to assisting software engineers in conducting performance studies from the early stages of the systems development life cycle. This methodology is called UML-EQN[9]. The UML-EQN methodology provides a systematic process for gathering performance parameters needed to build the performance model and converting the design model to an equivalent EQN (Extended Queuing Networks) performance model. This methodology was implemented in a tool called UML-JMT[11] which extends the JMT (Java Modelling Tool) suite [118] that will operate as its UML interface. This chapter is arranged in six sections. Section 6.1 will define the methodology's objectives and steps. Section 6.2 will discuss the first step of the UML-JMT methodology, which is the gathering of performance parameters; this step distinguishes this methodology from a lot of its rival methodologies. Section 6.3 will discuss the software model and the algorithm used in the building of the methodology. In Section 6.4, we explain the algorithms used to build the machine model which represents the base model for the end performance model. In Section 6.5, we explain the algorithms used to finalise the projected performance model. And finally in 6.6 we will evaluate the UML-EQN methodology using the criteria discussed in 4.2. During the explanation of each of the steps of the methodology, we will use an example of a video depository system (explained in 2.3.4) where we will study the performance indices of a suggested design for such a system, and compare it to an existing system.

6.1 Explaining the Methodology

The UML-EQN methodology is classified as a performance evaluation methodology, similar in its functionality to the methodologies discussed in 4.3. As we recall, these methodologies play the role of the performance verification test in performance engineering methodologies. The UML-EQN methodology is dedicated to assisting software engineers in deploying performance engineering methodologies throughout a system's life cycle. The ability of the methodology to work with different levels of abstractions allows this methodology to be deployed from an early stage of system development. The name of the methodology suggests the input and output models

involved. The methodology utilises UML structural and behavioural models in the process of building an equivalent Extended Queuing Networks (EQN) performance model. The methodology includes multiple steps that start with assisting the user in gathering performance data needed to build the model. The other steps involve multiple algorithms used to convert the UML models to an EQN performance model. The UML-EQN methodology (like all UML based methodologies) was designed with the objective of providing software engineers with a method for conducting performance verification tests required in performance engineering methodologies, with limited effect on the overall project budget. This can be done by allowing the software engineers themselves to perform the performance verification task. This is possible as the methodology was designed with a main objective of bridging the knowledge gap between software engineering and performance engineering. This gap is caused by the skills required to gather performance related information and the process of abstracting the systems' architecture and behavioural aspects in a suitable performance model.

As stated earlier, the UML-EQN methodology provides methods and algorithms that will assist the software engineers with limited knowledge in performance evaluation terminology, in the process of conducting a performance study, starting from the data gathering step and building the performance model from the UML design model. Another objective that was considered during the design of the methodology was to adopt a standard design model notation. This is why UML was chosen to represent the architectural and behavioural representation of the studied system. We have already discussed in Section 3.5 why the queuing networks are the "best" performance model for validating the performance of a software design. And we saw in 3.3.3 why the EQN provides a more general performance model because of the limitations of the analytical solution provided for product form queuing networks. One of the key objectives that was considered during the design of the methodology involved the methodology complying with one of the best known software performance engineering methodologies, PASA. The original PASA methodology suggested the use of SPE methodology for the performance evaluation task, but the use of non-standard behaviour models (Execution graph) could affect the deployment of this methodology (see 4.2.2). The UML-EQN takes advantage of the fact that, in SPE, software and machine models are separated, giving the analyst the ability to study different design alternatives. The methodology uses available system data at each stage of the design to construct an abstract performance model of the system. The level of abstraction and the accuracy of

the produced model will depend primarily on the stage of the design cycle where the model was constructed, in addition to the accuracy of the data used. Another objective of the UML-EQN methodology was for it to be *light-weighted*. By light weighted, we mean that the deployment of this specific methodology should be easy and with minimal resources. This is essential for it to be deployed in agile development performance engineering methodologies such as CPASA. Next we will provide summaries of the steps of the methodology, and the example we will use to explain it in the next sections of this chapter.

6.1.1 The Methodology Steps

The UML-EQN methodology is composed of four main steps. These are as follows: Performance data gathering, Software Model construction and Machine Model construction, then finally merging these models and transforming them according to an algorithm to produce an EQN performance model. Next we will summarise each of these steps:

The performance data gathering is the first step of deploying the UML-EQN methodology which should be adopted as a part of the requirements collection tasks. The methodology arranged the required data needed in the deployment of the methodology in what we called a performance data card (PDC). The PDC consists of information about the structural and behavioural aspects of the system under study. Also, it lists the required performance and workload characterisation expected from the system. As we explained earlier, these steps are intended to guide software engineers through the first step in software performance engineering. We will explain this step in Section 6.2.

Software Model Construction: The construction of the software model SM refers to the identification of the key scenarios of the software system. This involves defining the main use-cases in the system and their scenarios as use-case and sequence diagrams, and assigning performance measures gathered in the first step, such as the workload intensities and service demand on the resource requirement to the different scenarios. At the end of this step, a meta software model known as a *communication map* will be produced. A communication map is a probability graph representing the behavioural aspects of the system under study. This step will be further explained in Section 6.3.

Machine Software Model Constriction: The machine model MM is a basic model representing the components composing the system and their relation to the hardware platform. This model is based on Extended Queuing Networks EQN [6]. The building of the MM is dependent on the UML Deployment Diagram (DD). We will further explain this step in Section 6.4.

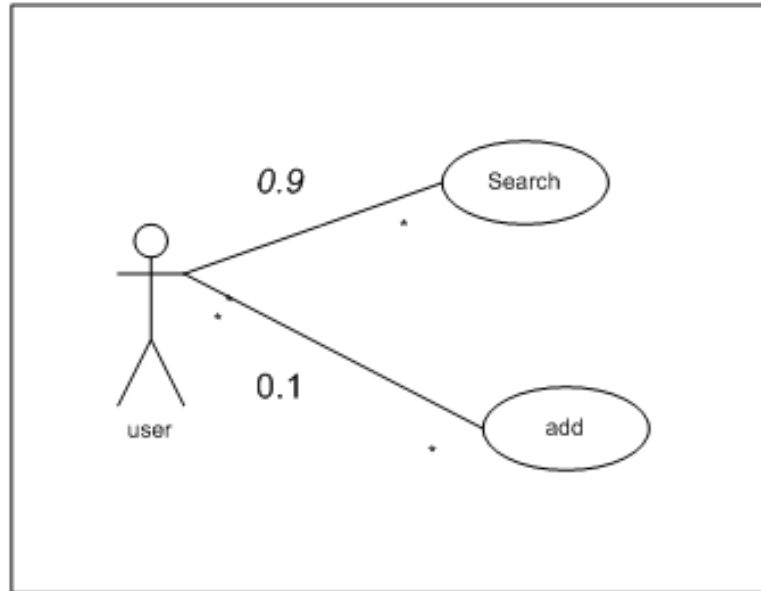


Figure 6.1 Annotated use-case diagram

Finalising the Performance Mode: At this stage, we have a SM representing the software as a communication map and a MM representing an initial view of the queuing network. The last step of constructing the performance model is to finalise the EQN model. This includes connecting the service centres of the MM according the communication maps, defining the QN job classes and routing these classes through the QN according the communication maps. This step will be explained further in Section 6.5.

6.1.2 Explanation Example

During the explanation of the UML-EQN methodology in the next four sections, we will explain the methodology with the aid of an example. This example is for the same video system discussed in Section 2.3.4. As we recall, this system will cache all clips previously stored or of interest to the user (according to his/her profile) when the network usage is idle. The main goal of the performance study is to compare the architectural alternatives for video streaming systems. We have suggested that architecture that allows caching related video clips in the user's station, which will decrease the time required to search and access video clips in future searches. The study

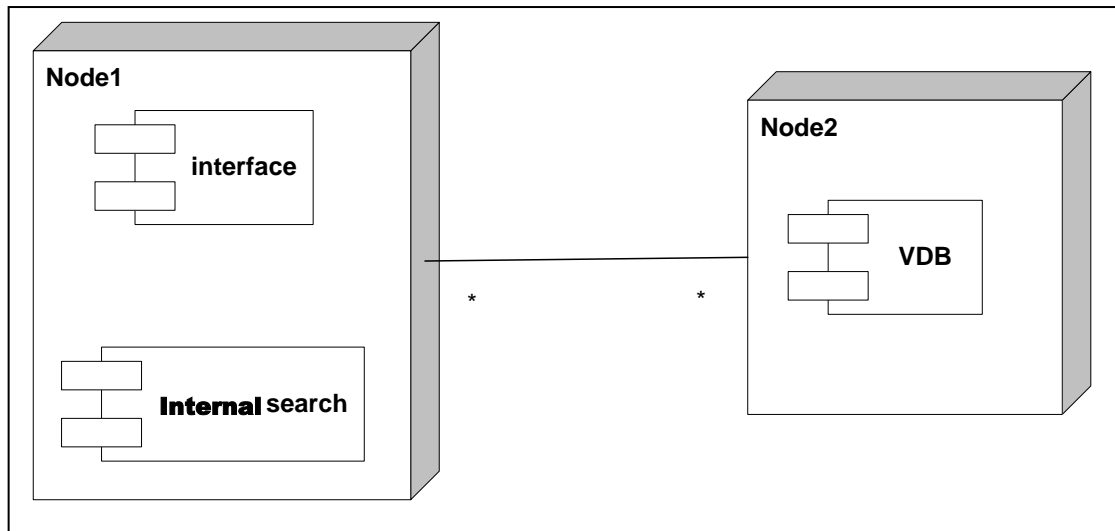


Figure 6.2. Deployment diagram of the video stream system architecture

will examine the response time in the suggested architecture and compare it to the average response time measured on a sample of video streaming systems.

Figure 6.1 displays the use-case diagram of the system. According to this diagram the video search system allows users to either add video clips or search for them. We assumed that 90% of requests to the system involve searching the video depository, whereas add requests represent only 10% of the workload on the system. The suggested system is composed of three main components; interface, internal search and video database (external). The connection between external and internal components is through the internet. The suggested architecture of the video stream system is shown in Figure 6.2, which illustrates the deployment diagram of the system. In this diagram, we

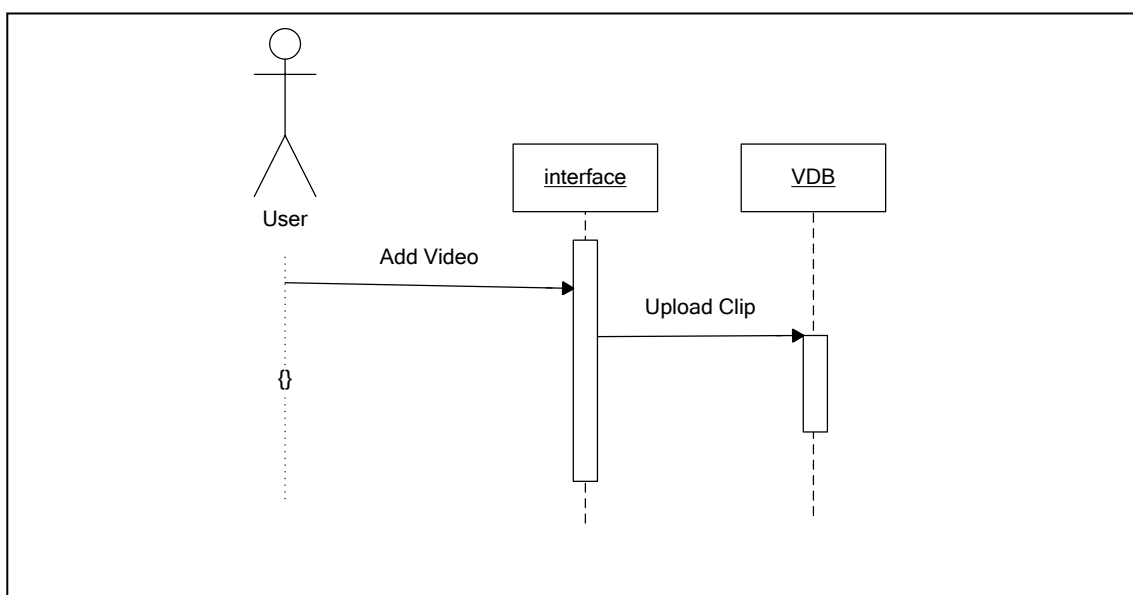


Figure 6.3. SD diagram of add Use-case

have two nodes representing the local user's station and the remote video database. The association between these nodes represents an internet connection.

As we explained earlier, the use-cases defining the behaviour of the system are the "add" and "search" use-cases. For each of these use-cases, we define the possible scenarios of behaviour. These scenarios are shown in Figures 6.3 and 6.4 as sequence diagrams. Figure 6.3 shows a possible scenario for the adding a video, and Figure 6.4 describes the scenarios for the search use-case. The two scenarios are for when the requested video is available locally, when it will be provided directly to the user, or, when it is not local, the video will then be searched for in a video database and, if found, played to the user.

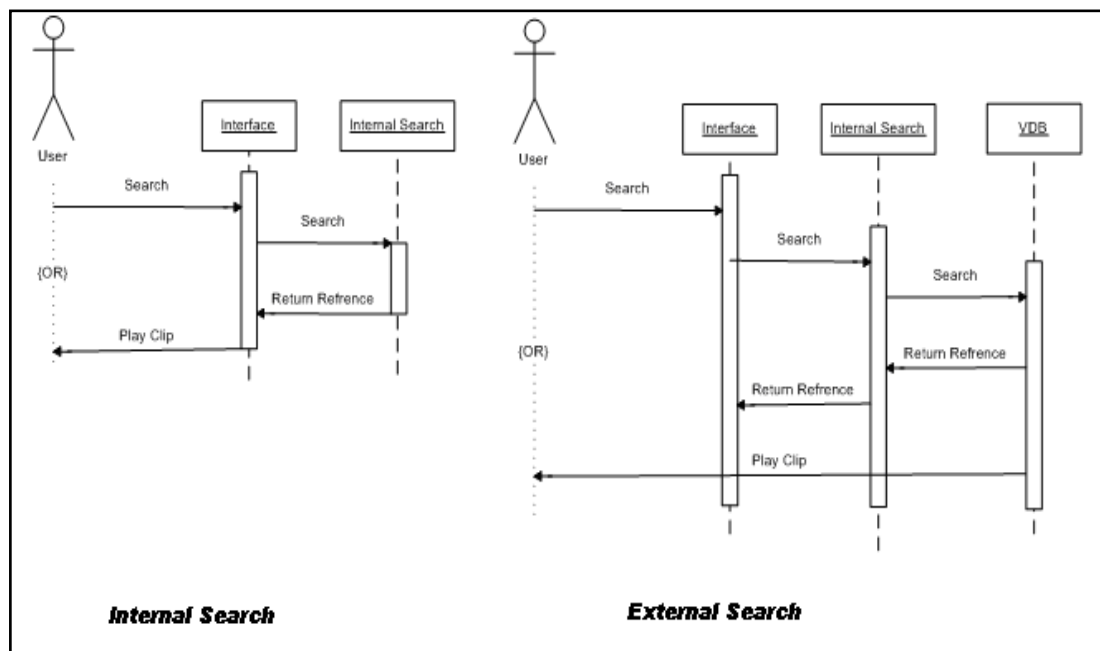


Figure 6.4 Sequence diagram of Search Use- case.

6.2 Performance Parameters Capture

One of the essential tasks in any performance engineering methodology is performance parameters capture task. The performance parameters are the parameters defining the performance critical architectural and behavioural aspects of the studied system, as well as the workloads, frequencies and resources demand defining the usage of this system. The performance parameters capture is known to be one of the most difficult tasks in software performance engineering. This is due to a number of reasons which include:

- The difficulty of defining the nature and source of these parameters without extensive knowledge in performance evaluation terminology.
- The difficulty of abstracting the software into the performance critical parts.

- The capturing and prediction of the performance parameters relating to workload and temporal data defining resource demand is difficult.

Most of the performance evaluation methodologies ignore the performance parameters capture support task. This will have an impact on the deployment of these methodologies as the user will have a vague view of the inputs to this methodology. Defining a set of clear performance parameters and a method that assists the user of the methodology in capturing these parameters is essential for strengthening the cost efficiency of the methodology.

6.2.1 Performance Parameter Required

The majority of software performance engineering frameworks describe a set of performance parameters, which are required in the performance analysis task. Williams and Smith have grouped these parameters in [119]. The categories that they provided were as follows:

- *Performance objectives*: Performance objectives describe quantitative criteria for evaluating the performance characteristics of the system under study. These objectives can be expressed as constraints on the performance characterisation (i.e. response time, throughput or resource usage) or as explicit performance tests (e.g. design validation or stress tests). We already discussed these performance characterisation indices and performance tests in chapter 3 of this thesis.
- *Workload specifications*: the workload specification is defined by the intensity of use for each use-cases representing the system. Usually in performance studies, the most used use-cases are the only ones taken into consideration. This return to the 80/20 rule that states that 20% of the use-cases, represent 80 % of the system load. Each of these use-cases are defined by a number of scenarios, the most frequent of these scenarios are called the *critical scenarios*. The workload specification is represented by the intensity of each of these critical scenarios. The workload intensity is determined by the rate at which these scenarios are executed. The intensity will depend mainly on the type of the system under study. The intensity of interactive systems can be articulated as the arrival rate that would trigger these scenarios or as the number of concurrent users and the amount of time between their requests. For real-time systems, the intensity is described in terms of the arrival rate of the events that activate and maintain the workload.
- *Software behaviours*: The software behaviour describes the software execution path(s) -*Scenarios*- for each use-case. The software behaviours should identify the

software components that involved in implementing this scenario, the order in which they execute, and any repetition, in addition to conditional and/or parallel execution of components for the corresponding workload.

- *Execution environment*: The execution environment describes the platform on which the proposed system will be executed. This environment consists of the hardware configuration, which will include the distribution of the software components on the hardware nodes, the internal configurations of these nodes (e.g. the processing power and operating system used) and the type of connectivity between these nodes.
- *Resource requirements*: Resource requirements approximate the amount of service time required from key components representing the system. Software scenarios specify resource requirements, in terms of the components visited during the execution of that scenario. *Service times* reflect the performance related characteristics of the execution environment. The service time is measured by calculating the average time required by the component in order for it to complete the service.
- *Processing overhead*: Performance related characteristics necessitate the inclusion of external/internal overhead processing that would have an impact on the overall performance. Such overheads include networks delays and users' thinking time. Overheads are treated as a software resource, in the sense that, the overhead specification would list resource requirement as the average time of this overhead.

As discussed before, one of the challenging parts of any performance studies lies in defining the performance requirement. This return mainly to the fact that different performance studies require different representations of the categories specified above. Currently, determining what information is required and the most appropriate way of expressing it requires expert judgment [119]. Table 6.1 summarises these categories in a requirement elicitation form. In this table, we have classified the categories as questions that will determine the type of information required and the source that should be consulted in order to answer this question. This table was the basis for the PDC used in the UML-EQN methodology. We have classified the systems to real time and information system, as the type of performance information depend on the system type. As we declared in the beginning, this thesis is only considering information systems.

Table 6.1: summary of the performance requirements categories, their source and rationale.

Question	Source of information	Rationale	System type	
			Information System	real-time systems
What is the system type?	System Specifications	Type of system will alter the performance specifications.	Information System or real-time systems	
What are the expected performance objectives?	NFR Specifications	Information needed to compare the expected performance specification of the understudy design with the requested quantitative specification in NFRs.	Response time: The number of seconds to respond to a user request.	Response time: The amount of time required to respond to a given external event.
			Throughput: Number of transactions to be processed per unit time.	Capacity Throughput: Refers to the number of events of a given type that the system must be able to process in a given amount of time.
			Resource usage: Expected utilisation on a specific resource.	Load Throughput: Refers to the number of events of (multiple) different types that must be processed in a given amount of time.
What are the components composing the system?	Class Diagram	To describe the static inter component communication.	Resource usage: Expected utilisation on a specific resource.	Resource usage: Expected utilisation on a specific resource.

workload	description	Sequence Diagram	The most frequent functions that the system performs determine the overall performance of the system, thus these functions need to be captured.		
	Intensity	User Req.	Specifies the rate at which each use of the system being modelled is requested.	Arrival rate for requests/ the number of concurrent users and the amount of time between their requests.	arrival rate of the events that trigger and sustain the workload
Resource Requirements		Workload description	Resource requirements estimate the amount of service required from key devices in the hardware configuration. Software plans typically specify resource requirements for processing steps in terms of the software resources.		

Table 6.2. List of important performance data required for model building (performance data requirement card)

Information	Source of information	Value
Performance objectives	NFR Specifications	Response time/ Throughput/ Resource usage or type of experiment.
System Components, functionalities (software)	Sequence Diagram, Use-case diagram	Actors/ use-cases/ scenarios/ components/ interactions
System Components (platform)	Deployment Diagram	Nodes/components/ intercommunication types
Workload	Use-case Diagram	Probabilities of the use of each functionality
	Sequence Diagram	Probability of use of each component
Resource Requirements	Deployment Diagram	Execution times/delays

6.2.2 Performance Data Card

We have composed a method for assisting the use of UML-EQN methodology in the first stage of deploying the methodology (performance parameters gathering). It is clear that performance gathering should be adopted as a part of the system's requirements collection phase of software development. Requirement gathering cards have been a common method used in collecting requirements and user stories in both conventional and agile development methodologies. We have therefore adopted a similar approach in assisting the user of the UML-EQN methodology in finding the required parameters for the deployment of the methodology and the source of these parameters. We arranged the required data in what we called a performance data card (PDC).

Table 6.2 gives a summary of the types of performance data that an analyst should look for. *Performance objectives* describe the expected performance measurements of the system which are needed to compare the predicted performance specification of the design under study, with the requested quantitative specification in the system's Non-

Functional Requirements (NFRs). This information is needed only if a QoS requirement is defined in the NFR specification. The *components involved* in the system and the connectivity of these components are important in the model building in order to describe the dynamic aspects of the system. This is represented by the UML diagrams. This is a scenario based methodology where the scenarios, defining the system behaviour, are used to construct the model (software model). Consequently we choose use-case and sequence diagrams as bases for extracting information about the system components and their connectivity. The platform design, on which the system rests, is represented by a deployment diagram. *Workload* defines the rates and distributions of each of the functionalities (in UML terms, use-cases) and the rates at which each of the components composing the system are invoked. *Resource requirements* estimate the amount of service required from key devices in the hardware configuration. This information can be taken from the UML deployment diagram along with the system specifications for components involved in the system [93].

6.2.3 Example

This section we describe the PDC for the example explained previously in 6.1.2. The PDC for that performance study is shown in table 6.3. As explained previously, a PDC starts with an objective of the performance study, which is explained in the first row of the table. The system's components, functionalities and the critical scenarios defining these functionalities are explained in the second row of the table. Important to any performance study is the expected workload for that system, which is defined here by the expected arrival rate of the search and add jobs for the system. The frequencies row –fourth row- explains the expected frequency for each of the functionality as a part of the coming jobs. We assumed in this example that 90% of the users will be searching the system and 10% will be adding new clips. For the *add* use-case, there is only a single scenario, representing a successful addition of a clip. Which means that 10% of the total workload will represent *add* scenario. On the other hand, the *search* use-case has two main scenarios which are internal and external. We assumed that each have a frequency of 60% and 40% of the total search respectively (i.e. we have a probability 60% to find clips locally). As the *search* use-case covers 90% of the total workload, we can calculate the individual workload for each of the scenarios covered in this use-case as shown in table 6.3. Another key entry necessary in the PCD is the resource requirement. The last row of the table contains the processing time required by each of the components to handle a jobs.

Table 6.3. PCD for the Video System discussed in 6.2.1

Information	Value	Description	
Performance objectives	Decrease response time/ utilise the internal search	The main goal of this system is to decrease the time required to conduct the search by utilising the internal search, as this system is an information system the response time will be measured in the time (sec) for the user to conduct a search/add. We will compare this against searching on YouTube where the average response time found in a small study by the author, was 8 seconds.	
System Components, functionalities	See Figures 6.1,6.2,6.3,and 6.4	The system is composed of three main components; interface, internal search and video database (external). The connection between external and internal components is through the internet.	
Arrival rates	7.26 jobs/second		
	Add	As this is the only scenario for add it will have a frequency of 1. By multiplying it by 0.1(as the add use-case is assumed to have 10% of the total number of operations), the frequency of this scenario is 0.1.	
Frequencies	Search internal	We assumed that 60% of the items being searched will be found internally. Taking into account the search/add ratio, this means that the internal search have a frequency of 0.54	
	Search. External	We assume that 40% of searches done by the users, the user will not find items in an internal search, which means that an external search is required. This means that the external search frequency is 0.36	
	Search	Assuming that 90% of time user search	
	add	Assuming that 10% of time user add new clips	
Resource requirements	<i>resource</i>	<i>type</i>	<i>Time(Avg.)</i>
	Interface	Process	0.3 sec
	Internal search	Process	0.5 sec
	External search	Process	0.9 sec
	internet	Network	5 sec

6.3 Constructing the Software Model

We have discussed previously that the UML-EQN methodology adopts the separation between the software and machine models in the process of extracting the EQN performance model. The software model is a meta-model used to define the behavioural aspects of the modelled system. The behaviour of a component based software system is represented by the possible communication routes between its components. The software model used in the UML-EQN methodology is known as *Communication Maps*. Each of these maps models the possible execution routes for each of the use-cases representing the functionalities of the system. These maps will be used to define the job classes in the final EQN model, and the routing of these jobs in the queuing network. The communication maps are constructed using an algorithm from the use-case and sequence diagrams, and the performance parameters gathered in the PDC. This section will explain the process of constructing the communication maps.

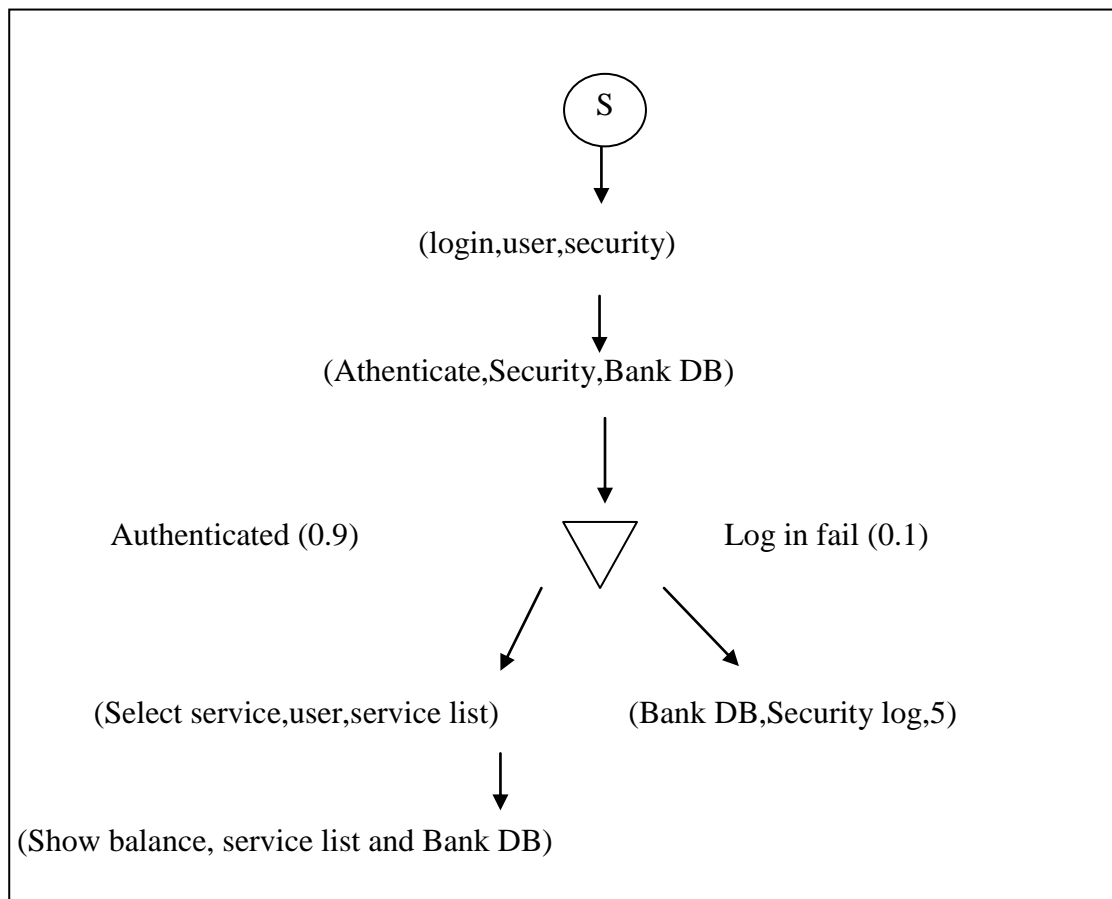


Figure 6.5. Communication map of a simple bank system.

6.3.1 Communication Maps

A communication map is a graph representing the behavioural communication between the components of a system for a specific functionality of the modelled system. This

involves defining the main use-cases in the system and their performance critical scenarios, and assigning them with performance parameters gathered in the first step, such as the workload and intensities of these scenarios. The communication maps are built from the use-case, defining the use-cases, and sequence diagrams representing the behaviour of the scenarios representing these use-cases. Each communication map will represent the behaviour of a single use-case, where each route in the communication map will represent a possible scenario of this use-case. The elements of the graph representing the communication maps are the messages representing the communications, which we call *demand vector*. A demand vector is a vector (n, A, B) that defines the name of this communication, n , the origin component A , and the goal component, B . Each of these vectors represents a transaction in the scenario of the functionality. The communication map representing a use-case is a reduction of all the routes representing the scenarios which define this use-case. The reduction algorithm is defined in the next subsection. The change in behaviour of the scenarios in a use-case is represented by a *probability split* separating the transaction route with the probability of executing this scenario. This can be calculated by multiplying the probability of executing the use-case by the frequency of the specified scenario. The probability split is represented graphically in the communication map as a triangle.

Figure 6.5 shows an example of a communication map for a simple online banking system. This communication map represents the *show balance* use-case. The two scenarios are for the user to login correctly and select the ‘show balance’ option from the service list or to have an incorrect login. After the two transactions of the login process we have the probability separator with the probability value separating the routes of processing. On the edges of this separator are the probability values of each route. In this example we assumed that only 10% of time users may log in incorrectly.

6.3.2 Communication Map Construction

The construction of the communication map starts with the identification of the performance critical scenarios for each of the use-cases that define the functionalities of the system. For each of these scenarios, we define the demand vectors. As these scenarios are represented as sequence diagrams, the demand vectors represent the transactions in the sequence diagrams with the name of this transaction as the name of the communication, the origin as the sending component, and the goal component as the receiver component. These demand vectors are chained together according to the order

of the transactions they represent. Other kinds of notation of the sequence diagram (conditional, loop, concurrent) are taken into account according to the following rules:

- *Conditional transactions*: Introduce a probability split separating the transaction route with the probability of each branch executing, to a branch representing the conditional transaction.
- *Loop transactions*: Make a probability split on the loop condition with the arc leading back to the top of the loop by adding the appropriate probabilities.
- *Concurrent transactions*: Introduce a fork/join communication with the probability of each branch executing to a true value.

As we explained before, each communication map will represent the behaviour of a single use-case, where each route in the communication map will represent a possible execution scenario. Therefore, we will need to reduce the different execution routes representing a use-case generated in the previous step, to a single communication map. This can be done according to the following rules of reduction:

- If transactions from different scenarios have the same demand vector, we reduce them to a single transaction assuming them to be representing the same function.
- If different transactions exist, we apply a *probability split* separating the transaction route with the probability of executing this scenario which can be calculated by multiplying the probability of executing the use-case by the frequency of the specified scenario.

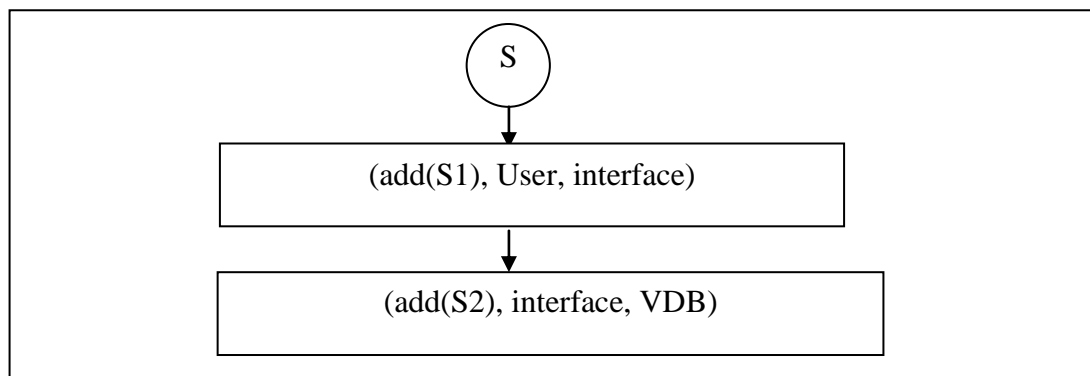


Figure 6.6. Communication map of use-case “add”

Applying this algorithm to generate the software model from the video system discussed in 6.1.2 will result the communication maps shown in Figures 6.6 and 6.7. In Figure 6.6, the communication map is applied for the “add” use-case. Figure 6.7 shows the communication map for the use-case “search”. As this use-case has two scenarios, with

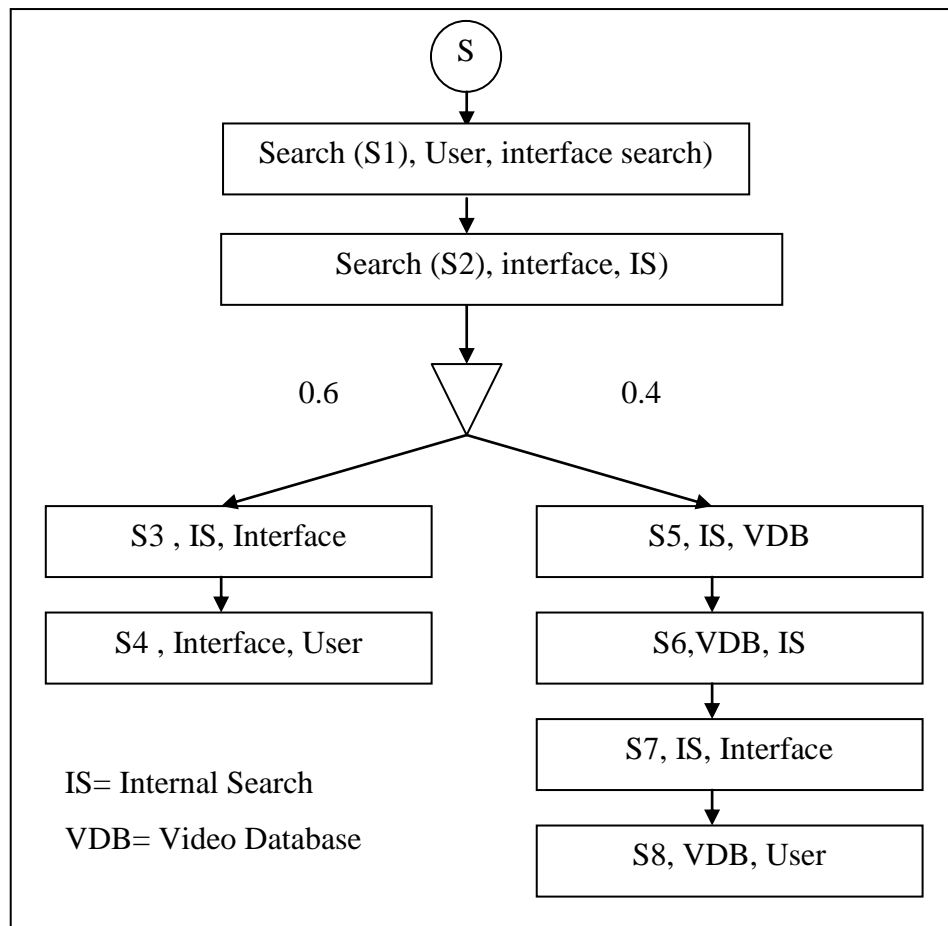


Figure 6.7. Communication map of use-case "Search"

different execution paths, after the search into the internal database, a probability split is administrated with the probabilities taken from the performance data card.

6.4 Constructing the Machine Model

The machine model MM is a basic model representing the components of the system and their relation to the underlying hardware platform. This model is based on an EQN and it represents the service centres and delays in the final performance model. This simple MM is usually exploited in early stages of the development life cycle where the analyst or designer has limited knowledge of the underlying hardware platform. The process concentrates on helping the designer to assess different architectural design alternatives in the early stages, and as the knowledge of the system increases, a more detailed model can be developed. The building of the MM is based on the UML deployment diagram model representing the architecture of the projected system. As we recall from 2.3.3, a deployment diagram defines the components in the underlying hardware platform and the topology of the connectivity between them, and that a deployment diagram is a set of interconnected nodes, where each of these nodes houses a set of components. In this methodology, we assume the nodes to represent the

hardware servers, and that each of these nodes houses a set of software services (represented as the components). The type of connection between these nodes (which is defined in the deployment diagram) will assist us in characterising the properties of the delay centres that will be used to simulate the network latency (i.e. the holding time in these delay centres).

The first step of constructing the MM is defining the type of network. The type of network (open/closed) depends mainly on the type of system and knowledge of the users. If the system is classified as closed, with a limited number of users requesting services from the system in continuous basis (a system servicing a department in an organisation), we choose the network to be closed. On the other hand, if the number of users is unknown (i.e. a web service system) we choose the network to be open. Adding the components that will classify the network include:

- *If* the queuing network is chosen to be *open*, we add source and sink stations that represent the origin and destination of all jobs entering the system. These stations will be used to calculate important performance measures, such as response time and throughput.
- *If* queuing network is chosen to be *closed*, we add a delay station representing the thinking time of the users in the network. The thinking time represents the average time between the users' requests for services from the system.

The process of constructing the MM components involves defining the service and delay centres. These represent the software service centres and simulation of communication overhead between them. The rules used to define those are as follows:

- Each component in each node defines a service centre. These service centres have some properties that need to be defined, such as the mean service time, maximum queue length etc.
- Each connection between the nodes defines a delay which depends on the type of connection between these nodes. Delay centres are infinite queues, which are used to simulate communication overhead.

By applying the previous rules to the deployment diagram in Figure 6.2 for the video search system, we will arrive at the basic MM shown in Figure 6.8. In this model, we have a source and sink stations which represent the origin and end of all jobs entering the system in open queuing networks. The service centres represent the three components (Interface, Internal DB and VDB), and the delay centres are found to

simulate the internet connection between interface, the VBD and the VDB and the interface components.

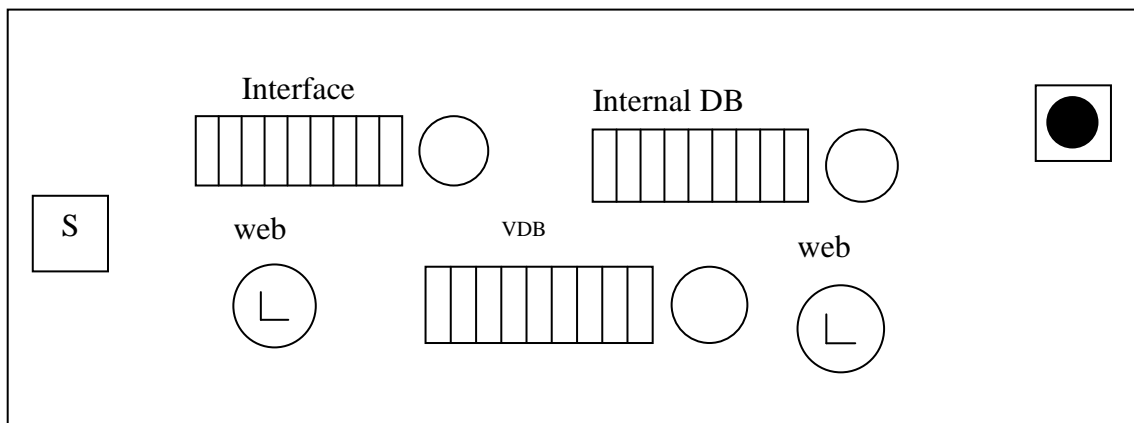


Figure 6.8: The components of the EQN representing the MM of the video system.

6.5 Finalising the Performance Model

At this step, we have an SM representing the software behaviour (inter-component connectivity) as a communication map and an MM representing an initial queuing network (the queuing network components). The inter-connectivity of the service centres is defined by the interactions between the components declared in the SM communication maps. The last stage of constructing the performance model is to finalise the EQN model. This includes:

- Defining the job classes of the queuing network
- Connecting the network according to the topology defined in the SM(s)
- Parameterising the performance characteristics of the queuing network

At the end of this stage, an EQN model is produced; this model is a non-product form network as it allows notations which are not allowed in product form queuing networks (i.e. fork/join) to be used. This section discusses the steps of finalising the performance model in detail.

6.5.1 Defining Job Classes

As described above, the end performance model is represented as a multi-class queuing network. Each of these classes will present one of the scenarios representing the overall behaviour of the system. Multi-class queuing networks are usually used to model systems with complex routing and varied performance characteristics. As we recall, this methodology is targeting software systems which can be obviously modelled as a multi-class queuing network, due to the assortment of behaviours a system could have, each with its own performance demand and characterisation. Therefore, the process of

defining the job classes for the potential performance model can be summarised in defining a job class for each of the scenarios representing the functionalities of the modelled system. The processes of defining the job classes and parameterising these job classes will be done according to this algorithm:

For each leaf node in a communication map(s):

- **Define** a job class named as the scenario they represent.
- Depending on the network type:
 - **If** the network is open: This class will have an arrival rate equal to the frequency of the communication route represented by the scenario defining this route.
 - **If** the network is closed: This class will have a number of users equal to the total number of users, multiplied by the frequency of this scenario.

6.5.2 Connecting the Network

The last step of the construction process of the performance model includes connecting the queuing network, routing the job classes and parameterising the service stations. Connecting the queuing network is done according to the following algorithm:

- **For** each communication map we start by making the connections between the components of the queuing network, according to the following rules:
 - **For** each demand vector in the communication map:
 - **If** the two components in the demand vector are within the same node, add a connection between the service centre representing these components *else* add the connection to the delay and then to the other component.
 - **If** there is a probability split, connect to each of the goal components with appropriate probability.
 - **If** there is a fork communication, add a fork station to the network.
 - **If** there is a join communication, add a join station to the network.

The calculation of the job routes for each of the job classes will depend on the routing of the scenarios representing these job classes in the communication maps. The routing of the job classes depends on the job's distribution probability in each of the service centres for jobs leaving each of the service centres. A problem may arise from jobs visiting a service centre more than once in a specific scenario. We avoided this problem by calculating the the probability of exit from a loop is the reciprocal of the average number of iterations of the loop. The algorithm for routing the job classes is as follows:

- **For** each of the service centres
 - **For** each job class:
 - **Set** the departing probability to 0 if the communication route for the scenario does not lead to a service centre connected to this service centre.
 - **Set** the departing probability according to the frequency calculated from the probability splits and the number of visits to this service centre.

The last step of constructing the performance model is to parameterise the service centres and delay stations with appropriate time demands gained from the PDC. The service time can be defined as a random variable defined as the exponential function $f(x) = \lambda e^{-\lambda x}$ where $1/\lambda$ is the average number of seconds the job spends in the service centre. This distribution will be used to simulate the time spent in the service centre or the delay station by each job.

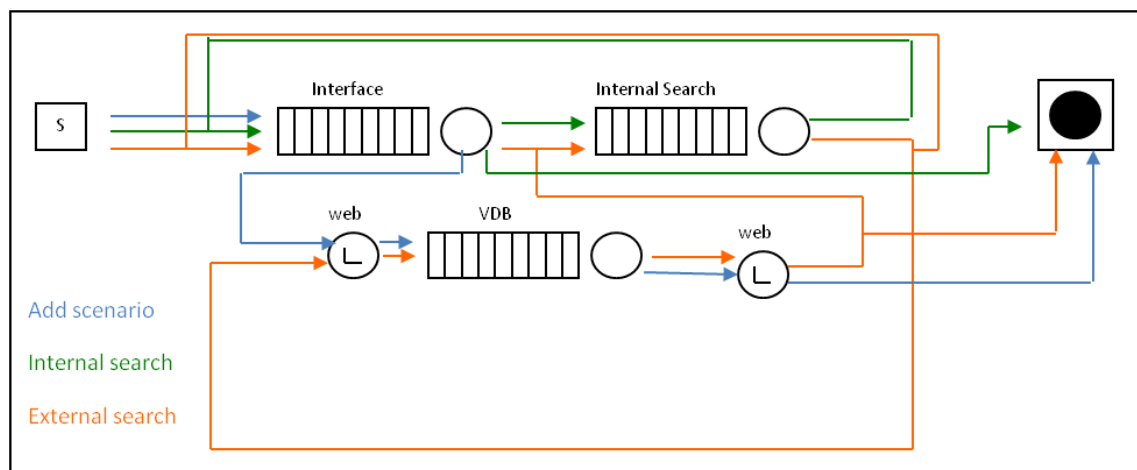


Figure 6.9: the resulting EQN with after applying the transition rules.

6.5.3 Example

Figure 6.9 shows the resulting EQN after applying the algorithms in Section 6.5.2 on the MM in Figure 6.8. From the possible communication maps represented in Figures 6.6 and 6.7, we define three possible job classes that we associate with the arrival rates in Table 6.3. The three possible communication routes represented in the communication maps correspond to the “*add*”, “*internal search*” and “*external search*” scenarios; in Figure 6.9, we differentiate them using three coloured routes. We set the arrival rates for each of the classes with the frequency values shown in Table 6.3, and for each of the service centres and the delay stations we define the required time to

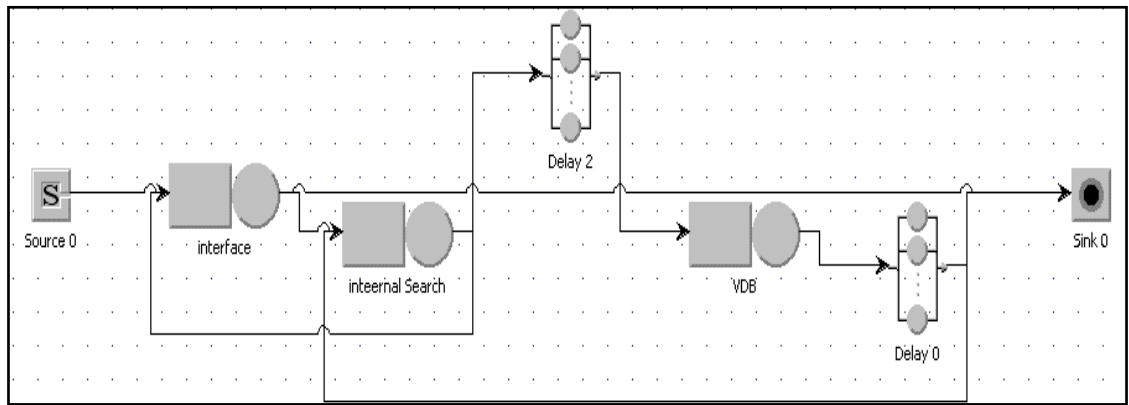


Figure 6.10. EQN obtained by introducing UML model of the video system to the UML-JMT tool

be spent in each of them, according to the performance data card resource values. The departure probability from a service centre is calculated according to the number of visits this job class makes to this specific queue. In the case of the internal search job class, the jobs visit the interface queue, then the internal search queue and then finally back to the interface. The jobs are routed from the interface to the internal search or the sink station. The probability is given as a 0.5 for each centre as the jobs are either new jobs or jobs returning from the internal search.

Table 6.4. Values of the system response time as the rate of job demand increase

	100%	122%	144.44%	166.67%	233.34%	255.56%	278%	300%
Avg (s)	6.157	6.074	5.719	6.287	7.013	7.392	7.959	8.993
Max(s)	6.663	6.37	6.122	6.758	7.414	7.925	8.604	9.388
Min (s)	5.652	5.779	5.317	5.815	6.613	6.859	7.314	8.597

To solve the model we used the UML-JMT tool which implements this methodology (Chapter 7) to translate the UML diagrams in XMI format into an EQN model. This model was designed to be solvable by a non-product form queuing network simulator included in a queue solving suite called the Java Modelling Tool (JMT). The resulting queuing network, as it is extracted from the tool, is shown in Figure 6.10.

The study of the system involved observing the response time as the job requests increased to 300% from 7.26 requests/second. Table 6.4, taken from the JMT tool, shows the effect on the response time as demand for jobs increases. The increase in the response time as the demand increases is shown in the graph of Figure 6.11.

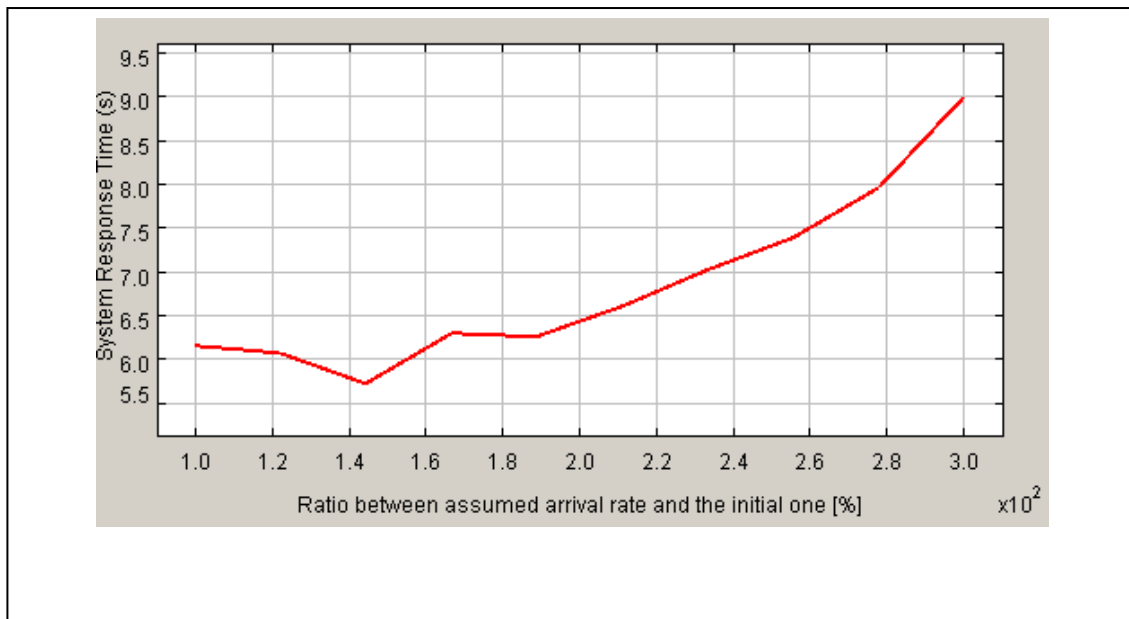


Figure 6.11. Chart for the effect of user demand on response time from the UML-JMT tool

6.6 Evaluating the Methodology

As we described in Section 4.2, the main objective of introducing performance model building methodologies is to reduce the cost of conducting the performance study tasks. In 4.3.1 we introduced a set of criteria which we assumed would have to be met in order for the proposed methodology to provide the user with the best assistant in the performance evaluation task. It is only fair to evaluate the UML-EQN methodology against these criteria, as we did in 4.3.2. As we recall from 4.3.1, the criteria that we set in order to evaluate the performance evaluation methodologies are:

Time Efficiency: The UML-EQN methodology was built to provide a cost efficient method for evaluating the performance of projected software systems to an acceptable degree of accuracy; we will discuss later, in Chapter 8 the validation of the resulting performance model produced by this methodology. During the build of this methodology, we chose the input and output to be employed and asserted that only standard modelling notations that will not require the user to spend more time learning new modelling notations, would be used. Furthermore, the meta-models used during the deployment of the methodology (software and machine models) were chosen to be simple or a subset of a standard modelling notation, although the tool implementing this methodology masks the user entirely from interacting with these meta-models.

The resulting performance model from this methodology was chosen to be easy to evaluate and this is why we chose EQN. It is clear from the steps of the methodology that the performance model produced was intended to be prepared for a specific EQN simulation tool. This tool is the JMT queuing network simulation suite[118]. This suite provides the user with a powerful and user friendly queuing network analysis and simulation tools. We will discuss further, the efficiency of the UML-EQN methodology when we consider the results of the trial of the UML-JMT tool, along with a sample of software engineers from the industry, in Chapter 9.

Generality: the resulting performance model from the UML-EQN methodology represented by EQN provides no limitation on the class of system architectures that can be modelled using this performance model. Thanks to the generic representation of the queuing networks to the component based software system, and the extra features provided by the EQN, the resulting performance model provides a comprehensive modelling notation that is capable of representing the most important architectural and behavioural features in most modern software systems.

Transparency: UML-EQN was developed with the aim of providing a methodology that not only assists the performance engineering, but can also assist in the reverse engineering process. The performance model represented by a multi-class EQN model is designed to be routed back to the original architectural and behaviour models used to construct it.

Automation: the methodology was implemented as a tool named UML-JMT. This tool will be discussed in detail in the next chapter.

6.7 Summary

The UML-EQN methodology is a methodology dedicated to assisting software engineers in conducting performance studies from the early stages of the systems development life cycle. The ability of the methodology to work with different levels of abstractions allows this methodology to be deployed from an early stage of system development. The methodology utilises UML structural and behavioural models in the process of building an equivalent Extended Queuing Networks (EQN) performance model. The methodology includes multiple steps that start with assisting the user in gathering performance data needed to build the model. The other steps involve multiple algorithms used to syntactically convert the UML models to an EQN performance

model. These systematic steps will help in achieving our main objective of bridging the knowledge gap between software engineering and performance engineering. This chapter discussed the UML-EQN objectives and steps. Each of the steps was discussed in details and explained with an example. We also evaluated the UML-EQN methodology using the criteria discussed in 4.2. Our evaluation of the methodology showed that the simple syntactic algorithms provided by this methodology for building the performance model increased its transparency and time efficiency. Also, the comprehensive output modelling paradigm produced by this methodology is capable of representing the most important architectural and behavioural features in most modern software systems; this in return was accounted in favour of the generality of this methodology.

**Realisation of the Method: UML-JMT
Tool**

The previous chapter discussed the UML-EQN methodology and how this methodology can be used to derive EQN performance models from a system's UML diagram. In this chapter, we will discuss the UML-JMT tool[11], a tool that implements the UML-EQN methodology. The UML-JMT tool was designed to work as a UML interface for the queuing network solving tools in the Java Modelling Tools suite JMT[118]. The UML-JMT Tools is a graphical user interface tool that will help users in building a performance model for their software system in a wizard like approach. The user will provide the tool with the performance data card entries in a question and answer approach. The tool will then use the UML-EQN conversion algorithms to construct a performance model based on the user entries. This model can be solved and analysed in a simulation based queuing network solver, provided by the JMT suite. This chapter provides a full technical specification of the UML-JMT tool.

Section 7.1 will discuss the design and implementation of the USDX XMI parser; which is a parser specially written for this tool. Section 7.2 will briefly describe the JMT suite, discussing its solving and analysis tools and how can it be extended by our tool. Section 7.3 will discuss the design of the UML-JMT tool by listing the key components that define this tool and explain the class diagram of this tool. Finally, Section 7.4 will describe how the design can become a reality by discussing the implementation aspects of the UML-JMT tool.

7.1 USDX Parser

The USDX parser (Use-case, Sequence and Deployment diagrams XMI parser) is a Java library developed specifically for the UML-JMT tool. It provides classes and operations that will help the analysis of UML models represented in XMI document. It is built on top of the javax DOM XML parser. This section will explain the design and functionalities of this parser. Section 7.1.1 will discuss the class diagram of the parser by explaining the classes that represent the UML model after the parsing operation. Section 7.1.2 will explain the model extraction algorithm for each of the UML

diagrams. The complete Java documentation for the USDX parser library can be found in Appendix A.

7.1.1 USDX Class Diagram

Figure 7.1 shows a class diagram of the USDX parser. The USDX parser consists of two main parts which are the XML document reader, represented by the file reader class, which is responsible for opening the XMI document file and making essential checks, such as the *well-formed* check and check with the XMI DTD. The file reader then generates a *document object*. The document object is generated by the javax DOM XML parser. This document object is passed back to the USDX class in order to be used in UML model extraction. The second part of the USDX parser is the UML structure represented in the UML Model class. The UML Model class is responsible for providing a structured and easy to use container for the UML model. Our assumption for this version of the parser is that there is only one UML model per XMI file.

According to our class definition, a UML model consists of a use-case diagram, a deployment diagram and a set of scenarios which implement the use-cases. In XMI specification, all the UML notations have an *ID* and a *name* as well as other attributes that we are not concerned with in this version of the parser and therefore, we will not include them in the extracted model. We have defined a super class named UML Notation; this will include all the common attributes and operations required by any UML Notation subclass (xmiID and Name and their getters and setters). The Use-case Diagram class contains a set of actors and a set of use-cases and the association between them. These are represented by lists of the sub-classes; Actor, Use-case and Association. The Deployment Diagram class includes a list of nodes, and the association between them this is represented by the two sub-classes:

- DDNode: Consists of a set of Component classes representing the components of that node.
- Association: Represents the connectivity between the nodes.

Sequence diagrams are represented by the class scenario. Class scenario contains the following attributes:

- Components: These are the interacting components in the sequence diagram. They are represented by a list of Component classes.
- Associations: The connectivity between the components is defined here by a list of Association classes.

- Messages: All the messages in the interaction are represented as a list of Message classes. Each Message class has a Sender and Receiver of type component.

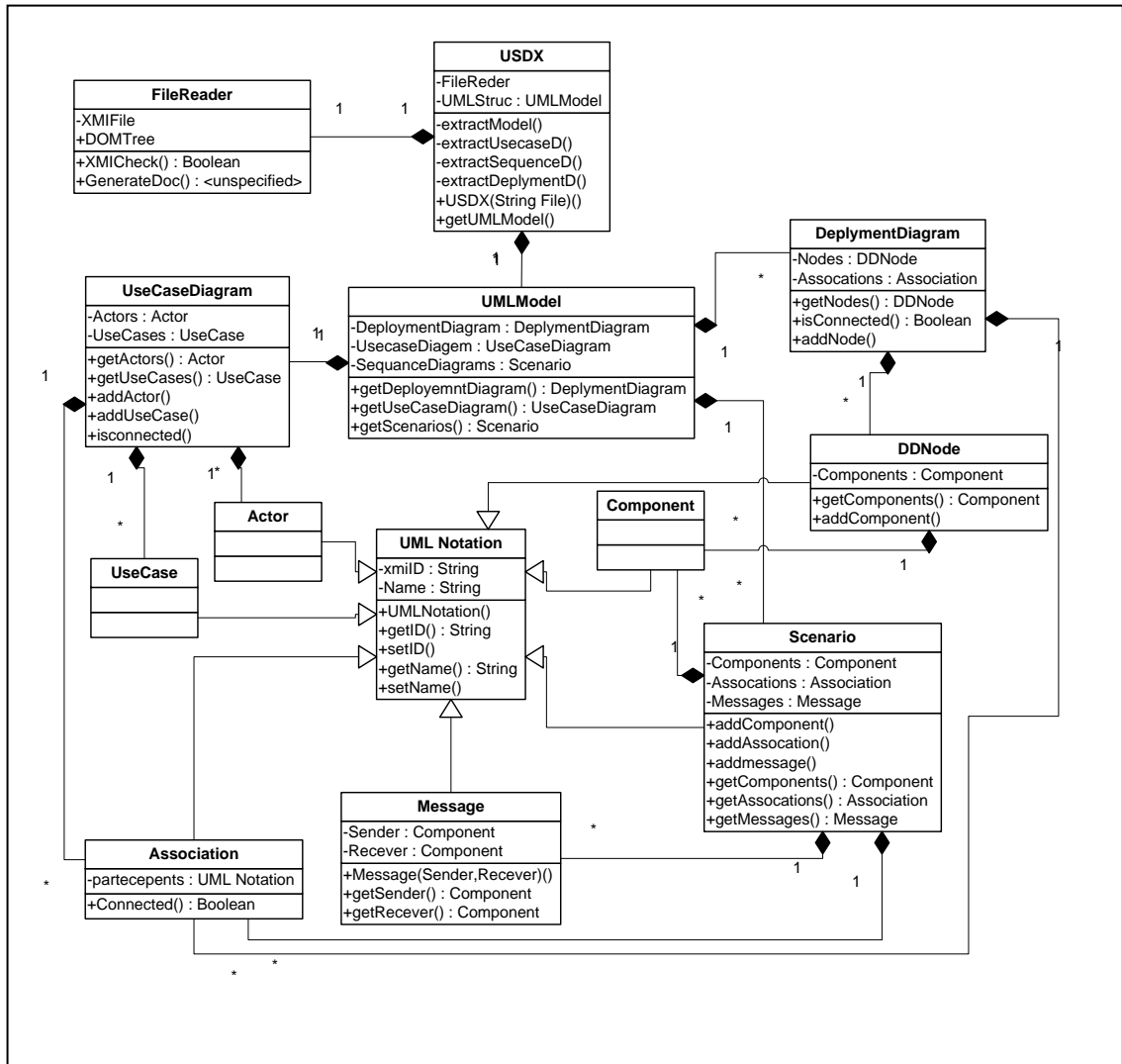


Figure 7.1: Class Diagram for the USDX Parser.

7.1.2 USDX Model Extraction Methods

The extraction and build of the UML model is done using the model extraction methods in the USDX class. These methods are derived from a method called an extract model which comes from the constructor of USDX. The constructor of the USDX parser is invoked with a string parameter representing the XMI document file name. This file name is passed to the file reader object which will return (if all checks are passed) an object of type org.w3c.dom.Document. This object will be used to traverse the DOM tree representing the XMI file and extract the UML model notation. Each diagram is extracted in a separate method named: extractUseCaseD(), extractDeploymentD() and extractSequenceD(). First we will briefly describe the Document class and the method

we are going to use from it. Next we will discuss the extraction algorithm for each of the UML diagrams.

DOM Document

The org.w3c.dom.Document interface is a java interface which represents a whole XML or HTML document. In practice, it references the root of the document which can be used to access the rest of the tree representation of the document; see Figure 2.9. In this representation, the tree is constructed from node classes which are used to encapsulate elements, attributes, text and comments etc. The methods that we are going to use from Document class are shown in the next table[120]:

getElementById (String elementId)	Returns the Element whose ID is given by the “elementId” parameter.
getElementsByTagName (String tagname)	Returns a node list of all the Elements with a given tag name in the order in which they are encountered in a pre-order traversal of the Document tree.

The methods we are going to use from node class are shown in the next table[120]:

NamedNodeMap: getAttributes ()	Return a “NamedNodeMap” containing the attributes of this node (if it is an Element) or null otherwise.
NodeList: getChildNodes ()	Returns a NodeList that contains all children of this node.
Node: getNextSibling ()	The node immediately following this node.
Node: getParentNode ()	Return the parent node of this node.
String: getNodeName ()	Return the name of the node.
Short: getNodeType ()	Return number representing the type of the node to be compared to constants.
String: getNodeValue ()	Return the value of the nude.
Boolean: hasChildNodes ()	Returns whether or not this node has any children.

Use-case Diagram Extraction

The extraction of the use-case diagram is split into three main steps which are as follows:

First, extracting the actors and creating the actors list this can be done using the following pseudo code:

```
NodeList : Actors ← Document.getElementByTagName("UML:Actor");
```

This will collect all the XML element in the tree with the name tag UML:Actor and store it in the node list named Actor. The problem will arise because the name tag actor is not only available when defining the “actors” notations but also in the association (see section 2.5.1). To solve this we will look for the elements that have an id attribute which means that they are being defined. This can be done as follows:

```
for (i=0; i<Actors.length(); i++)
{
    NodeMap Attributes ← Actors.item(i).getAttributes();
    Node N ← Attributes.getNamedItem("xmi.id");
    If (N!=null)
    {
        Create a new Actor object
        Get the name and xmi id using getNamedItem and set the name and
        Id for the Actor
        Add the Actor to the Actor list in the Usecase object
    }
}
```

Second, extracting the use-case using the same method used for extracting the actors, but with changing the tag name in the ‘get ElementByTagName’ method to “UseCase”.

Third, extracting the associations. Note that we have already extracted the actor and use-case elements in the associations using the ‘getElementByTagName’ method. For each of these elements, we traverse the parents until we find the parent named UML.Association and collect the name and ID from it. We then get the other UML notation connected to it by traversing this association tag until we find the other notation. This will be done by comparing the XMI id. The addition of the association found will be conditioned with the uniqueness of the association id in the list of

associations. Note that we assume that the associations are always between actors and use-cases. A pseudo code for this operation is as follows:

```

for (i=0;i<Actors.length();i++)
{
    NodeMap Attributes = Actors.item(i).getAttributes();
    Node N= Attributes.getNamedItem("xmi.idref");
    If (N!=null)
    {
        Node=Actors.item(i).getparentNode();
        While ((Node.getName()!="UML.Association") && (Node!=Document))
        {
            Node=Actors.item(i).getparentNode();
        }
        If (Node.getname()=="UML.Association")
        {
            o Get the id and name.
            o Check if the association already exist in the list by
              checking the id against the ids in the association list.
            o If it is new create new association.
            o Add one of its ends as Actors(i).
            o Find the other association by traversing the child nods
              until it is found.
            o Add association to the List
        }
    }
}

```

Deployment Diagram Extraction

The deployment diagram extraction operation involves the extraction of all the nodes in the document, then creating the objects defining these nodes from type DDNode by analysing the node element to extract the component information from them. The next step is to define the associations between the nodes using the same method used in the use-case diagram association extraction. The extraction of the nodes is done using the following line:

```
NodeList : Nodes ← Document.getElementByTagName("UML:Node");
```

At this step, all the elements with tag UML:Node are in the elements list named 'nodes'. As we recall, this includes the elements that defines the deployment diagram nodes and the ones defining the associations. These can be differentiated by the attributes in that element (i.e. if the attributes list includes name attributes or xmi.id attributes). Because the component elements are not direct children of the node element, as they are nested

in structuring elements, we need a method that will search for them down the children tree of the node element. The following ‘get Node’ method is a recursive method that will search in the children of the given node until it finds the child that has the tag name given in the parameter list, and return its parent. The pseudo code for the ‘getNode’ method is as follows:

```

Node:getNode(String name,Node N)
{
NodeList: Children ←N.getChildNodes();
for (int k ← 0; k < Children.getLength(); k++)
{
Node: aChild ← Children.item(k);
if (aChild.getNodeType() == Node.ELEMENT_NODE)
if (aChild.getNodeName() == name) return aChild.getParentNode();
else
if (aChild.hasChildNodes()) return getNode(name, aChild);
}
return null;
}

```

The pseudo code for extracting the nodes and adding them to the node list is as follows:

```

for (i ← 0; i < Nodes.length(); i++)
{
NodeMap:Attributes ← Nodes.item(i).getAttributes();
Node:N ← Attributes.getNamedItem("xmi.id");
If (N != null)
{
DDNode:N ← new DDNode(theAttribute.getNodeValue());
N.setName(attributes.getNamedItem("name").getNodeValue());
Node: childParent ← getNods("UML:Component", Nodes.item(i));
NodeList:Components ← childParent.getChildNodes();
for (j ← 0; j < Components.length(); j++)
{
if (Components.item[j].getNodeName() == "UML:Component")
{
if ((Components.item[j].getNodeName() == "UML:Component"))
{
NamedNodeMap:attr ← Components.item[j].getAttributes();
Component:C ← new Component(attr["Name"]);
}
}
}
}
}

```

```

        C.setID(attr["xmi.id"]);
        N.AddComponent(C);
    }
}
}
}

```

- Add Node to the List of Nodes in the Deployment Diagram

```

}

```

Sequence Diagram Extraction

Extraction of sequence diagrams from an XMI document includes extracting all the scenarios elements tagged with the name "UML:Collaboration". Unlike the case diagram and the deployment diagram, all of the elements related to the sequence diagram are inside this element. Therefore, after the extraction of the sequence diagrams nodes using this line:

```

NodeList:Scenarios←Document.getElementByTagName("UML:Collaboration");

```

We will be working with the scenario nodes to extract the components collaborating in this scenario, associations between them and the messages defining the interactions in this scenario. The pseudo code for extracting the scenarios and adding them to the sequence diagrams list is as follows:

```

for (i←0; i< Scenarios.length(); i++)
{
// create a Scenario Object and set its name and xmi.id
NodeMap:Attributes ← Nodes.item(i).getAttributes();
Node:N← Attributes.getNamedItem("xmi.id");
Scenario:S←new Scenario(theAttribute.getNodeValue());
S.setName(attributes.getNamedItem("name").getNodeValue());

//find all the components in the sequence diagram
Node: childParent← getNods("UML:ClassifierRole", Scenario.item(i));
NodeList:Components←childParent.getChildNodes();
for (j←0; j< Components.length(); j++)
{
    if (Components.item[j].getNodeName()=="UML:ClassifierRole")
    {
        if ((Components.item[j].getNodeName()=="UML:ClassifierRole"))
        {
            NamedNodeMap:attr← Components.item[j].getAttributes();

```

```

// check if the element is a new component or an association
Node:IDN← attr.getNameItem("xmi.id");
if (IDN!=null)
{
// create a component object and set its name and xmi.id
Component:C←new Component (attr["xmi.id"]);
C.setName (attr["Name"]);
S.AddComponent (C);
}
}
}

```

- We will find and add associations using the same method we used for use-case and deployment diagrams as the association involved elements are already in the list of components extracted earlier. Note that we will check the parent of the Association participant first to be positive it is not in a message element.

```

// find all the messages elements
Node: childParent← getNods("UML:Message", Scenario.item(i));
NodeList:Messages←childParent.getChildNodes();

```

```

for(j←0;i< Messages.length();j++)
{
if (Messages.item[j].getNodeName()=="UML:Message")
{
NamedNodeMap:attr← Messages.item[j].getAttributes();
// create a Message object and set its name and xmi.id
Message:M←new Message (attr["xmi.id"]);
M.setName (attr["name"]);

```

- Find the sending and receiving by their id and set the sender and receiver in the M object.
S.AddMessage (M);

```

}
}
}
• Add S to the List of Sequence Diagrams list.
}

```

7.2 JMT Suite

The performance model generated by the UML-JMT tool is structured to be solved and analysed by the queuing network solution and analysis tools provided by the JMT suite. The Java Modelling Tools (JMT)[81] suite is a free, open source suite that consists of a

number of performance evaluation tools. The suite provides different tools that offer analytical and simulation solutions for the queuing networks. Among these tools are functionalities that will help the user to perform analysis experiments on individual performance indices, with different control variables[81]. As we recall from Section 3.6, one of the main reasons for choosing this tool to expand was the fact that is built on an XML data layer; that is, all the models provided to this suite are structured in an XML document format. In this section, we will discuss the tools that provide solutions and analysis for queuing networks and describe how these queuing networks are represented in the suite's tools. Section 7.2.1 will discuss the *JSIMgraph*, the tool that we will use to solve the queuing network. Section 7.2.2 will describe the queuing analysis tools available in the JMT suite, and finally, in 7.2.3 we will outline the structure of the XML file that will contain the performance model.

7.2.1 Queuing Network Solution Tools

The JMT suite provides two main methods for solving a queuing network; analytically or through simulation. The analytical solution provided by the *JMVA* tool provides the detailed analysis of product-form queuing networks through a stabilised version of the MVA algorithm[81]. The simulation solution is provided by a discrete event simulator for the analysis of queuing networks called *JSIM*. The *JSIM* supports several probability distributions for characterising service and inter arrival times, as well as different routing strategies[86]. JMT suite provides simulation solution through two tools, the *JSIMwiz* which is a wizard interface for the *JSIM* simulator, and the *JSIMgraph*. The *JSIMgraph* is a graphical user interface tool that provides a workbench that allows the user to design and edit a queuing network model. As the model generated by the UML-EQN methodology may include non-product-form aspects (i.e. fork and join), we choose to use the *JSIM* simulator as the main queuing network solver. The model produced by the UML-JMT tool is configured to be opened by the *JSIMgraph*, where the user can adjust the model and manage the performance analysis experiment. A great feature of the *JSIMgraph* tool is that it can open a model designed inside it in the *JMVA* tool, provided that this queuing network is a product-form queuing network. This means that if the model produced by the UML-JMT tool does not have any non-product-form aspects, this model can be solved either analytically or by simulation.

7.2.2 Queuing Network Analysis Tools

The JMT queuing network analysis tools provide a set of analysis functionality that will help the user of the tool to study the performance indices of the system. These

functionalities can be working on a fixed set of input parameters or a variable control parameter. An analysis tool available in the JMT suite, called the *What-if* analysis tool, allows the user to set one or more control parameters (can be the number of users, workload ... etc), and the tool will evaluate the performance model for the performance indices that the user selected along the ranges and that they selected for this control variable. This will allow the user to observe the change in the system behaviour as the conditions around the system change. The JMT suite provides different performance indices for the user, such as throughput, utilisation and respond time. These can be for the entire system or a specific station or job class. Other performance indices, which describe the performance of specific stations, include queue length, queue time, residence time, response time and utilisation. The reader can refer to the JMT suite user manual[87] for more information about the tool's analysis functionalities.

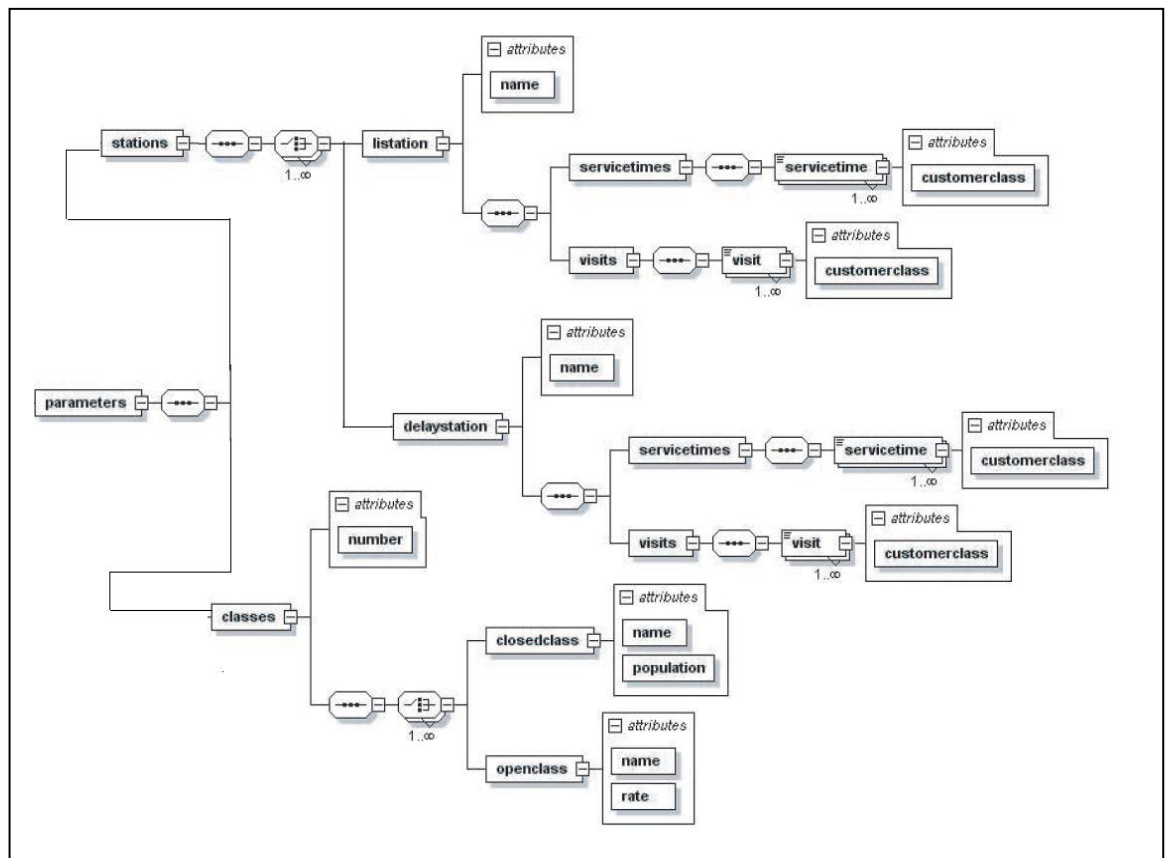


Figure 7.2: XML file schema for the performance model in JMT suite [2]

7.2.3 Queuing Network Representation

As we mentioned earlier, the performance models in the JMT suite are saved in XML format. An extracted part of the XML schema that represents the performance model design is shown in Figure 7.2. As the Figure shows, the performance model is presented as a set of *stations* and a declaration of the *user classes*. The stations represent both the

service stations and the delay stations. Each station contains a queuing part and a service part. The queuing part defines the specification of the queue, such as the queue's maximum length, drop strategy ... etc. The service part includes information about the service specifications such as the service time, number of services etc. The difference between the service station and a delay lies in the queue length, as the delays have unlimited number of servers by default. The job class declaration defines an element for each job class with its source station and name, and number of customers or workload, depending on the type of network. Other elements not shown in Figure 7.2 include the *sink/source* elements in an open network which define the start and end stations for each job and *connection* elements which define the connectivity between the different elements in the network.

7.3 UML-JMT Tool Design

We saw in Section 2.5.1 of this thesis how a UML diagram is represented in an XMI document. The previous section showed how we used a special XML parser to represent the UML models' notations as a Java UML Model object. We also talked about the JMT performance model solver and its analysis tools. This information was essential to discuss the design of the UML-JMT tool. This section will explain the main components composing the UML-JMT tool and how these components interact with each other in order to generate the EQN performance model. In 7.3.1 we will provide details of these main components by defining them and explaining their responsibility. Section 7.3.2 will explain the structure model of the tool by explaining the class diagram of the UML-JMT tool. The behaviour model of the tool will be explained in 7.3.3 by discussing the activity diagram representing the tool's behaviour. In 7.3.4 we will explain how the UML-JMT tool can be integrated with the JMT suite and how they can both be a part of the design model in the performance model framework described in Chapter 4.

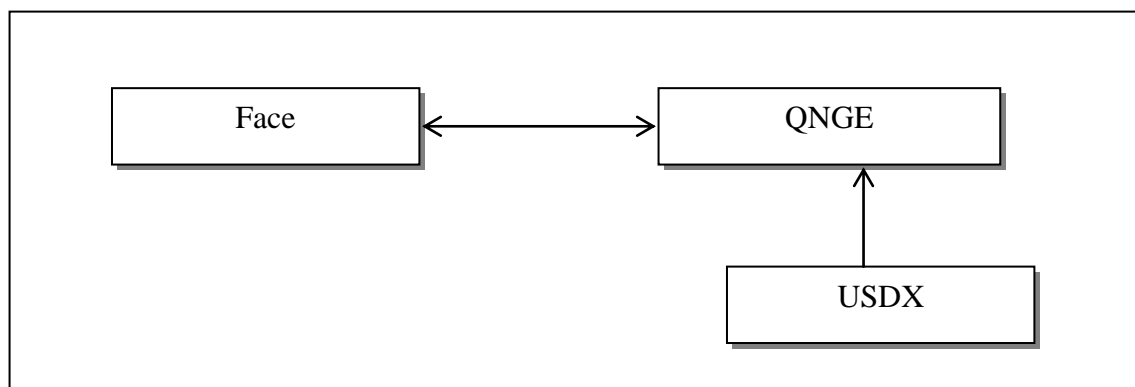


Figure 7.3: UML-JMT Components

7.3.1 UML-JMT: Components

To give the tool the advantage of extendibility and maintainability, we have adopted a component oriented design for the UML-JMT tool. If we want to view the UML-JMT as a set of interacting components, an abstract component oriented view of the tool can be seen in Figure 7.3. The tool is composed of three main components which are: Face, QNGE (Queuing Network Generator) and USDX. The Face represents the main interface of the tool; it defines the wizard responsible for gathering the UML model and performance data from the user (implementing the performance data gathering task). The QNGE represents the main model converting engine. These two components pass information to each other, relating to the UML model and the performance data. The QNGE takes advantage of the USDX parser, mentioned previously, to generate an object representation of the UML model. This model will be analysed by the QNGE to prepare the performance data card that will be queried in the UML-JMT interface to be filled by the user. The QNGE will also generate the communication map used in the UML-EQN methodology. It will also define the delay centres that will simulate the communication delays. Table 7.1 explains the Rationale of each of these components.

Table 7.1: the main components composing the UML-JMT tool explanation

Component	Name	Rationale
Face	Interface component	The main interface of the UML-JMT tool:
		<ul style="list-style-type: none"> • Implements the performance data collection of the UML-EQN methodology, • Collect input/output files names, • Collect the performance data, • Launch model generator.
QNGE	Queuing	The model conversion engine:
	Network	
	Generator Engine	
		<ul style="list-style-type: none"> • Starts the model extractor • Make the assumption checks • Generate and write the model according to users requirements
USDX	XMI parser	It will be used to extract the UML model from the input XMI file(see 7.1)

7.3.2 UML-JMT: Class Diagram

Figure 7.4 illustrates the class diagram of the UML-JMT tool. The UML-JMT class is the main class that launches the program. This can be done by creating and starting the wizard implementation defined in the class GUI. The GUI class implements a graphical

user interface wizard that will guide the user, step by step, in the generation of the performance model. It will do this by first asking the user for the input UML model XMI document. This document will be passed to the to a model conversion engine. This engine will make a number of checks on the input document; these checks are for the structure of the XMI document and the assumptions of the UML-EQN methodology. If the document passes the test, the GUI class will receive the names of use-cases, scenarios and delays from the conversion engine, and use this information in the creation of the performance data card. The performance data card will be passed back to the conversion engine in order to generate the performance model. The GUI class represents the Face component in the component representation of the tool. The USDX class represents the XMI parser we discussed earlier and also represents the USDX component.

The QNGR component is represented by a set of classes; the main class that implements the functionality of the QNGR component is the QNGen class. The other classes are used to represent the EQN components; these classes are as follows:

1. **Service Centre**: As its name suggests, this class will represent service centres in the queuing network. Each service centre is composed of a server and a queue. The server defines the number of servers in this service centre and the service time. The queue defines the maximum number of waiting jobs the queue can hold (will be -1 if the queue is infinite), and the drop strategy. This class has getter for its fields and “writeCS” method which will write the XML element representing this service centre, according to the JMT structure.
2. **Delay**: A delay represents an infinite queue that will hold jobs for a specific length of time. These will be used to simulate communication delays and thinking time in the performance model. The delay class contains a list of the involved service centres (i.e. the delay centres that will connect through this delay), delay time and the queue specification. The delay methods include the method write delay which writes the XML element that will define this delay.
3. **ComMap**: This class represents the communication map used in the UML-EQN methodology to route the communication between the service centres.

The QNGen will use the algorithms defined in Chapter 6 to generate the service centres, delays and communication maps. The method convert of the QNGen class will start the process of writing the performance model XML file. This will be done by calling the

preparation methods that will print the header and footer of the document, define the job classes and the type of network. Next, the service centres and delay centres will be added to the document by invoking their right method. The communication between the service centres will be defined by the communication map ‘getDestinations’ method. We will explain this process in more detail in the next section. The PDC class defines the Performance Data Card; it will be used to pass data on the UML model to the GUI, and the performance data gathered from the user to the queuing network generator engine.

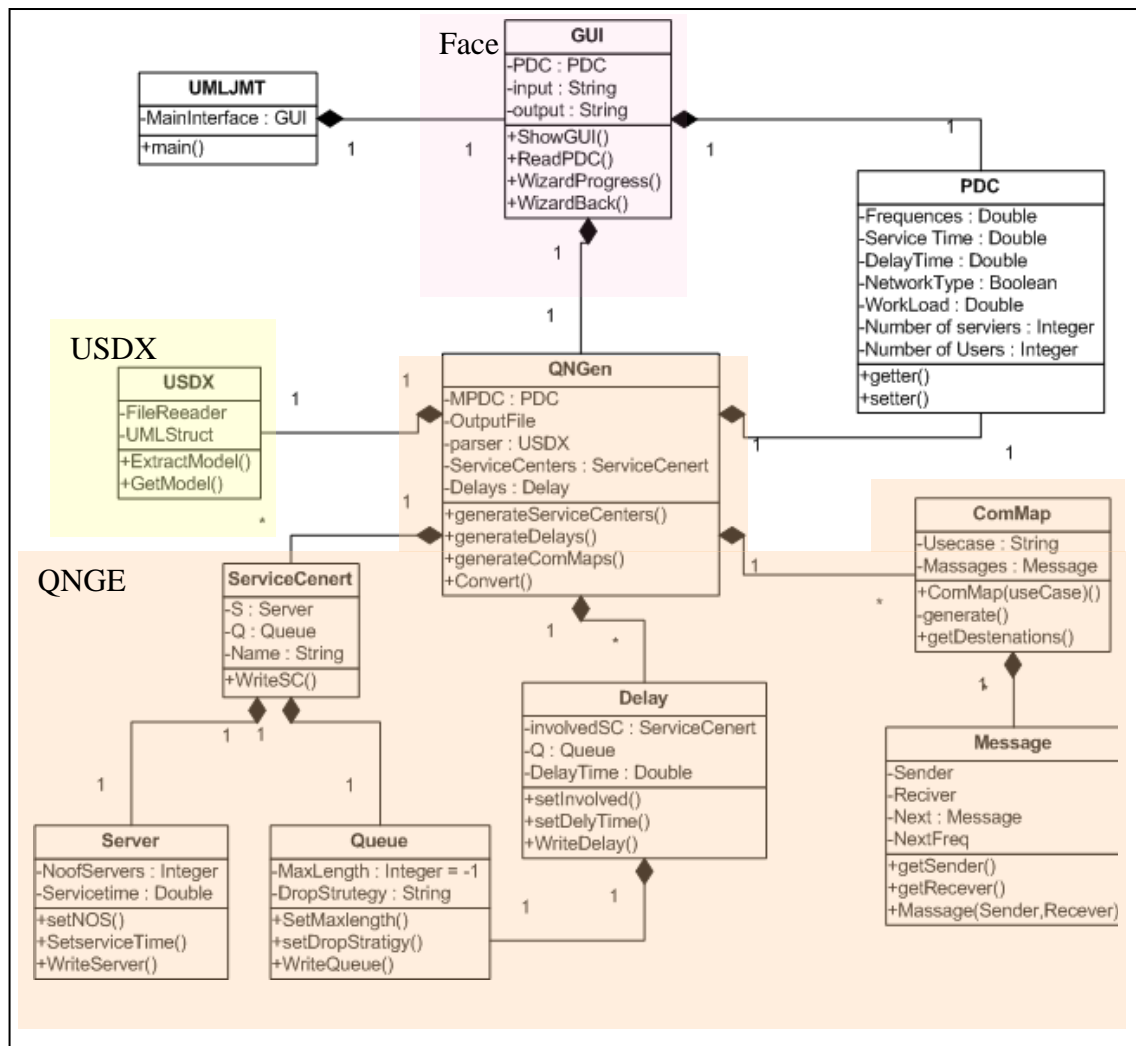


Figure 7.4: Class Diagram for the UML-JMT tool.

7.3.3 UML-JMT: Activity Diagram

An activity diagram showing the process of interaction between the UML-JMT tool components is shown in Figure 7.5. The GUI will ask the user for the file name of the XMI document containing the UML model. This file name will be passed to the model generation engine where a USDX parser object will be created. The parser will be given the file name of the XMI document where it will conduct a check on the file structure

and content (check the notations available in the XMI document). If the document

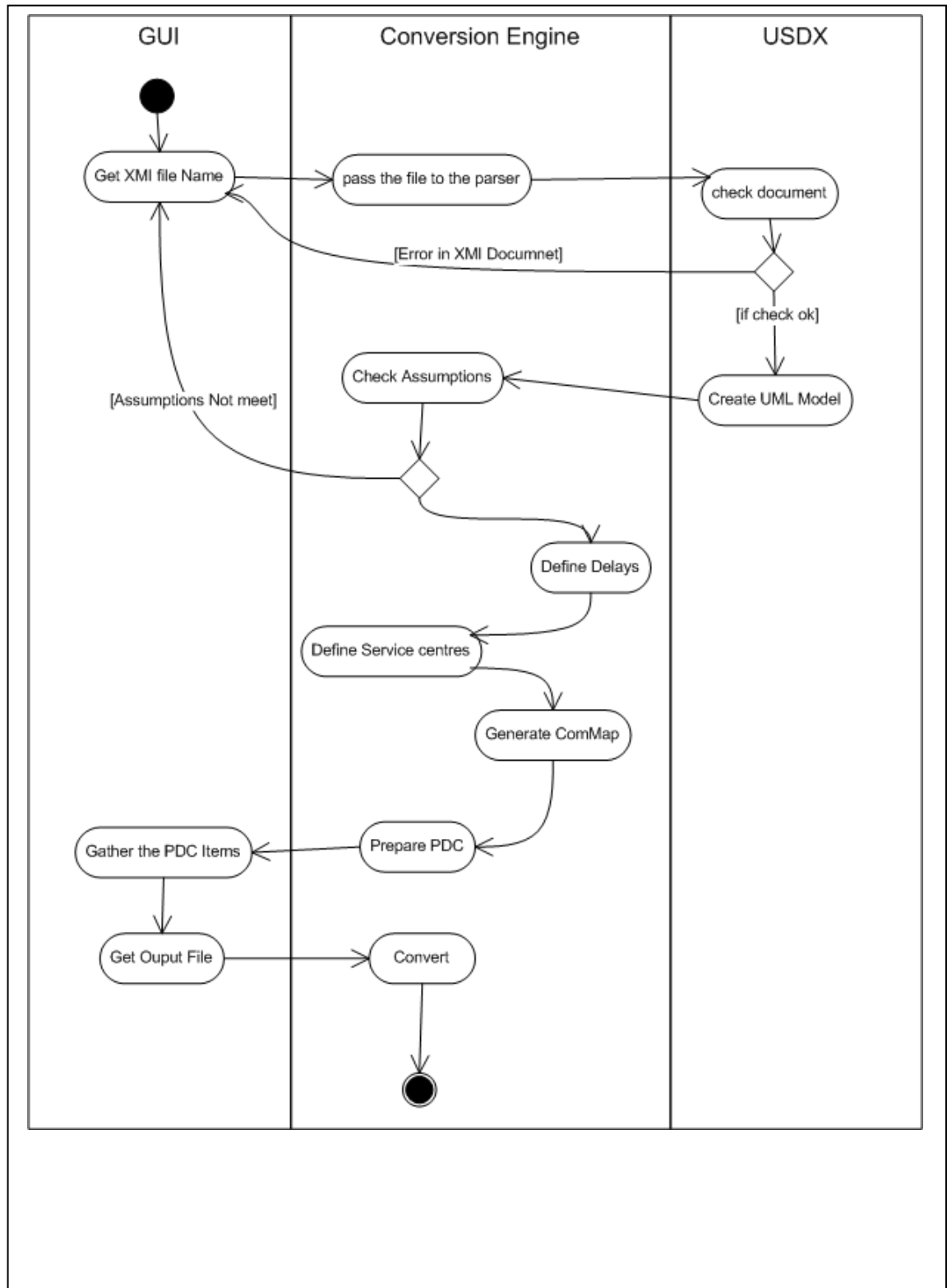


Figure 7.5: Activity Diagram for the UML-JMT tool.

passes the checks, a UML model object will be created and if not, the user will be given an error message and asked to supply another document. If the UML model is created, this model will be checked for the methodology assumptions explained in the previous

chapter. If the assumptions are not met, the user will be notified and asked to enter a new file. Otherwise, the conversion process will start. The delays, service centres and communication maps will be generated from the UML model and the performance data card required from the user will be prepared. This performance data card will be passed to the GUI where it will be requested from the user in an interactive way. After the performance data is gathered from the user, it will be passed back to the conversion engine where service centre objects and delay objects are updated, and the convert method is invoked.

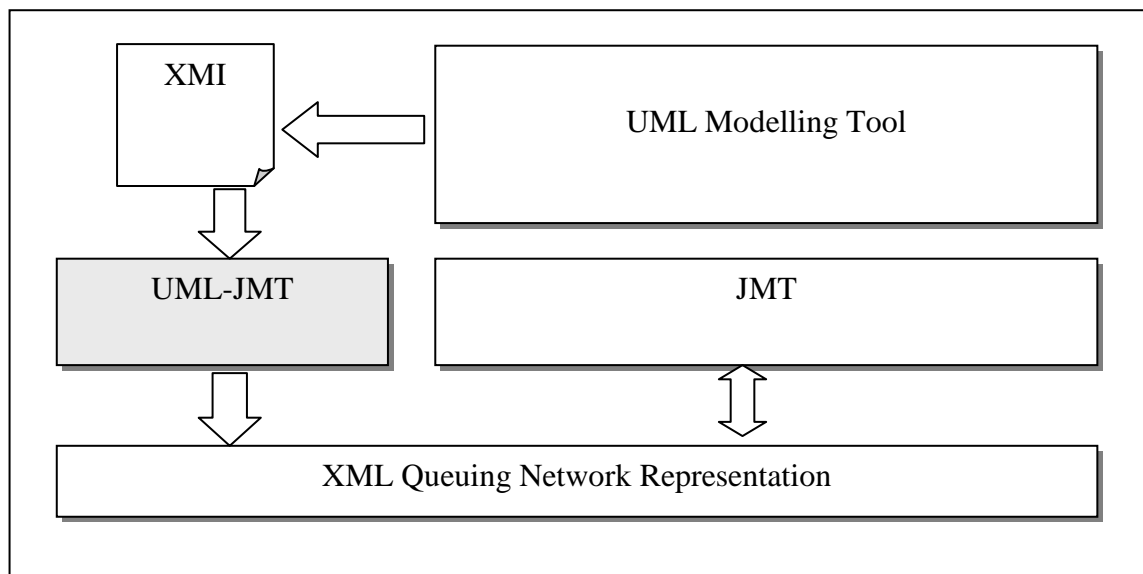


Figure 7.6: UML-JMT tool location in the process of producing performance model from design model.

7.3.4 UML-JMT and JMT: the Integration

As mentioned in Section 7.2.3, JMT suite is built upon an XML communication platform. This means that all the queuing network analysis tools in the JMT suite save the structure of the queuing networks, and the analysis and results in the form of an XML document. This gives us the opportunity to implement a tool that uses the UML-EQN methodology as the UML interface agent for the JMT suite. This interface is implemented through the UML-JMT tool. The UML-JMT tool will deal with the UML diagram of the system being modelled in XMI format and then it will generate the corresponding EQN model in accordance with the UML-EQN methodology. This EQN model will be appended with the performance data collected from the user using the performance data card wizard (which we will discuss later in this chapter). The EQN model will be outputted in an XML document formatted in the JMT suite queuing network DTD. This model can be solved and analysed using the JMT QN simulator and

analysis tools discussed in the previous section. Figure 7.6 illustrates the role of the UML-JMT tool in the performance model generation process.

7.4 UML-JMT Tool Implementation

This section will explain the implementation aspects of the components defining the UML-JMT tool shown in Figure 7.3. We have already explained the design and implementation of the USDX parser in Section 7.1 and are therefore only considering the implementation of the two other components in this section. Section 7.4.1 will talk about the implementation of the interface component represented in the UML-JMT class diagram by the class GUI. The implementation of the functionalities of the queuing network generation engine will be described in 7.4.2.

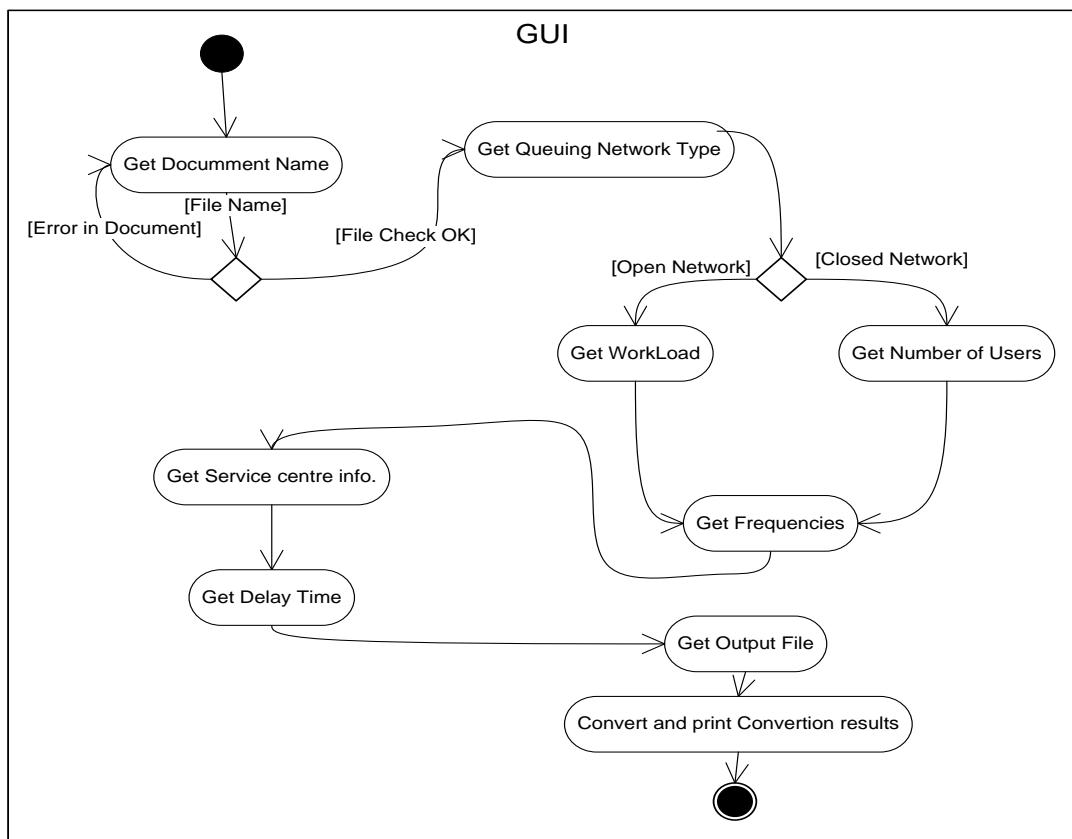


Figure 7.7: UML-JMT interface flow.

7.4.1 Implementation of the Interface

The interface of the UML-JMT tool is implemented as a graphical user interface wizard. This wizard is an interactive question and answer method used to increase the usability of the tool. Figure 7.7 shows an activity diagram that illustrates the flow of the wizard. The wizard will start by asking the user the name of the XMI document that includes the UML model. If the document does not pass the essential checks discussed earlier, the

user will be notified of the type of failure and given a suggestion to fix it. If the document passes the document check, the GUI object will be supplied with information about the model's use-cases, job classes, service centres and delays in the PDC passed from the conversion engine. The wizard will ask the user for the type of queuing network (Open/Closed) and, according to the user's response, he/she will be asked for the number of users, or the workload for each job class, for closed and opened queuing network respectively. The user will progress by supplying performance data regarding the frequencies of each scenario; service centres specifications and delay centre timing. The user will then be asked to select the output file name. When the user is satisfied with the performance data he/she supplied, they can then proceed with the conversion operation. The user will then be given the results of the conversion operation and will then be able to open the resulting performance model in the JMT suite.

7.4.2 Implementation of the Network Generation Engine

As explained in the previous section, the heart of the queuing network generation engine component is the QNGen Class. The methods QNGen class can be divided into pre-conversion and conversion methods. The pre-conversion methods are responsible for preparing the performance model elements that do not depend (or depend partly) on the user entered performance data. These elements include:

- **Communication Maps:** The communication maps are used to define the connections between the service centres. They are created by reducing the communication routes (message flow) for scenarios representing the same use-case. The representation of the communication map class in Figure 7.4 shows that a communication map is a set of messages. The messages are presented as a linked list with each of them pointing to one or more next messages. The pseudo code for creating a communication map for a use-case is as follows:

```

CommunicationMap: UC=new CommunicationMap(ucase name);
// get all the messages of the Scenarios belonging to US
UMLDiagram.Message:M[][];
for(i←0;all scenario belonging to UC)
{
    for(j←0;all messages in scenario i)
    {
        M[i++][j++]←USDX.UMLMode.Sequencediagrams.get(i).getMessage(j);
    }
}
//create in initial COM tree with the first row of messages
message:prev;

```



```

for (i←0; i<M[0].ColCount; i++)
{
    message:mes←new message (M[0][i]);
    if (prev!=null)
    {
        prev.next.add(mes);
    }
    prev=mes;
    UC.addmessage(mes);
}
//create the COM tree branches with the remaining rows of the messages
// if the message is not found in the initial tree add a branch
for (i←1; i<M.rowCount-1; i++)
{
    message:prev;
    for (j←0; j< M[i].ColCount; j++)
    {
        if (M[i][j]!=UC[j])
        {
            message:mes←new message (M[i][j]);
            if (prev!=null) {UC[j-1].next.add(mes);}
            UC.addmessage(mes);
            prev=mes;
        }
    }
}
}

```

The process of creating a communication map involves creating an initial tree with one of the scenarios of the use-case (the one with the longest message list). The next step is the creation of branches of that tree. This process includes comparing the initial tree with the messages of the other scenarios. If the messages differ, then we create a new branch.

- **The job classes:** These will be extracted from the scenarios of the UML according to the algorithm in 6.5.1. Although the identification of the job classes does not depend on the performance data, the declaration of the job classes in the performance model file depends on the queuing network type. We can classify the identification of job classes as a pre-conversion operation, because the job classes

are needed in the in the PDC to identify the number of users or workload, for closed or open queuing networks. The declaration of job classes in the PCD can be completed using this pseudo code:

```
for (i←0; i<USDX.UMLModel.SequenceDiagrams.getLength(); i++)
{
PCD.JobClass.add(USDX.UMLModel().SequenceDiagrams.get(i).getName);
}
```

This will produce a list of scenario names that will represent the list of job class names.

- **The service centres:** According to the algorithm in 6.5 for defining the service centres, the service centres are represented by the components of the deployment diagram. We need to specify the service centres before the conversion operation because we need to consult the user about the specifications of the service and queue parts of it. The list of service centres will be passed to the interface in the PDC where they will be updated with performance data, which is essential for the building of the performance model. The body of the method that creates the list of service centres has the following pseudo code:

```
length← USDX.UMLModel.DeploymentDiagrams.getComponents().getLength();
for (i←0; i< length; i++)
{
Name=USDX.UMLModel.DeploymentDiagrams.getComponents().get(i).getName();
;
ServiceCentre:SC=new ServiceCentre(Name, new Server(), new Queue());
PCD.ServiceCentres.add(SC);
}
```

This loop will create a service centre for each component in the deployment diagram with empty server and queue sections.

- **Delay stations:** Delay stations are created if there is a connection between two components in a sequence diagram, but these components are in different nodes in the deployment diagram. As with the service centres, the time of the delay will be updated by the user in the PDC. The pseudo code for extracting the delay centres is as follows:

```
for (all M:Messages in the Communication maps)
{
if (!USDX.UMLModel.DeploymentDiagram.RInTheSameComp (M.sender, M.receiver)
)
```

```

{
    Delay D=new Delay (M.Sender.getName ()+M.receiver.getName ());
    D.setInvolvedM (M);
    Delays.add (D);
}
}

```

The method ‘RInTheSameComp’ in deployment diagram class remain true if the two sent parameters of type component are in the same node, and false if otherwise. The method will be tested against all the messages in the communication map. Each time a message involves components in separate nodes, a new delay is defined. It will be named with a concatenation of the names of the two components, and this message is added to the involved messages list in this delay. The delays list is defined as a set in which there are no duplications. If another message is involved in this delay, it will be added to the involved list in this delay.

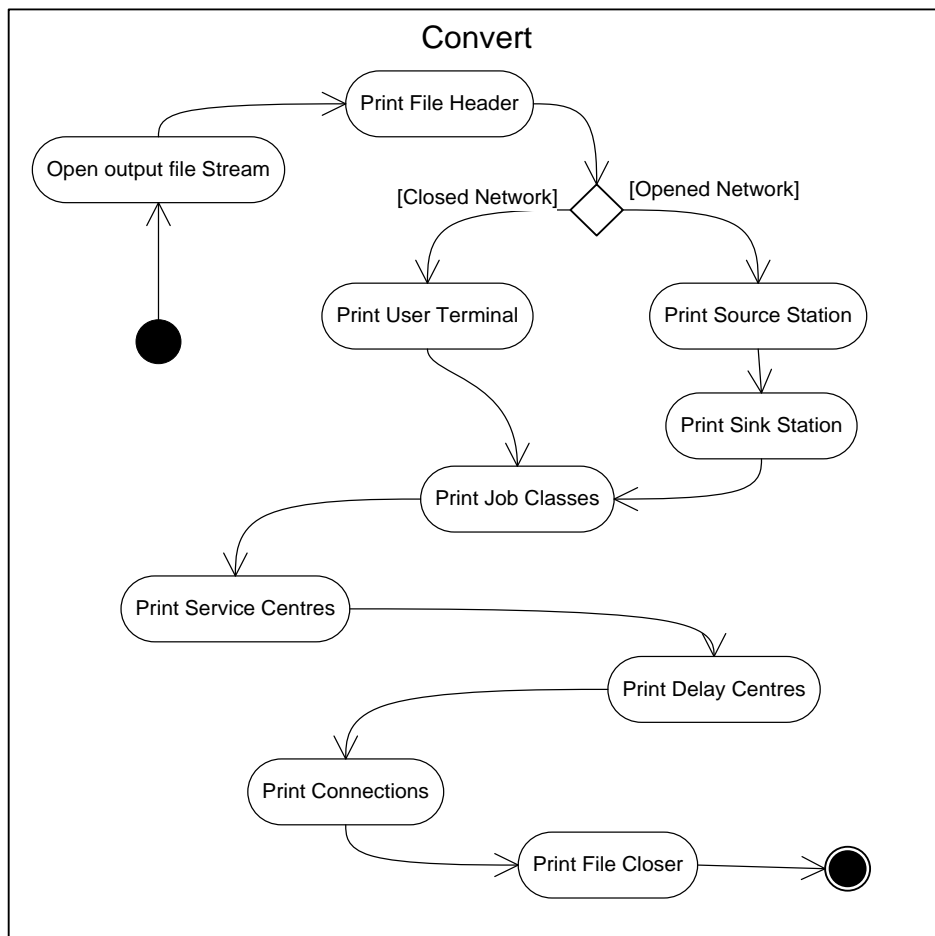


Figure 7.8 Convert Method Activity Diagram.

The Convert method of the QNGen class implements the main functionality of the tool, which is the generation of the performance model. As we explained in 7.2.3, the queuing network model in JMT suite is represented as an XML document. The Convert method performs its function by writing the XML document representing the performance model. It will do this by calling internal methods that will add elements to the XML document. Figure 7.8 illustrates the main activities defining the Convert method. The structure of the generated XML document will depend on the type of queuing network chosen by the user. The main difference between the open and closed queuing network in the JMT structure is in the definition of the start and end points. In an open network, the start station is represented by a source station, and all the finished jobs go to a sink station. These stations are used in the calculation of the throughput. For the closed queuing network, the end and start stations are represented by a delay known as Terminal which simulates the users' thinking time. As we can expect, the type of the network will affect the coding of the elements representing the service centres, delay centres and communications. Therefore, the print methods defined in the classes representing these elements are implemented to cover open or closed networks.

The print methods defined in the service centre, delay and message classes are designed to accept a parameter as a file handle and to print the XML element code for the object they represent. The message class *print* method will print the *connection* element that defines the connectivity between the queuing network components. These are defined according to the communication map messages. All the messages in the communication maps will be reduced to avoid duplicated messages (have the same sender/receiver). In this instance, the resulting messages will be used to define the connection by printing them and using the sender/receiver as the *source* and *target* attributes of the connection element. If the message is one of the involved messages in a delay, then this message will be translated to two connections; one from the source of the delay and the other from the delay to the target.

7.5 Summary

The UML-JMT Tools is a graphical user interface tool that will help users in building a performance model for their software system in a wizard like approach. The user will provide the tool with the performance data card entries in a question and answer approach. The tool will then use the UML-EQN conversion algorithms to construct a performance model based on the user entries. This model can be solved and analysed in a simulation based queuing network solver, provided by the JMT suite. This chapter

provides a full technical specification of the UML-JMT tool. In this chapter, we have discussed the implementation of the UML-JMT tool. The implementation of this tool involved working with UML models in XMI format, this is why the USDX parser was designed. This chapter discussed the design and implementation of the USDX XMI parser. The performance model generated by the UML-JMT tool is structured to be solved and analysed by the queuing network solution and analysis tools provided by the JMT suite. We have discussed in this chapter the queuing network format deployed in the JMT tool and. The chapter also discussed the design and implementation of the UML-JMT tools and how it was divided into components that implements the conversion algorithms discussed in chapter 6.

In the last two chapters, we have seen the description of the model transformation methodology (UML-EQN) and how it was realised as a performance evaluation tool (UML-JMT). In this chapter and the next, we will discuss the evaluation of our methodology from both qualitative and the quantitative points of view. In the qualitative evaluation in Chapter 9, we will investigate the attitude of a sample of software engineers toward the methodology and the tool. In this chapter, we will investigate the methodology and the tool in the context of the deployment as an aid in conducting a performance study and the degree of accuracy of the results provided by the UML-JMT tool. What we are looking for is to demonstrate the use of the tool and to verify that the performance indices provided by performance models built by the UML-JMT tool are valid to a degree of accuracy. The methodology we are considering for the quantitative evaluation is by demonstrating the deployment of the tool as an aid for conducting a performance evaluation study and comparing the results gained by a performance model produced by the UML-JMT tool and analysed by the JMT suite to performance indices provided by a deterministic benchmarking exercise. This chapter will be used for demonstrating the use of the UML-JMT and for validating the results gained from the tool in two case studies. In Section 8.1 of this chapter, we will explain the first case study where the performance of an information retrieval system will be studied. In 8.2, we will investigate the performance of a national payment switch.

8.1 Demonstrating UML-JMT

In this section, we will provide an example that will demonstrate the deployment of the UML-JMT tool as an aid in a performance evaluation study. This demonstration involves an information retrieval system discussed in [93], and it was used to demonstrate a similar methodology named PRIMA-UML which was realised with the XPRIT tool. The objective of this case study is to demonstrate the use of the performance data gathering mechanism deployed in UML-JMT tool and the analysis tools available in the JMT suite, and compare their role in the performance evaluation experiment to the role of a similar tool. We choose this example because the end performance model generated by the

PRIMA-UML is an execution graph which will be translated using the SPE methodology to an EQN, the same as the one produced by our methodology.

8.1.1 PDC for the IRS

The example in the PRIMA-UML paper showed an information retrieval system with internal and external search modes. The internal search was done on a local database and the external was performed using three browsers searching information on the Internet. In this section, we discuss the UML diagrams representing the architectural and behavioural characteristics of the IR system. The IR system offers the user two types of search - internal and external. Before the user can search the database, the system will authenticate the user by checking a username and a password.

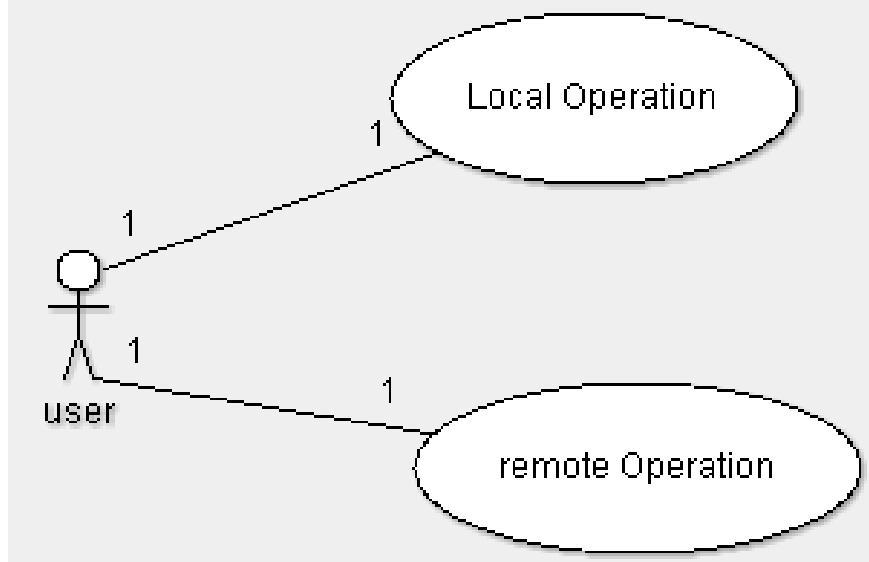
Table 8.1 explains in detail the PCD of the information retrieval system performance study. The objective of the study is to study the effect of increasing the number of users in the system on throughput and response time, and how are they effected by the utilisation of the different components of the system.. The architecture of the system is defined by the structural and behavioural UML models represented by the deployment (Figure 8.2), use-case (Figure 8.1) and sequence diagrams (Figures 8.3-6). The critical scenarios that define the system behaviour are as follows:

- S1: The authentication process fails. (Figure 8.3).
- S2: The authentication succeeds and the user finds the searched item Figure (Figure 8.4)
- S3: The authentication succeeds and the user did not find the searched item. (Figure 8.5)
- S4: The authentication succeeds and the user searched for a remote item. (Figure 8.6)

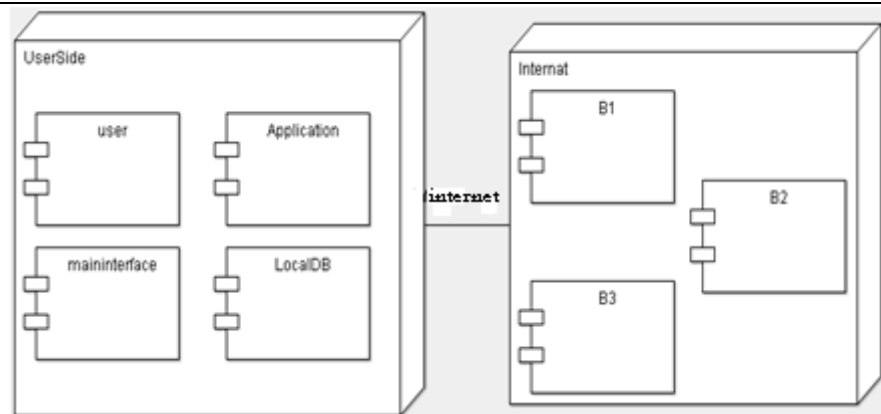
The performance characteristics of the IR system defined by the work load and the service demand are shown in tables 8.1.1 and 8.1.2 respectively.

Table 8.1: PDC for the IRS.

Objective	To study the effect of increasing the number of users in the system on throughput and response time, and how are they effected by the utilisation of the different components of the system.
------------------	--

Use-cases**Figure 8.1:** Use-case diagram of the IR system.

The IR system can have two use-cases, either to search the local database or the remote database. In both of these operations the system will authenticate the user first.

Architecture**Figure 8.2:** Deployment diagram of a suggested architecture of the IRS.

The suggested architecture of the IR system where the system is working in two different nodes: a user side node representing the user machine and the Internet node representing the server containing the database searched by the browsers. In the user side, the components available are application, main interface and the local database. The user component is placed in the user side node to

model the stakeholder position in the system. The two nodes are connected by an Internet connection

Scenarios

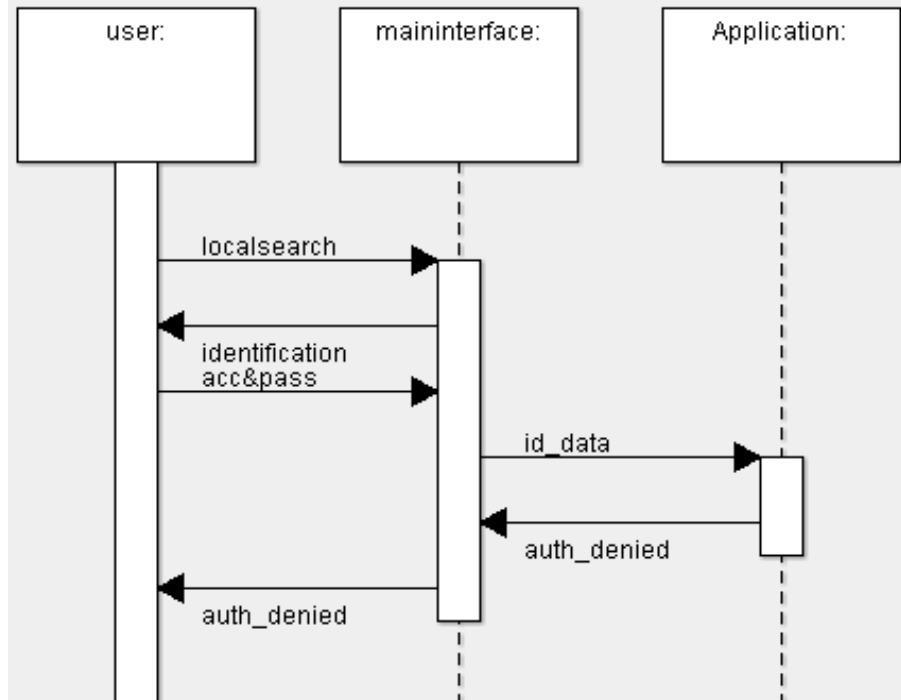


Figure 8.3: Sequence diagram of the authentication process fails scenario.

S1: Scenarios of a local operation where the user tries to access the IR database but the authentication operation fails.

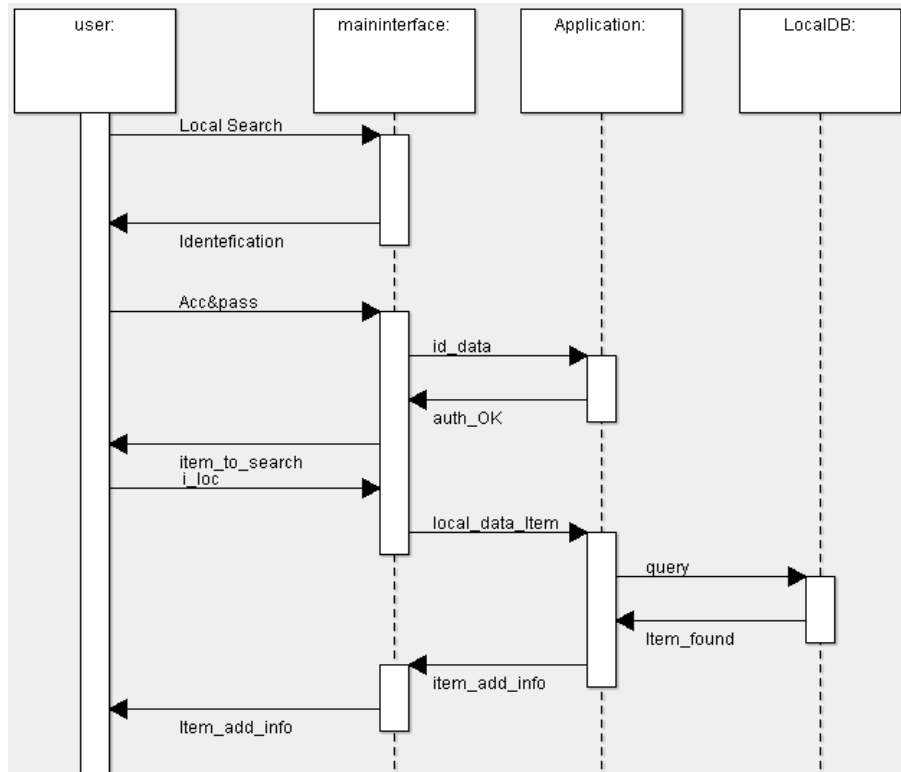


Figure 8.4: Sequence diagram of the authentication succeeds and the user finds the searched item scenario

S2: Scenario of a local operation representing a successful search operation in the IR database

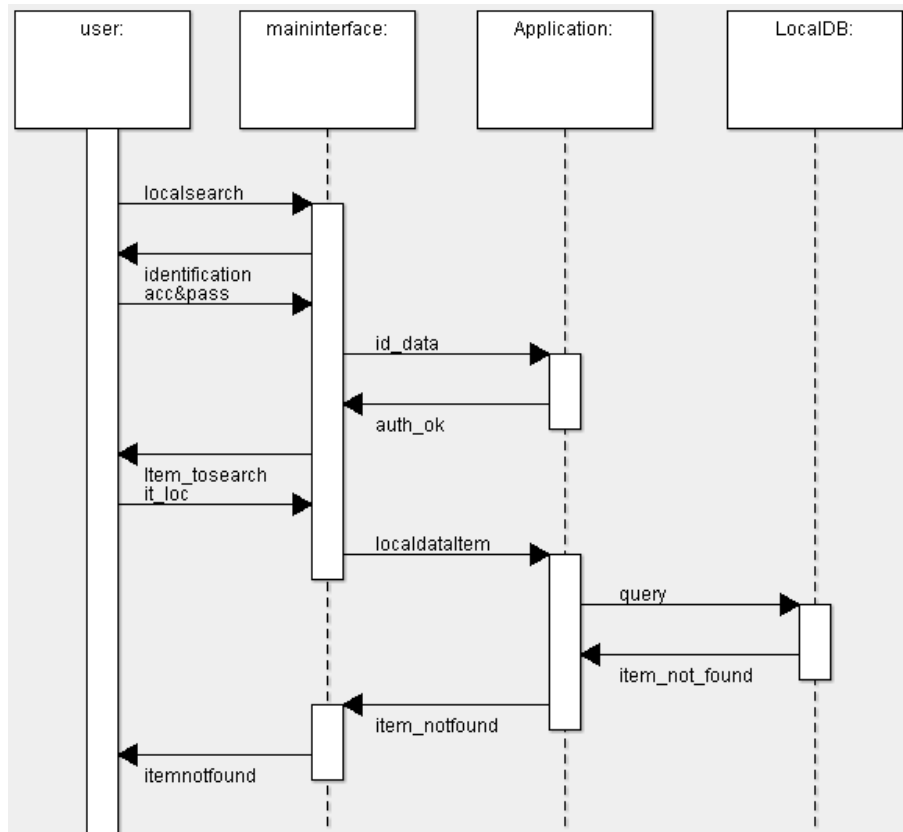


Figure 8.5: Sequence diagram of the authentication succeeds and the user did not find the searched item scenario.

S3: Scenario of a local operation where the user will not find the requested item of search.

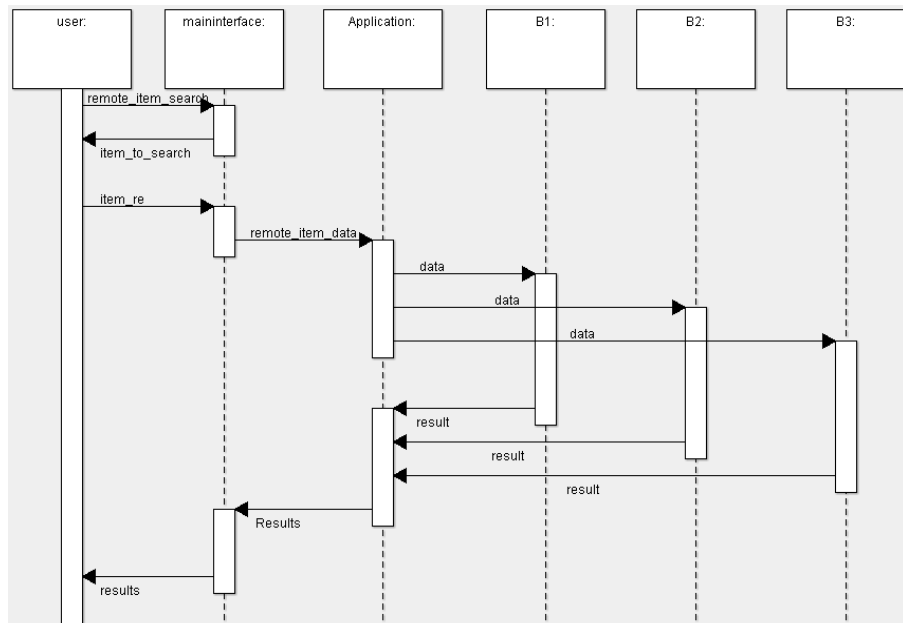


Figure 8.6: Sequence diagram of the authentication succeeds and the user searched for a remote item scenario.

S4: Scenario of a remote operation. The user will ask the application programme to conduct a search remotely and the application programme will conduct this search concurrently in three different browsers.

Table 8.1.1: <i>Workload (Assumed by the Author)</i>	S1	S2	S3	S4
	5%	25%	20%	20%

Table 8.1.2: <i>Service Demand (Seconds)</i>	Component	Service time (Seconds)
	Interface	0.00001
	Application	0.00001
	Local DB	0.0005
	Browser1	0.0005
	Browser2	0.0005
	Browser3	0.0005
	Thinking Time	5

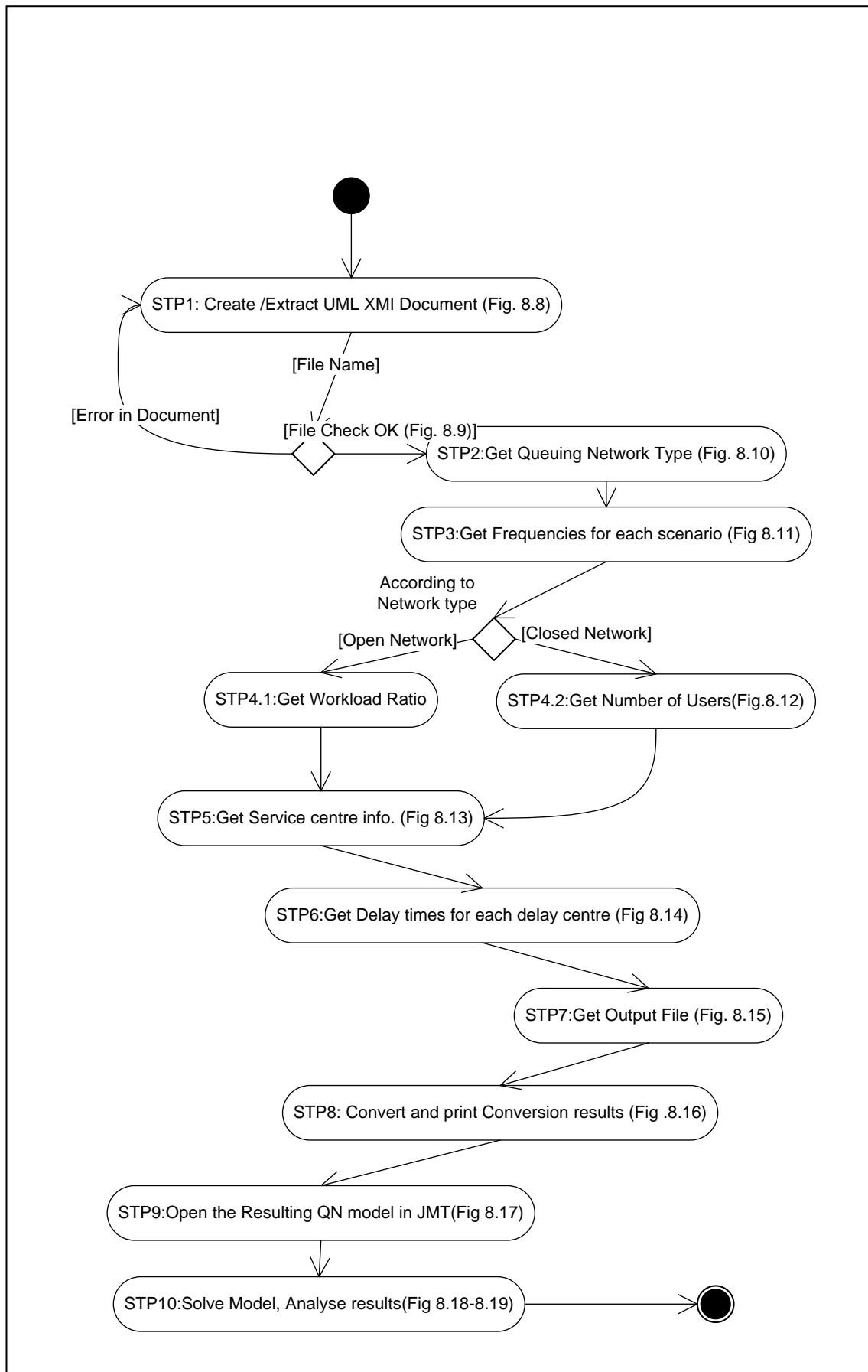


Figure 8.7: Flowchart for the process of conducting a performance study using the UML-JMT tool

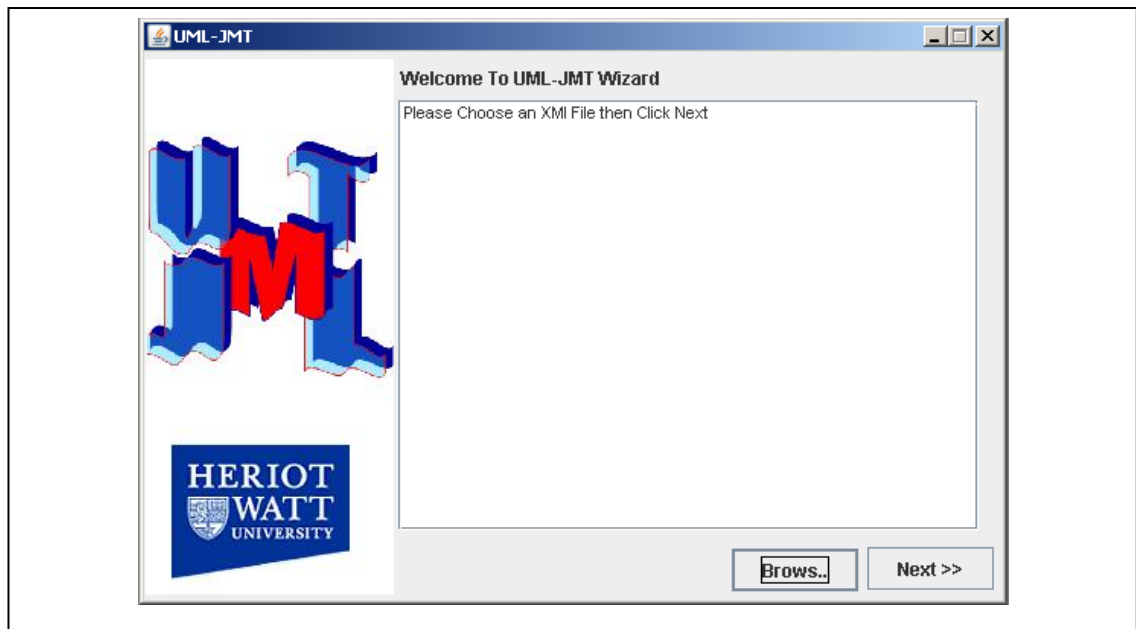


Figure 8.8: UML-JMT Tool Wizard interface

8.1.2 Using UML-JMT to study the IRS performance

In this section, we will explain the steps for conducting the performance evaluation test for the IRS using the UML-JMT tool. The steps for conducting a performance test using this tool are explained in Figure 8.7. This figure shows a flowchart diagram with the steps needed to conduct this performance study. As the diagram shows, there are ten steps for conducting a performance study. These are as follows:

Step 1-Creating the Design Model:

The first step is to model the structural and behavioural characteristics of the IR system. This can be done by defining the titles and owners of the main user stories as a use-case diagram (Figure 8.1), and then further explaining the scenarios of these user stories as



Figure 8.9: UML-JMT Tool Wizard interface after the XML file is chosen and error checked

sequence diagrams (Figures 8.3, 8.4, 8.5 and 8.6). The suggested architecture of components' distribution is modelled in a deployment diagram (Figure 8.2). We have used the ArgoUML [43] tool to model the design representation of this system. The ArgoUML tool allows us to export an XMI representation of this model which we will use as an input to the UML-JMT tool. We have exported the modelled UML design in a file named IRS.xmi. When starting the UML-JMT tool, a wizard like GUI will run. The interface for the GUI representing the tool is shown in Figure 8.8. The first screen of the wizard contains an instructions pane that will be used to provide the user with instructions and inform him/her of any errors (i.e. if the XMI document supplied does not pass the initial test and why). In the screen shot in Figure 8.8, the instruction pane requests the user to choose an XMI document file. After choosing the input XMI file using the browse button, the XMI representation of the UML model will be checked for consistency, and the result of this check will be displayed on the instruction pane (see 8.9). If the file passes all the initial tests, the user is instructed to proceed to the next step. When the user clicks 'next', the USDX parser will construct the internal object representation of the model. Next, the interface component will prepare the PDC for the user to complete.

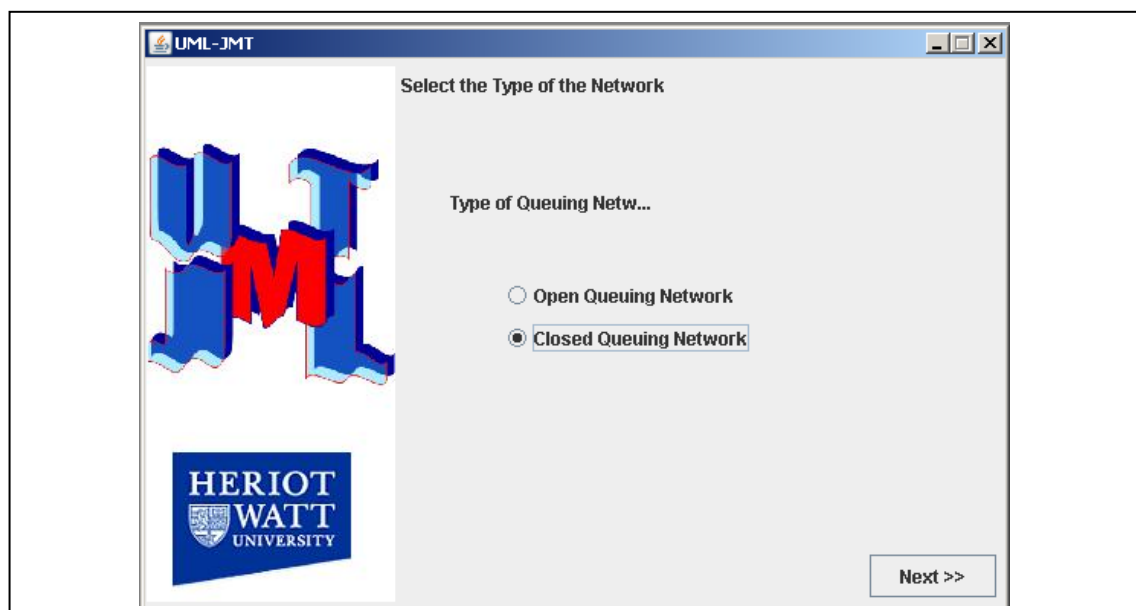


Figure 8.10: Choosing the Queuing network type Screen

Step 2 - Choosing Queuing Network Type:

This will be done by choosing network type depending on the type required by the performance study. For the example used, we choose to represent the model as a closed queuing network. This is because the network in the example which we are comparing the results gained in this example with is a closed queuing network. (Figure 8.10).

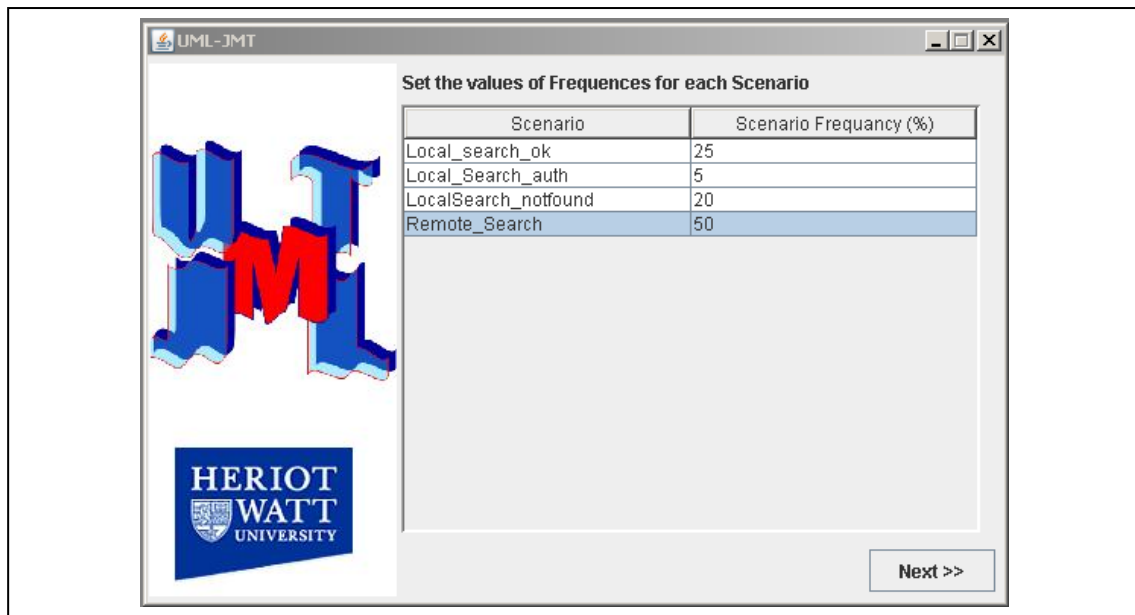


Figure 8.11: Entering the frequency of each of the scenarios

Step 3 - Setting the Frequencies:

After choosing the network type, the user will be asked to provide the frequencies for each of the scenarios identifying the system. The names of the scenarios will be listed in a table along with empty fields that will be used by the user to write in the frequencies. In this example, we assumed that half of the time users search locally and the other half they search remotely. For the local search, we assumed that 50% of the time the user will find their search item, 10% they will log in with incorrect authentication and 40% will not find their searched item (Table 8.1.1). The screen shot of the frequency collection step is shown in Figure 8.11.

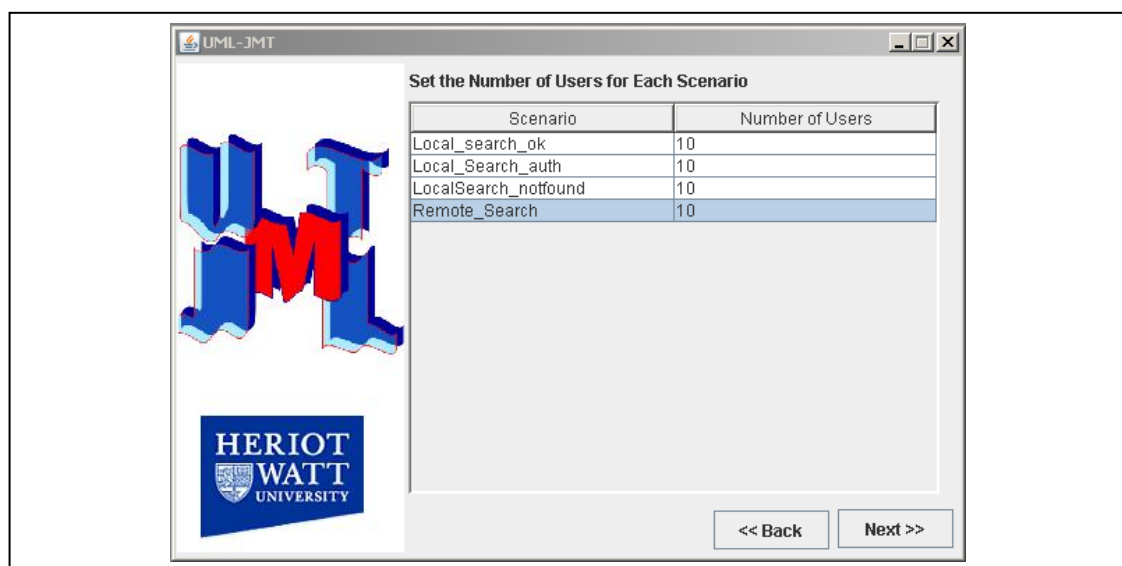


Figure 8.12: Entering the number of users of each of the scenarios

Step 4 - Setting the Number of Jobs/Workload:

Depending on the type of network chosen by the user, they will have to supply the number of jobs or the workload for closed and open networks, respectively. The screen that collects this information shows the list of scenarios and the fields that the user will use to insert the number of jobs (users) of this specific type. (Figure 8.12).

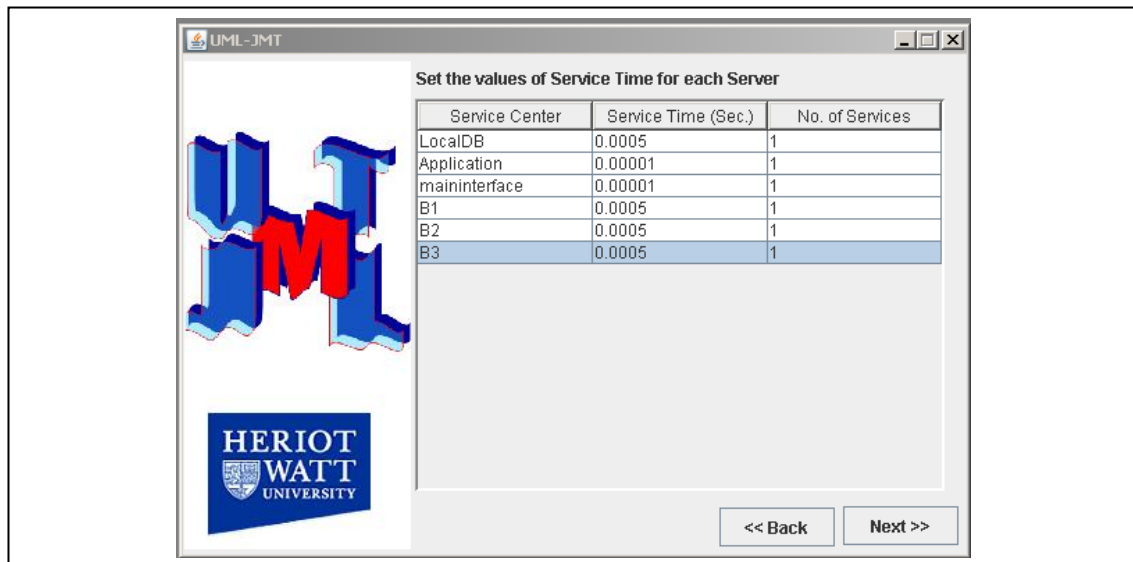


Figure 8.13: Entering the service demand time for each component

Step 5 - Setting the Service Centres:

In this step, the user will be provided with the components composing the system under study and he/she will be asked to supply the service time (in seconds) and the number of servers in each service centre. In this example, the application and main interface components need 0.01 ms to complete a job, whereas the Local DB and the browsers require 0.5 ms. the number of servers in all the service centres are 1. (See Table 8.1.2, Figure 8.13)

Step 6 - Setting the Delay Centres:

The Delay Centres extracted from the model will be shown to the user in order to provide the average delay time for each of them. The screen collecting the delay times is shown in Figure 8.14. We have decided to give half a second for each connection. As we are modelling a closed queuing network, another delay is added to the delay list. This delay represents the thinking time of the users supplying jobs to the network. We have assumed that the thinking time can be an average of 5 seconds.

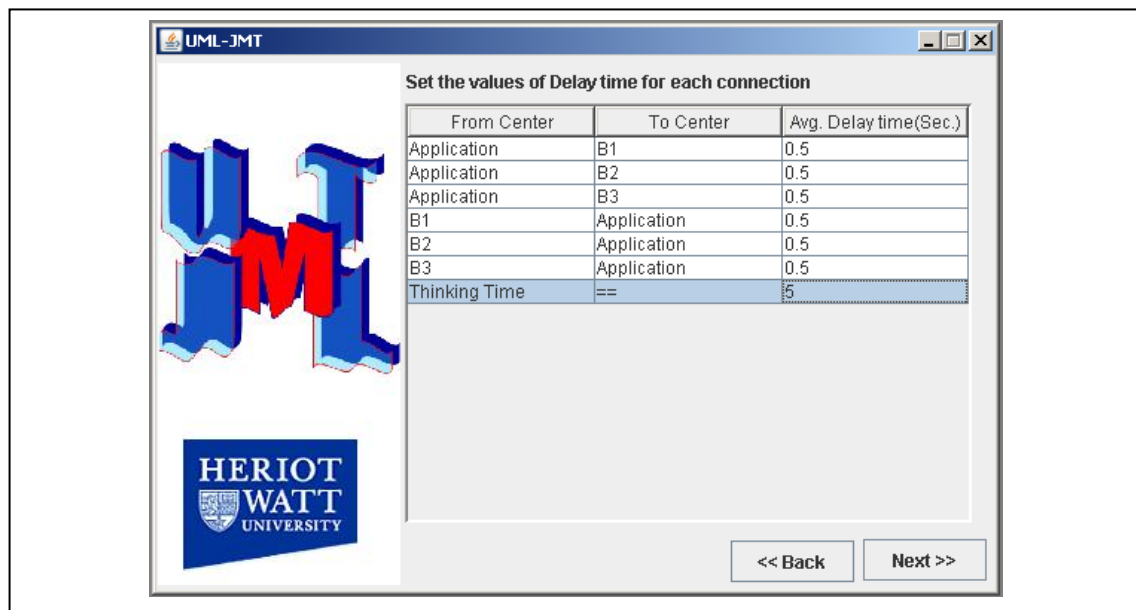


Figure 8.14: Entering the service demand time for each component

Step 7 - Choosing the Output File:

The last step before starting the conversion process is to choose the output file where the performance model will be saved. The file is a JMT suite simulation file with the extension (.jsimg). In this example, we chose to call the file IRS.jsimg. (Figure 8.15).

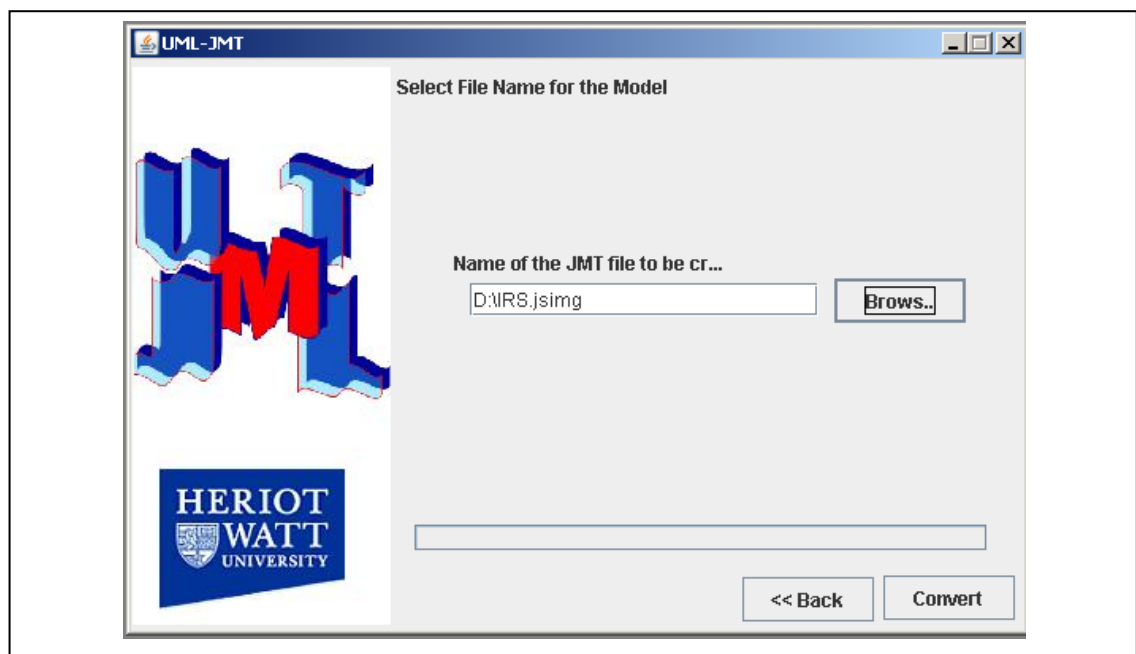


Figure 8.15: Entering the service demand time for each component

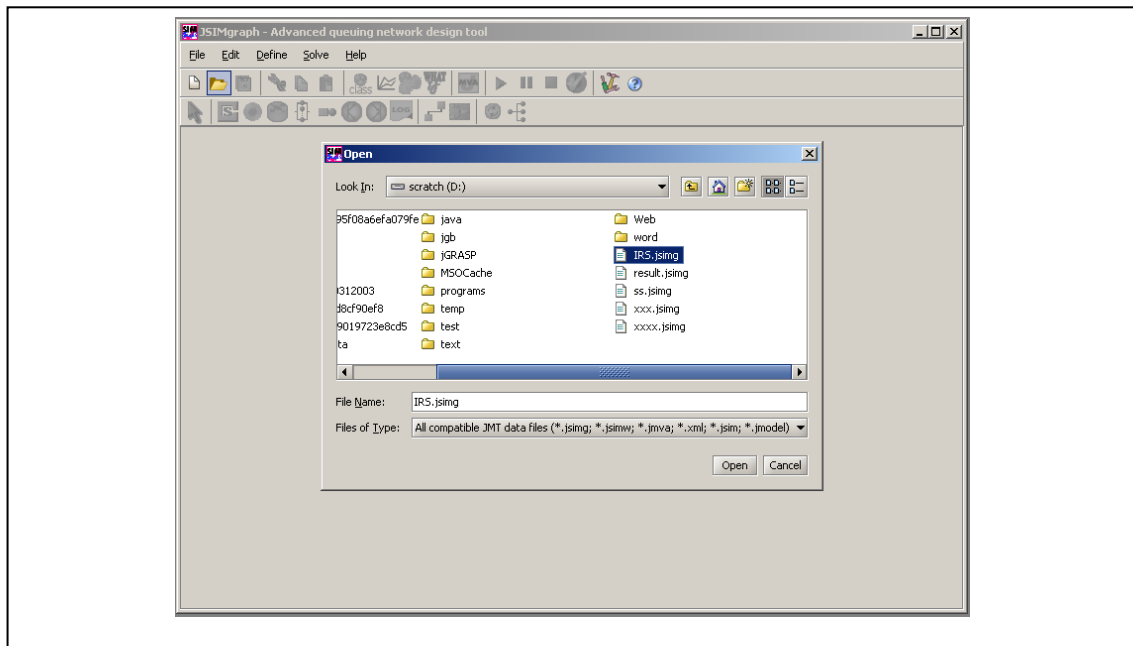


Figure 8.16: Opening the Resulting Performance model in the JMT suite

Step 8 - Convert and Print Conversion Results:

When the user clicks the convert button, the conversion progress can be monitored on a progress bar. The conversion process will include writing the performance model XML file. After the resulting file is created, the user can open this file in JMT suite to be solved and analysed.

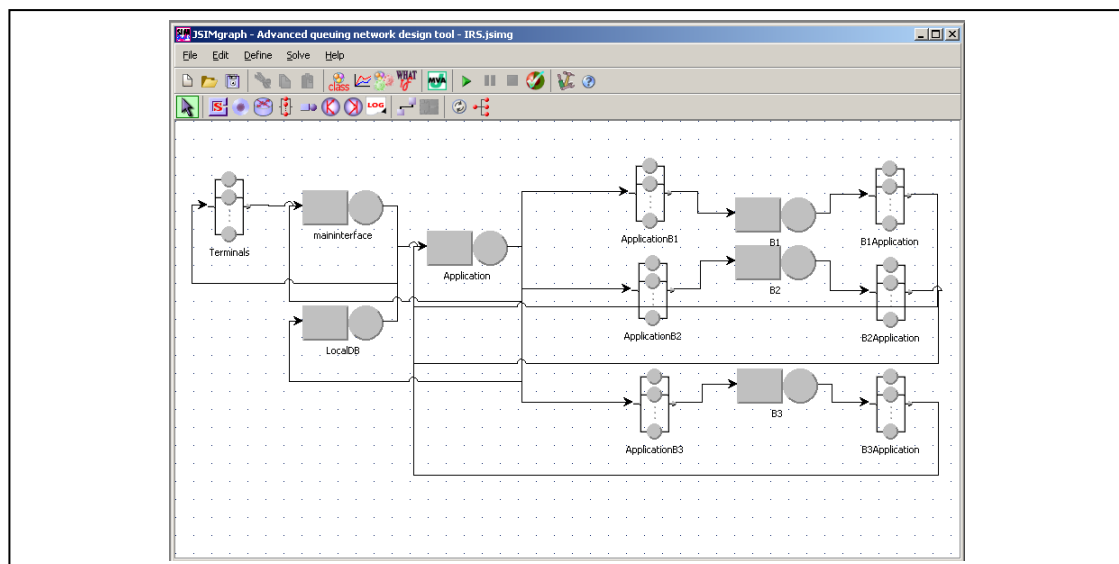


Figure 8.17: The performance model generated by the UML-JMT wizard

Steps 9 & 10 - Opening/Solving the Performance Model:

The last step of the performance study is to open the generated model in the JMT suite and to solve this model. As we mentioned earlier, the queuing network model is designed to be solved in the JSIMgraph(explained in 7.2.1) tool of the JMT suite (Figure 8.16).

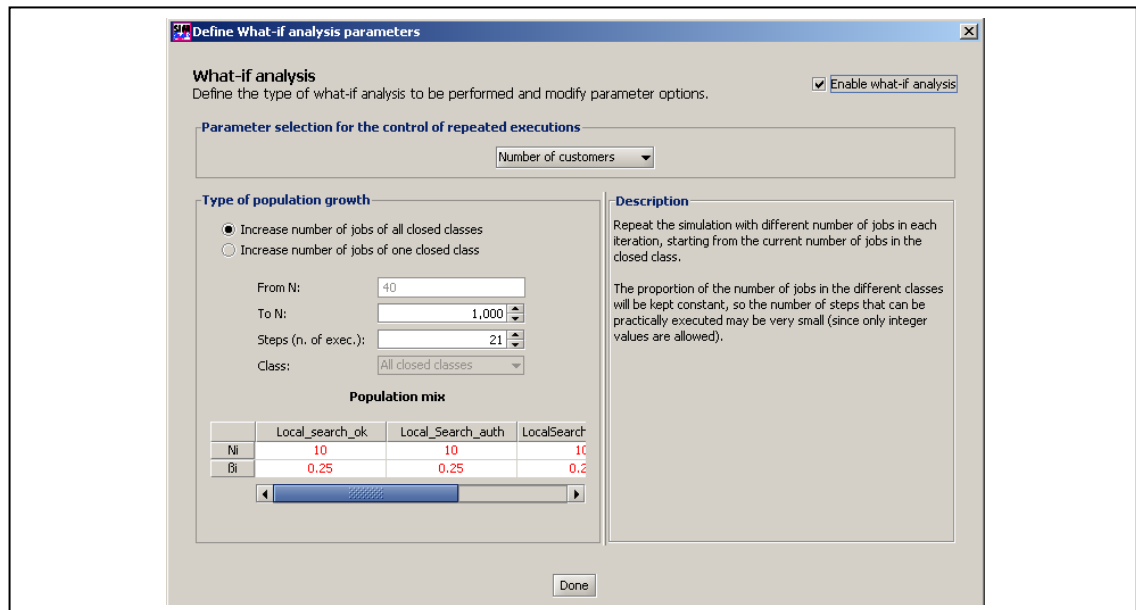


Figure 8.18: The What-if analysis tool used to study the throughput of the IRS

After choosing the file generated by the UML-JMT tool, the generated model will open (Figure 8.17). The next step is to choose the performance indices that need to be studied. The performance indexes' choice screen allows the study of any performance index for any class or component. This will supply the user with a wide range of performance readings for the systems' under study. For this experiment we chose the system's throughput, response time and the utilisation for the components. We will use the 'What-if' analysis tool to conduct a performance study of observing the change of the system's performance readings as the number of customers increases (Figure 8.18). Therefore, we select the control parameter in the 'What-if' tool to be the number of

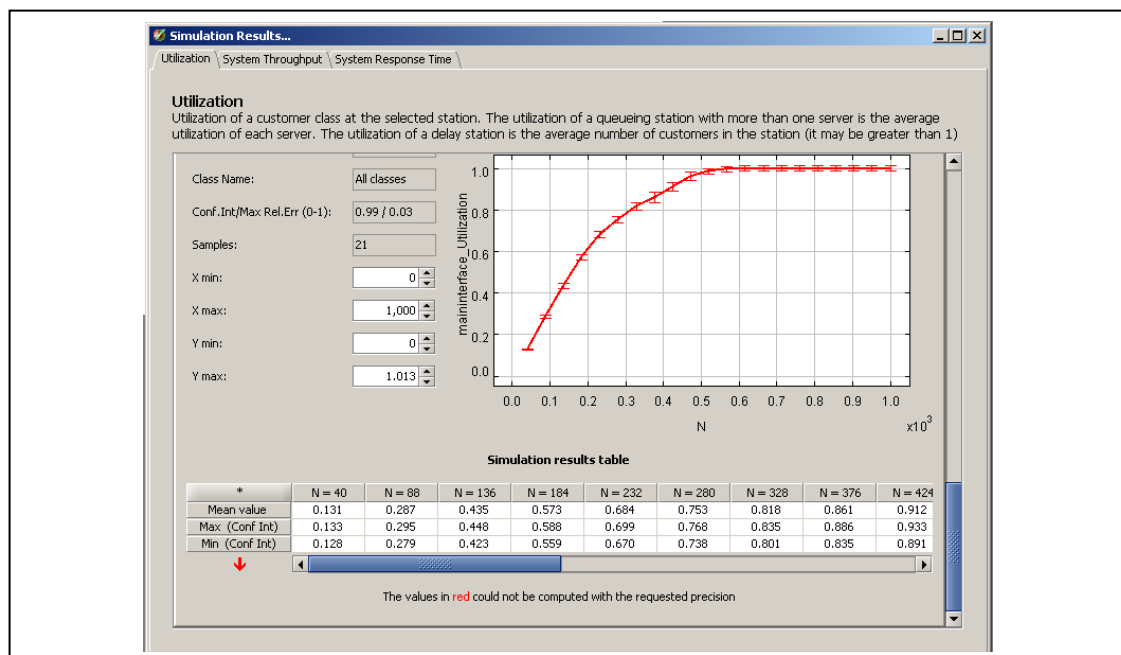


Figure 8.19: The result of the performance study showing the throughput/user growth

customers and set the ranges for that parameter and the number of executions of the simulation study.

After we start the simulation process, the ‘What-if’ study will start and will simulate the network with the required number of users for the specified number of executions. The results of the simulation will be shown with the maximum, minimum and mean values for the systems throughput for each case, with a specific number of jobs in the system (Figure 8.19).

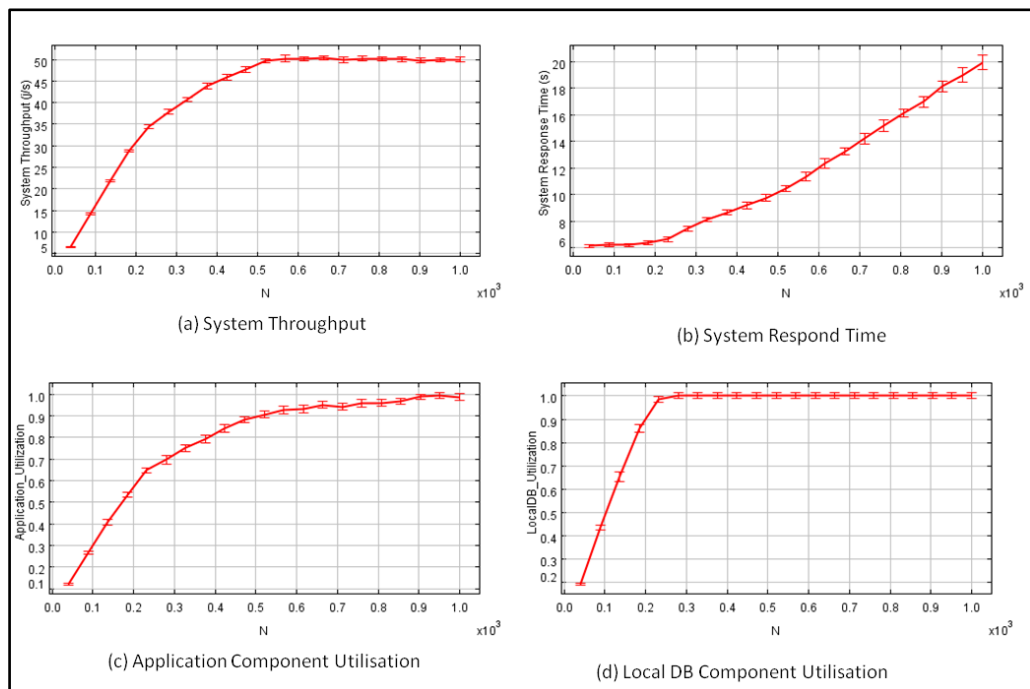


Figure 8.20: Performance results of the IR system extracted from the UML-JMT tool..

8.1.3 IRS Performance Results

Figure 8.20 shows the performance results gained after testing the effect of increasing the number of users in the system from 40 to 1000 users. These results can be used to evaluate the performance characteristics of the suggested design. For instant, the system throughput reading in 8.20(a) shows that the system will arrive at the peak throughput at 500 users when it will deliver 50 jobs/s. On the other hand, the response time will begin to extend when the number of users of the system exceeds the 200 mark (as 8.20(b) show) by comparing this to the throughput and utilisation graphs, we can clearly see that the cause of this increase in the response time is caused by the full capacity of the Local DB component. This gives us an indication that we need to improve this component in order to gain a shorter response time. Figure 8.20(d) gives us an indication that this component did

not reach its full capacity although the system reached its maximum throughput. This means that the current configuration for this component is suitable for now.

8.1.4 UML-JMT as an Experimentation Tool

In this section, we have demonstrated the use of UML-JMT tool and JMT suite in a performance evaluation experiment. This was done by explaining the performance data gathering wizard implementing the PDC, and the experimentation functionalities provided by the what-if analysis tool available in the JMT suite. This combination provides a semi-automated experimentation suite for aiding the evaluation of a system's performance.

We stated earlier that the closest combination to our own combination is the one provided by the XPRIT tool that implements the PRIMA-UML methodology and the SPE.ED tool for evaluation the resulting performance model, as they used the same UML models and generate the same output performance model. We can clearly view the differences between the two combinations particularly in the level of assistance offered by the UML-JMT tool in the automation of performance model building and the experimentation aid provided by JMT tool. The UML-JMT provides a UML interface for a user-friendly performance evaluation suite that provides easy to understand, standard visualisation of the resulting EQN model, also, experimentation tools that will assist the performance study. As we saw in this section, the user is asked to provide a UML representation of the system architecture. This model will be consulted to build a set of performance variables, which are required to conduct the performance study, these variables will be queried from the user in a question answer method. The resulting performance model can be inspected and amended by the user (if required) in the JMT workbench. We saw how can we select of the performance characteristics under study and the nature of the study can be easily done in the JMT tool. We have discussed earlier the what-if experimentation tool available in the JMT suite.

On the other hand, the XPRIT tool require the user to specify the temporal and frequency data in the UML model in the UML-SPT format discussed earlier in 5.2.1. This method was not ideal for the users as we will discuss in the next chapter, as *all* the participants interviewed preferred the question/answer method adopted by the UML-JMT tool. This UML model will be used to build the machine and the software (EG) models specified in the SPE methodology. These models can be opened in the SPE.ED

tool. In this tool, the user will be able to annotate the EG with the appropriate workloads and frequencies. The resource requirements are defined as to be predefined template (i.e. CPU, DB, and Screen) or it can be used defined, which means that it can be separated if a detailed study is required. Then the software and machine models can be accompanied to an EQN, which will be solved using a discrete event simulator for a predefined set of performance indices. The SPE.ED tool provides an excellent analysis tool that provides the ability to compare results of different configurations of the design, thanks to the SPE database which stores models and results of previous performance studies and provides functionalities to compare them. Although this analysis tool provide the ability to analyse the performance model using a query system that specify and goal performance characterisation and a testing workload, it lacks the ability of experimentation available in the what-if analysis which allow the experiment to be conducted within an changing environment(workload, service time ... etc). Also, the visualisation used in the SPE.ED tool for the performance model and the performance results do not offer the standard notational representation of the queuing networks as figure 8.21 shows.

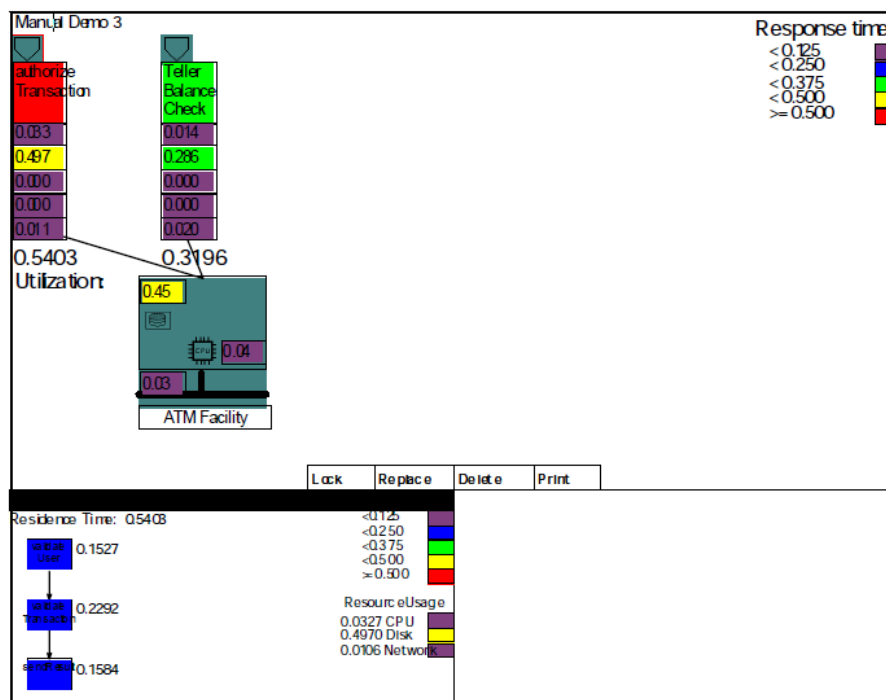


Figure 8.21: Snapshot of a performance model and its results in SPE.ED tool

8.2 Validating the Results' Degree of Accuracy

As part of the quantitative validation of the UML-EQN methodology and the UML-JMT tool, we will validate the degree of accuracy of the performance results gained from the

tool when realising this methodology. The degree of accuracy is measured by the margin of error between the performance results forecast by the tool for a system's design, compared to the real performance results taken from the system after it has been developed. As we explained earlier, the non-deterministic system modelling methodologies provide performance results ranging from 10-30% of accuracy. The results accuracy validation process that we will discuss in this section will involve studying the performance characteristics of a payment gateway by studying the effect of the suggested design on the throughput of this system. We will start this section by explaining the payment switch system. In 8.2.2, we will explain the specifications of the system under study by explaining the architecture, and scenarios of the system. 8.2.3 will explain the steps of the performance study, and finally, we will discuss the results and compare them to the results gained from the real system.

8.2.1 Case Study: Payment Switch

In this case study, we will consider a national payment switch designed to deploy electronic card based payments. The name of the payment switch will be anonymous in this thesis as the author has signed a non-disclosure agreement for all the information regarding this system. This payment switch was founded to allow payment operations for all the cards issued by the banks participating in this switch to be used in all the POS (Point Of Sale) terminals used by retailers who have a merchant account with any of the member banks. This can be accomplished by linking all POS terminals throughout the country to a central payment switch, which in turn processes and re-routes the acquirer financial transactions to the card issuer, whether it is a local bank, VISA, AMEX or MasterCard. The payment switch was founded in 1991, and since then, the number of POS terminals has increased from 18,537(1993) to 76,104(2008). The number of cards issued with this payment switch logo has jumped from 5.56m in 2001 to 13.23m in 2009, and consequently the number of POS transactions has jumped from 18m transactions in 2001 to 121m transactions in 2008. This massive increase in the demand of the services of this payment switch required the owners of the switch to upgrade the payment switch system. The system upgrade project was initiated in 2002 with a full system change at both hardware and software levels.

In this case study, we will discuss the performance characterisation of this payment switch system from the throughput perspective. This will include checking the architecture that was suggested for the new upgraded system for its ability to deliver the

number of transactions potentially required in the RFP for the upgrade project. The RFP of the upgrade project stated that the payment system should be capable of processing more than 100 transactions/second(TPS) This increase is reflected by the strategic plan of the payment switch to expand the number of POS terminals and payment cards and to introduce new types of electronic payments, such as internet and mobile payments. As the new payment system is currently online, we will use the UML-JMT tool to investigate the expected throughput of the original suggested system architecture and we will compare the results to the actual throughput of the online system. This will provide us with an indication of the accuracy of the results provided by the UML-JMT tool.

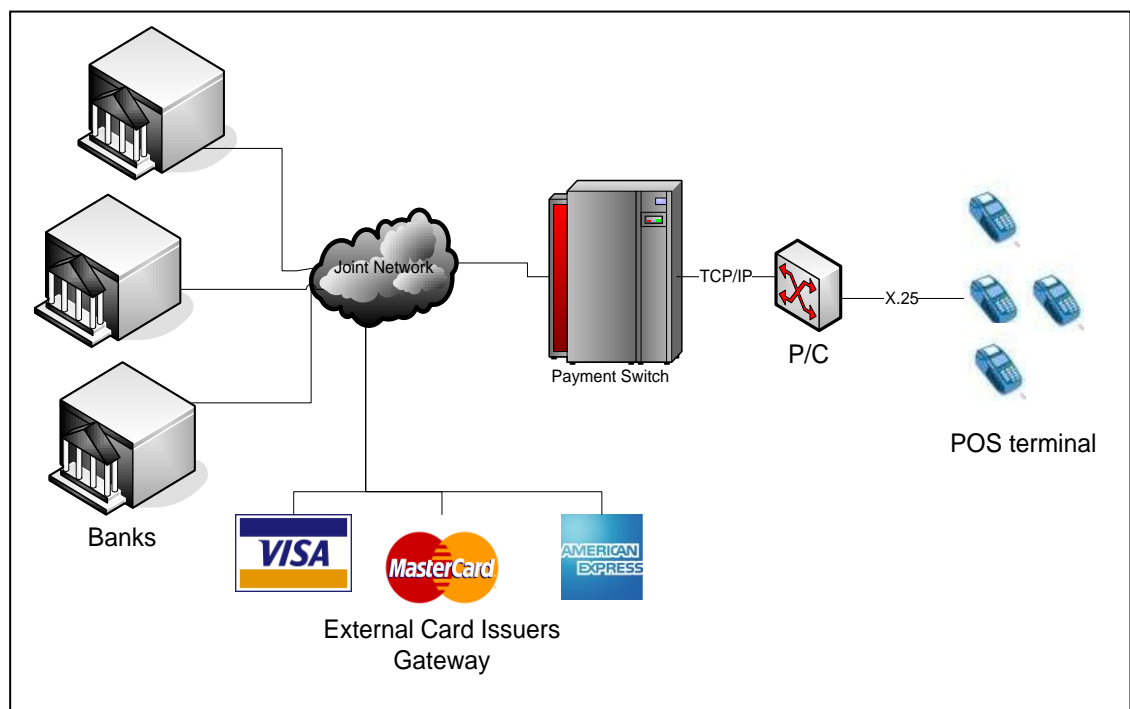


Figure 8.22: logical architecture of the payment system

8.2.2 Payment System Architecture and Scenarios

The payment system we are considering in this case study provides financial services that include both ATM and POS related transactions. In this case study, we will only consider the POS transactions and operations. Figure 8.22 shows the logical architecture of the portion of the system responsible for the POS operations. The payment system consists of a payment switch responsible for connecting the POS terminals to the member banks' systems. The banks are connected to the payment switch by a private secure high speed network. The payment switch is also connected to the major credit card issuers' gateways to forward any credit card operations. As credit card operations only represent a small proportion of the total number of transactions, we will only

consider the debit card operations passing through the payment system. The POS terminals are connected to the payment switch through a third party communication network. This network is an X.25 network. As the payment switch only recognises network packets formed as TCP/IP, a protocol converter is used to convert the TCP/IP packets to X.25 and back to TCP/IP for all transactions between the payment switch and the POS terminals.

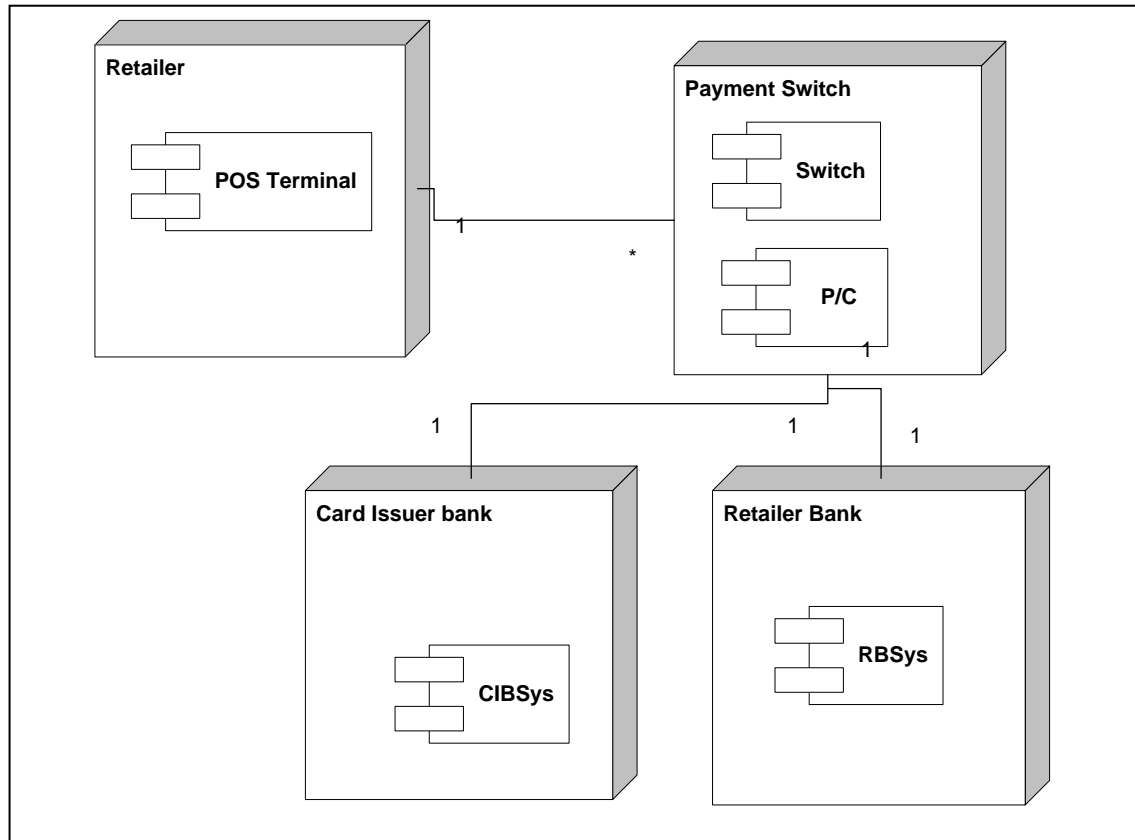


Figure 8.23: Deployment Diagram of the Payment system

System Architecture

Figure 8.23 shows the deployment diagram of the payment system. In this diagram, the system is scattered among four sites representing the retailer where the POS terminal is located, the payment switch, the retailer bank and the card issuer banks. In the payment switch site, the system has two components, which define the performance characteristics of the system. These are the transaction router component, responsible for analysing and forwarding the financial transactions from and to the POS terminal, and the member banks. The other component residing in the payment switch is the protocol converter (P/C). This component is responsible for encapsulating the TCP/IP networks packets travelling on the X.25 network connecting the POS terminal to the switch. The card issuer and retailer banks' sites contain the bank systems which are responsible for issuing the authentication and approval or denial transactions for online

payments. The card issuer bank is the bank where the customer holding the card has his/her accounts. The retailer bank is the one responsible for providing the retailer with the certified POS terminal and opening a merchant account for the retailer.

System Scenarios

This case will include the financial transactions passing in the payment system, as they represent the majority of the transactions and can therefore be seen as the critical transactions shaping the performance characterisation of the system. By applying the 80/20 rule, we have grouped the transactions with similar scenarios and found that the system will cover five main scenarios. These are as follows:

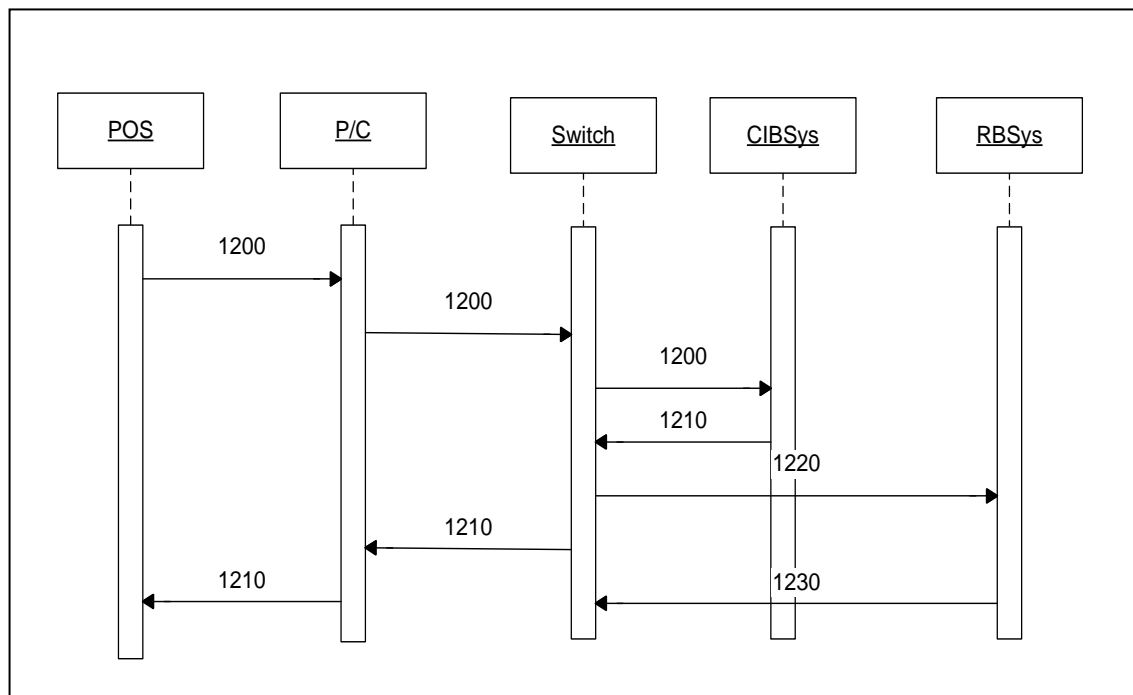


Figure 8.24: Normal transaction scenario in the payment system.

Normal transaction: Where a full normal transaction passes through the switch. Figure 8.24 shows the scenario of a normal transaction. The numbers on the messages represent the ISO payment transaction numbers illustrated in [121]. The transaction starts with a financial request (1200) which will be forwarded to the switch through the protocol converter. The switch will confirm the transaction format and the originated POS terminal and then forward this request to the card issuer system. The card issuer system will respond to this request after checking the customer's account and available funds, and will then reply to this message, instructing it to either go forward with the operation or decline it in a financial request response (1210). If the decision is to accept the operation, the switch will issue a financial advice (1220) to the retailer bank and forward the financial request response to the POS terminal. The retailer bank will send a

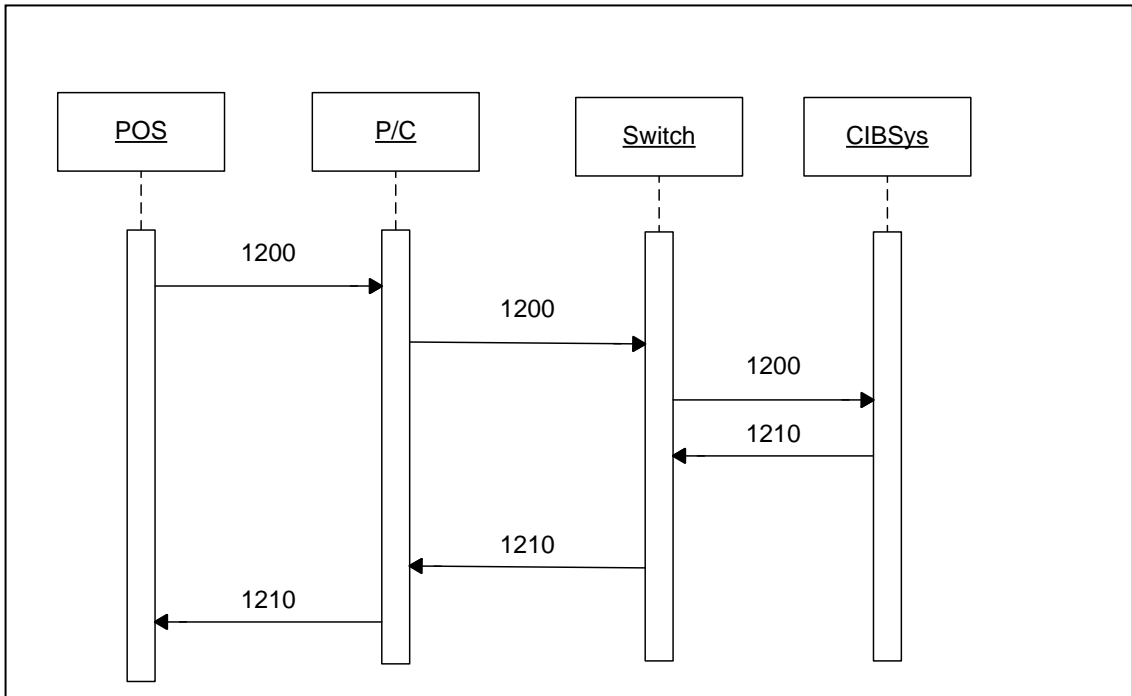


Figure 8.25: Transaction decline scenario in the payment system.

financial advice response (1230) to the switch to confirm the completion of the financial transaction.

Transaction Declined: this scenario covers all the scenarios where the transaction will be declined because of a problem with the card or card holder’s account. These scenarios include problems with the PIN entered in the terminal, invalid account; exceeding limits, no funds ... etc. The sequence diagram covering all of these scenarios is shown in Figure 8.25.

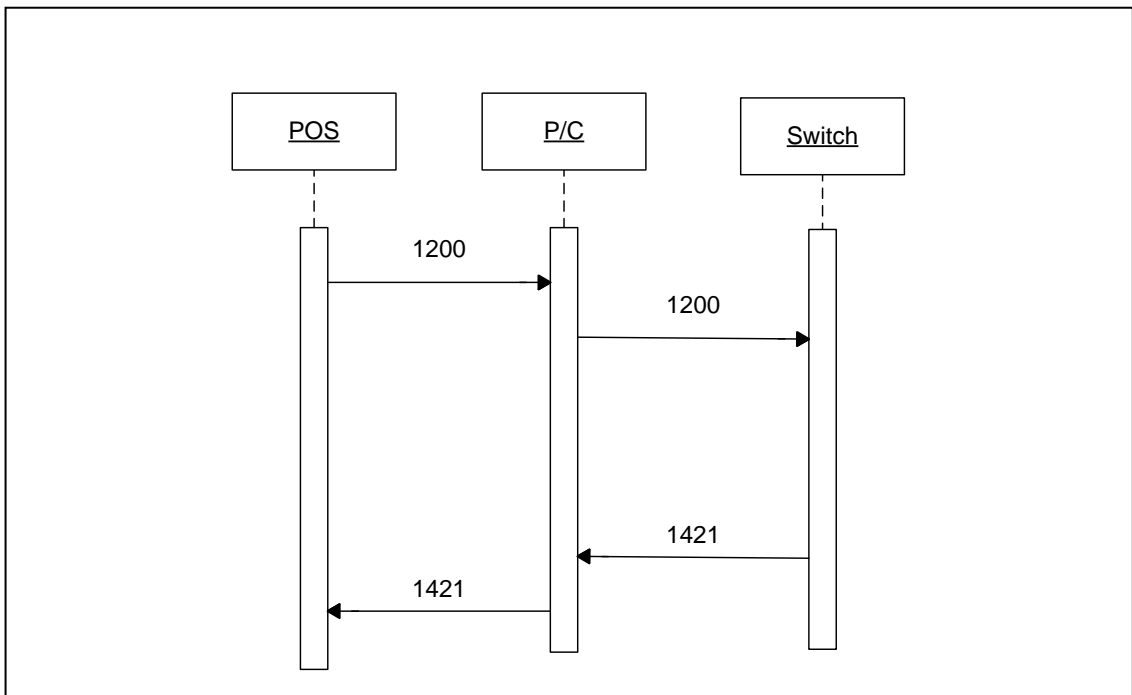


Figure 8.26: Problem with the transaction.

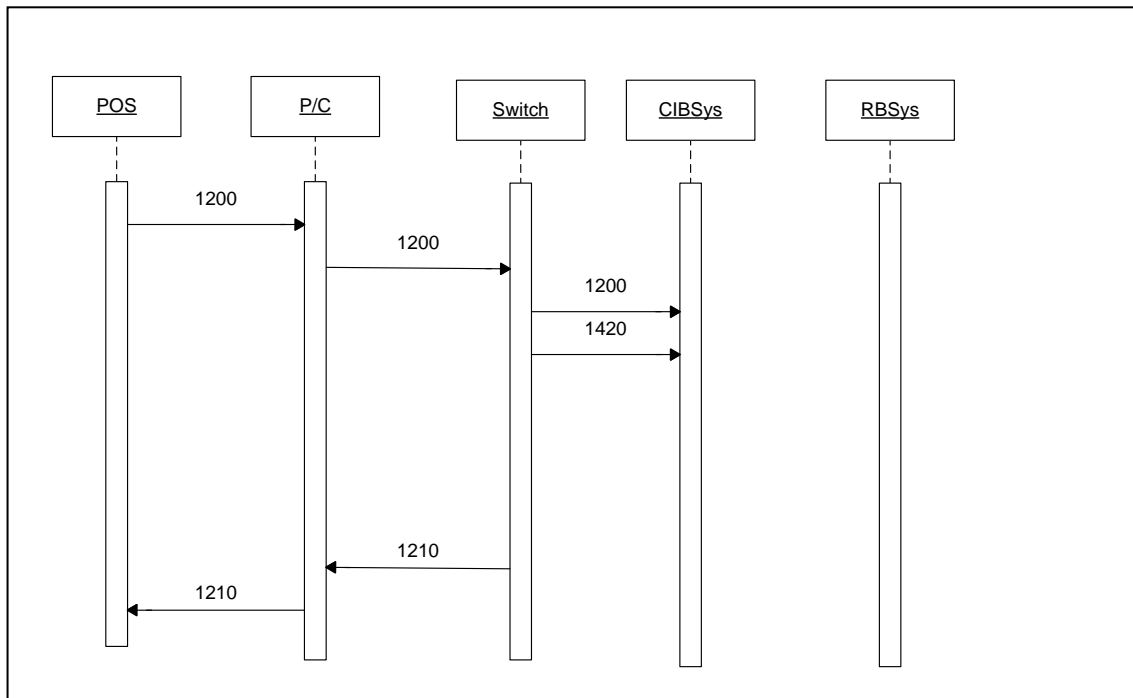


Figure 8.27: No response from the card issuer system.

The authentication of the card and card holder’s account is requested from the card issuer bank by a financial request message. This message will originate from the POS terminal and will be forwarded to the card issuer bank by the switch. The response to this financial request will arrive from the card issuer system with declaim if any of the refusal conditions accrue. The switch will then forward the response to the POS terminal and the transaction will end.

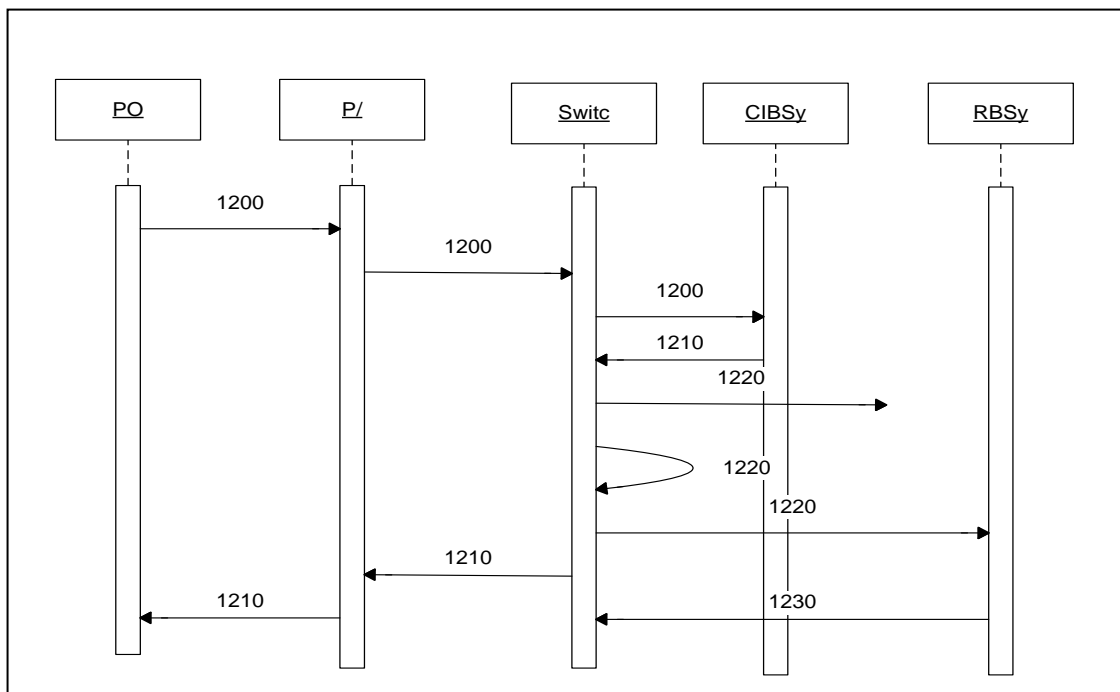


Figure 8.28: No response from the retailer bank system.

Problem with financial request: If there is a problem caused by network noise or faulty POS terminals. These problems will arise in the form of problematic financial requests. The financial requests arriving at the switch will be checked for the authenticity of the sending POS signature and retailer etc. If any problems are found, the switch will reply to the POS with an acquirer reversal advice message (1421).

No response from issuer bank: In the case that there is no response from the first financial advice sent to the issuer bank, the advice is sent again as a (1420) message. If there is no response for the second transaction, the operation will be declined.

Retailer bank time out: In the case where there is no response from the retailer bank, the transaction will be stored in the switch in a special (store and retrieve database). When the connection is resumed, the transactions will be sent to the retailer bank afterwards.

The frequency of each of these scenarios is shown in Table 8.2. These frequencies are taken from an average count of scenarios that occurred during the run of the original payment system. In this table, we can see the frequency of each of the scenarios and sub-scenarios covered by the general scenarios that we explained earlier.

Table 8.2: Scenarios frequency for the payment switch system

Scenario	Sub-scenarios	Frequency (%)
Normal transaction	Approved normal transaction	93%
Transaction declined	Invalid card, no funds, incorrect PIN, exceeds limit, restricted card, exceeds PIN retry , invalid PIN block, PIN key error, lost card	4%
Problem with financial request	Invalid merchant, no original, invalid transaction, invalid amount, invalid capture date, no from account, no to account, message format error, invalid issuer	1%
No response from issuer bank	Issuer down, invalid response code	1%
Retailer bank time out	Invalid acquirer , invalid response code	1%

System Components Demand and Network Delays

The payment switch components and their service times are shown in Table 8.3. For each of the components, we have explained briefly, the hardware and software components composing the system within the limits approved by the non-discloser

Table 8.3: Payment system's components service time and specifications

Component	Specification	Avg. Service time/ Message
Switch	The switch is running an IST switch system[122] deployed on an IBM p690 machine with 32 ways at 1.1 GHz Power4 CPUs and 100 GB RAM, with 8 processors and 26 GB RAM dedicated for POS transactions. The machine has 48 18GB disks running at 15000 RPM. The operating system on which the machine is running is an IBM AIX 5.1 maintenance level 2, Kernel 32&64 bits. The DBMS running the switch database is a Sybase ASE 12.5.0.3.	0.057sec/ message.
P/C	The Protocol converter is a 4 processor multi-threaded program dedicated to POS transactions which is used to convert the protocol from X.25 to TCP/IP and back again to X.25. The machine is located in the same switch rack.	0.008 sec/message
CIBSy RBSy	Each member bank has its own switch interface system we have taken the upper bound of the processing time between the banks systems.	0.32 sec/message

agreement signed to study this system. As the table reveals, the system's hardware is mounted on an IBM p690 rack, with the hardware for the switch and the P/C located in the same machine. The average service time for the switch and the P/C were rounded by taking the system specifications of the products used for the switch and the P/C components, and the time required to process and forward one of the transaction's messages when the system resources are fully utilised (as we are doing an upper bound analysis). The member banks' systems were calculated by averaging the result of the following formula:

$$\text{Average time} = \text{Average response time} - \text{Average network delay}$$

Although the results for the member banks are moderately diverse, we have taken the upper bound from all the banks and averaged them to represent the service time in the bank system components. Table 8.4 shows the average network latencies in the payment system. There are two kinds of networks; X.25 and the SJN. Both of these networks are explained in the table and the delays are taken from experimenting with the existing switch system.

Table 8.4: Payment system's network delays

Component	Specification	Avg. Service time/ Message
X.25	Provided by a third party telecommunication company, the network has a speed of 1MB. Represented in the queuing network as delay stations (POS_P/C and P/C_POS1).	0.3sec/ message.
SJN	SJN network is the backbone network of all services. It consists of backbone routers and switches, firewalls and L3 switches, to be utilised by any server within that network. It uses 1GB network cables between core devices and provides 100MB interfaces to end users. Some servers get 1GB interfaces depending on their needs. Represented in the queuing network as delay stations (Switch_CIBSys, Switch_RBSys, CIBSys_Switch and CIBSys Switch) .	0.135 sec/message

8.2.3 Payment System Performance Study

The main objective of the study is to investigate the throughput of the switch component in the payment system. This throughput performance measure will be compared to the throughput gained from the benchmarking experiment conducted earlier the system configuration. We will also study the effect of the P/C becoming a bottleneck in the system. This will be done by checking the utilisation of the P/C component and comparing the utilisation of the switch component when the P/C component reaches the full utilisation point. We started the performance study by designing the use-case, sequence and deployment diagrams using the ArgoUML tool. Next we extracted the XMI document representing the payment system architecture from this tool. This

document was fed to the UML-JMT wizard and used (as we saw in 8.1) to define the system’s performance characteristics (service time, delays). The performance model that was generated from the UML-JMT tool is shown in Figure 8.29. This performance model represents an open queuing network with four queues and six delay stations, each of which represents a network connection between the system’s components.

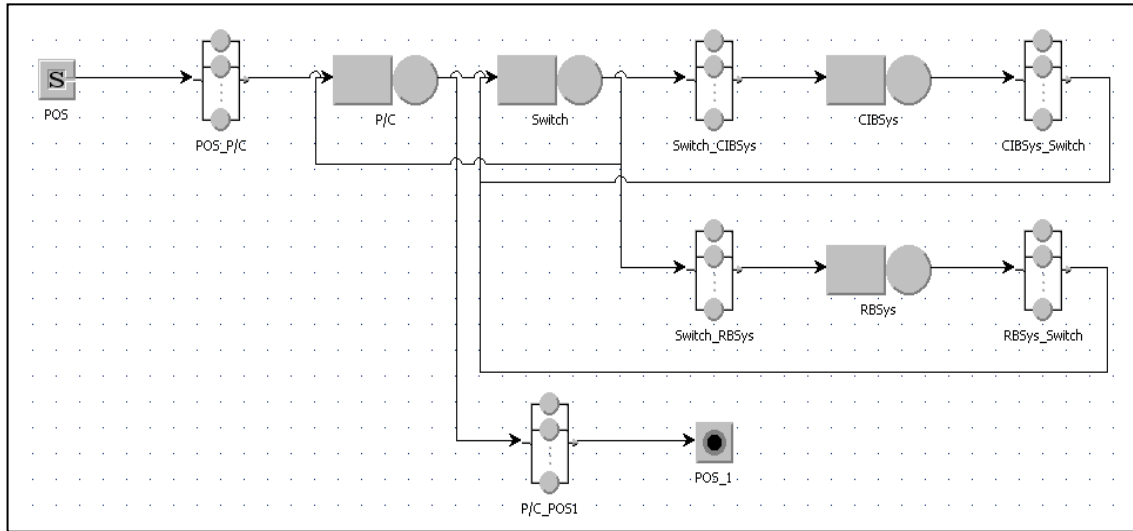


Figure 8.29: Queuing network generated for the payment system, as shown in the JMT suite.

The network is designed with five job classes, each of which represents a scenario route. These classes are characterised with the percentage tied to the frequency and the arrival rate of payment transactions to the system. We then define the performance indices that we seek to monitor during the queuing network simulation. As we explained earlier, in this performance study, we will concentrate on the throughput and the utilisation of both the P/C and switch components. The study will be designed to investigate the effect of increasing the arrival rate on the throughput and utilisation of the system. We will use the ‘what-if’ tool to design a study for increasing the arrival rate from 10 TPS to 300 TPS to monitor the system saturation point and the utilisation of the P/C compared to the utilisation of the switch.

Table 8.5: P/C component utilisation when the system load increases to 300 TPS from 10TPS.

Load (TPS)	10	42.2	74.4	106.6	138.8	171.1	203.3	235.5	267.7	300.0
Mean	0.63	0.73	0.83	0.91	1.00	1.00	1.00	1.00	1.00	1.00
Max	0.64	0.75	0.85	0.93	1.02	1.02	1.02	1.02	1.02	1.02
Min	0.61	0.72	0.82	0.90	0.98	0.98	0.98	0.98	0.98	0.98

Table 8.6: Switch component Utilisation when the system load increases to 300TPS from 10TPS.

Load (TPS)	10.0	42.2	74.4	106.6	138.8	171.1	203.3	235.5	267.7	300.0
Mean	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.95	0.96
Max	1.02	1.02	1.02	1.02	1.02	1.02	1.00	0.98	0.96	0.98
Min	0.98	0.98	0.98	0.98	0.98	0.98	0.97	0.93	0.94	0.94

8.2.4 Payment System Performance Results

As we explained earlier, the performance study is concerned with investigating the effect of the P/C with its current configuration of becoming a bottleneck, and comparing the throughput of the switch system and switch component, with the results gained from the benchmarking exercise deployed on the system. In this sub section, we will discuss the results gained from the performance study.

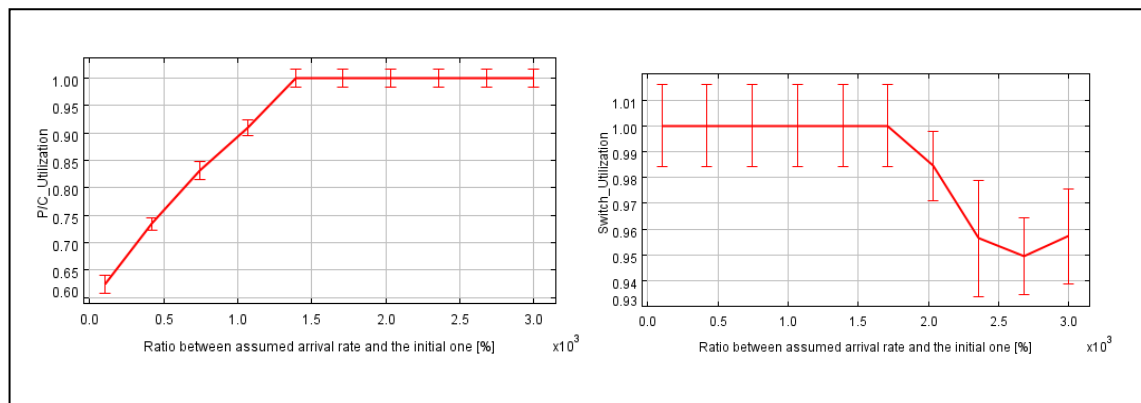


Figure 8.30: P/C and Switch components' utilisations when the system load increases to 300 TPS from 10TPS.

P/C Component effect

The effect of the P/C component will be studied by investigating the utilisation of this component and comparing it to the utilisation of the switch component which is directly feeding from the P/C. Tables 8.5 and 8.6 and Figure 8.30 show the results gained from running a simulation of the performance model representing the payment system in Figure 8.28 when the system load increases from 10TPS to 300 TPS. The results showed that the switch component was fully utilised (utilisation =1) from the early run of 10 TPS where the P/C component was only around half its capacity of an utilisation of (0.63). Figure 8.30 shows that the P/C component only reaches its full capacity when

the load reaches (138.8) TPS, which will slightly affect the switch component utilisation (0.95), but to an accepted degree.

The performance study results show that the P/C component does not cause a bottleneck in the system. This is clear from the fact that the switch component was fully utilised, though the P/C component did not reach its full capacity. Although Figure 8.30 shows a drop in the switch component utilisation and this drop occurred after the protocol converter reached its full capacity, this drop in the utilisation can be classified as a minor drop. This information can support the system designers with confidence in the current configuration of the system, in the context of design and resources, given that the switch system provided the expected throughput, as we will explain next.

Table 8.7: Switch component throughput (MPS) when the system load increases to 300TPS from 10TPS.

Load (TPS)	10.0	422	74.4	106.6	138.8	203.3	235.5	267.7	300.0
Mean	139.2	142.7	142.5	142.8	143.3	138.9	139.0	136.9	137.3
Max	141.6	144.8	146.3	146.9	148.5	140.7	149.1	144.0	145.8
Min	136.9	140.6	138.8	138.9	138.5	137.3	130.3	130.6	129.7

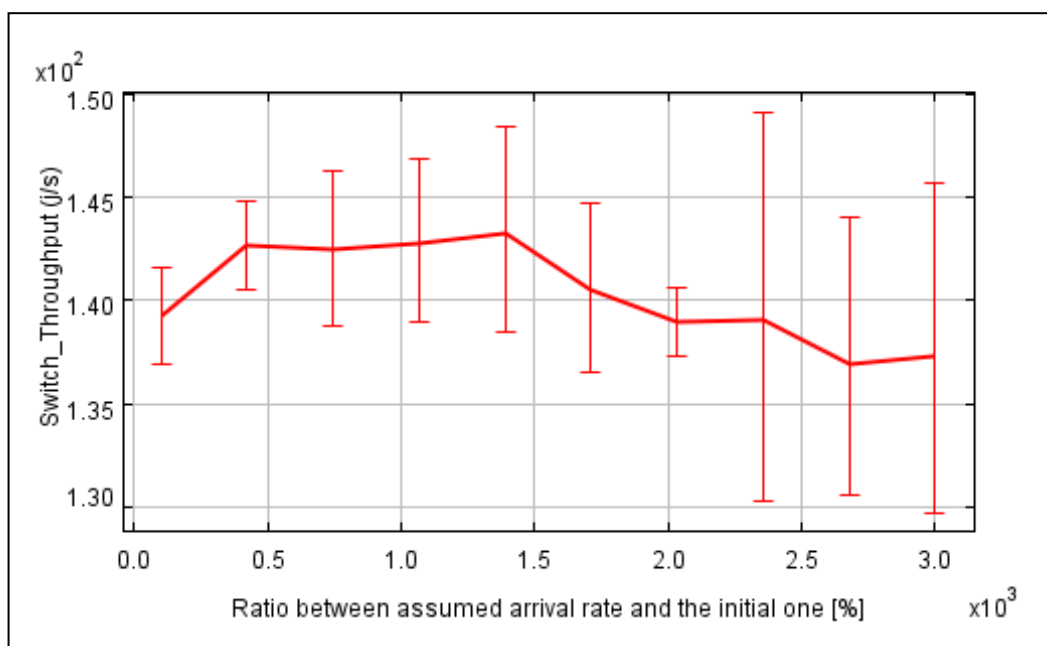


Figure 8.31: Switch component throughput when the system load increases to 300TPS from 10TPS.

Switch Component Throughput

The benchmarking exercise deployed on a system configuration similar to the payment switch configuration, with transaction generators representing the POS terminals and transaction handling agents representing the member banks systems, showed that the POS switch component is capable of handling a maximum load of (132 Message/Second). The performance results gained from the queuing network in Figure 8.29 are shown in Table 8.7 and Figure 8.31. The Figures in the table and that graph show that the switch component reaches its saturation point of 143 MPS. The graph in Figure 8.31 shows that the switch sustains an average level of throughput until the P/C component reaches its full capacity point.

We can validate the results gained from the UML-JMT tool by comparing the results gained from this performance study to the results found in the benchmarking exercise. The difference between the throughput provided by the model produced by the UML-JMT tool and the throughput gained from the bench marking test does not exceed the (7%) difference. This is acceptable since the average margin of error in non-deterministic model based performance testing for throughput, is around 10 %.

Table 8.8: *Payment system throughput when the system load increases to 300TPS from 10TPS.*

Load (TPS)	10.0	42.2	106.6	138.8	203.3	235.5	267.7	300.0
Mean	6.89	24.89	60.80	78.14	91.25	94.54	95.37	96.75
Max	7.59	25.66	62.71	81.49	94.33	97.05	97.46	98.41
Min	6.30	24.16	58.99	75.05	88.36	92.15	93.36	95.15

Payment System Throughput

As explained earlier, the potential throughput of the system was able to produce 100 TPS. The system throughput performance index calculated for the performance model representing the payment system showed that this figure can be reached by using the suggested design and system configuration. Table 8.8 and Figure 8.32 show the results of the system throughput. The results showed that the system reached a throughput of 96.75 TPS with the system still out of the saturation state. This indicates that the system with the current architecture and configuration is capable of reaching the targeted TPS rate.

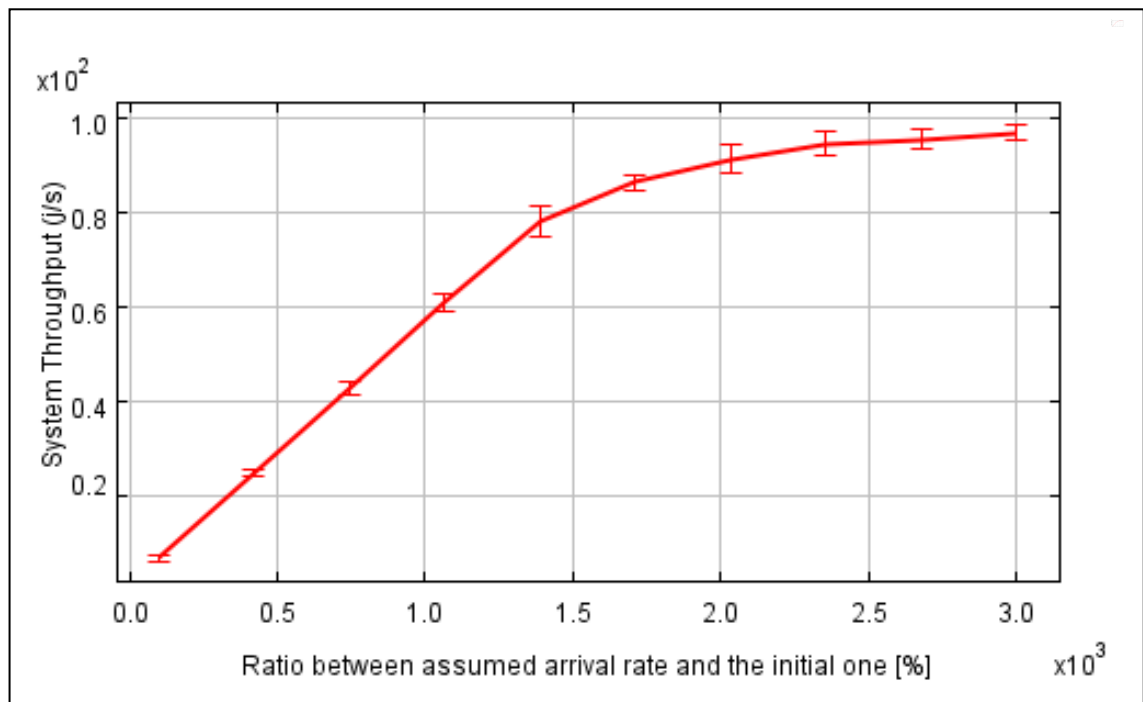


Figure 8.32: Payment system throughput when the system load increases to 300TPS from 10TPS.

8.3 Summary

The role of the UML-JMT tool in the process of validating a system's performance requirements can be summarised in papering the performance model required to conduct the performance study used for the validation. UML-JMT provides together with the analysis and experimentation tools available in the JMT suite a solution for semi-automating the performance evaluation task. This chapter is dedicated to evaluating the use of the UML-JMT tools as an aid for evaluating a system's design. We have demonstrated the UML-JMT tool in this chapter in two case studies. The first case study was dedicated to demonstrating the usability of the tool for conducting a performance evaluation experiment. This demonstration included snapshots that provide the reader with experience of the user when utilising this tool. As this tool is deploying a different method for collecting the system characterisation used to build the performance model different than the one utilised in other similar tools, we compared the performance evaluation experience for this tool and a similar tool at the end of the first case study.

The second case study was dedicated for evaluating the degree of accuracy of the performance results gained from the tool. The degree of accuracy is measured by the margin of error between the performance results forecasted by the tool for a system's design, compared to the real performance results taken from the system after it has been

developed. This case study showed that the results gained from the UML-JMT was acceptable to a valid degree as the error margin was inside the acceptable average error in non-deterministic model based performance testing. This case study demonstrated as will how this tool can be deployed in a real system context.

Non-deterministic performance evaluation methodologies were designed to provide a means of evaluating software systems designed from an early stage of system development. We have seen that the deployment of these methodologies is challenged by the complexity represented in the design and analysis of the performance models. Therefore, a range of methodologies have emerged for simplifying the performance model building process. These methodologies depend on transforming architectural models to equivalent performance models. The main goal of these methodologies is to simplify the performance model building and analysis task in order to make the non-deterministic model pass performance testing, which is part of an engineered system development process. These methodologies did not meet their goal as the non-deterministic performance testing was not a common practice in the software development industry. As one of the main components of this thesis is the UML-EQN methodology and the UML-JMT tool, we decided to conduct a qualitative validation test that will investigate the attitude of a sample of software engineers towards the methodology and the tool. This study will investigate the methodology's level of efficiency and the tool's usability. This study will also investigate the accuracy of the assumptions taken by the model transformation methodologies.

The main objective of this qualitative study is to investigate the effectiveness of the method transformation methodology by studying the level of knowledge that members of the software engineering community support for this specific paradigm of performance requirements validation, and the reasons for the lack of utilisation for this paradigm in the software development industry. Our hypothesis state that, the lack of deployment of this performance requirements validation paradigm returns mainly to the knowledge gap between software and performance engineering domains. The introduction of the UML-EQN tool was aiming to bridge this knowledge gap by introducing methods for assisting the performance study initiation, starting at gathering the performance data required for the performance study and ending with efficient, easy to use experimentation functionalities available in the JMT tool. As a part of the

objectives, we need to investigate if the cause of this knowledge gap returns to the absence of knowledge about the paradigm itself or does it return to problems in the model transformation methodology.

The study involved interviewing a group of software engineers from different sectors of the software development sectors. The study involved demonstrating two performance requirement validating tools based on the model transformation methodology one of them is the UML-EQN. The reason for demonstrating the second performance validation tool is to investigate the user's acceptance and attitude toward the method usually adopted in collecting the performance study data (UML-SPT) compared to the method adopted in the UML-EQN (PDC). The study was aiming to assess the participants' level of acceptance and their attitude toward the non-deterministic performance validation as a design aid in general and the model transformation methodology for deploying this methodology specifically. These satisfaction metrics were measured before and after the introduction of a treatment represented as workshop explaining the validation paradigm deployment using the demonstrated tools. The study also involved studying the usability of the UML-EQN tool using a standard usability test.

This chapter contains four sections. Section 9.1 will discuss the qualitative study design in detail. Section 9.2 will discuss the results and analysis of the first part of the study, which investigates the effectiveness of the model transformation methodologies in the software development process. Section 9.3 will explain and analyse the results of the UML-JMT usability test, and finally, Section 9.4 will conclude this chapter by summarising the results and outcomes of the study, and discussing the improvements suggested for the methodology and the tool.

9.1 The Study

In this section, we will explain the design and steps of the qualitative study. We have composed this section in the same format suggested in the ISO9241-11 standard format for usability reports. This section will explain the objectives, method and design of the qualitative study.

9.1.1 Objectives

The main objective of conducting this qualitative study is to investigate the efficiency of the general methodology of the non-deterministic study of systems performance, and usability of the UML-JMT tool compared to similar tools. The efficiency of the

methodology will be investigated by identifying the challenges against deploying the performance evaluation in real software system projects in the industry. These challenges are represented in the causes of the knowledge gap between software performance and system engineering, and the availability of tools which support software engineers in automating the build and analysis of the required performance models. The study also investigates any other factors that may cause disregarding the performance evaluation at the system design stage, such as the system size and the ability to interpret the resulting performance indices gained from the performance study. The study will investigate the usability of the UML-JMT tool from the perspective of learnability, effectiveness and user satisfaction. The learnability factor will investigate knowledge gained by the user in the software performance engineering context after learning to use the UML-JMT tool. The effectiveness factor tests if the functionalities and results provided by the tool reaches the users expectations. The satisfaction factor will test the tool's ease of use and appearance.

9.1.2 Method

The experiment was composed of four phases. In two of these phases, the participants were involved in a structured interview. A structured interview is conducted with a moderately open framework which allows for a focused, conversational, two-way communication[123]. They can be used to both, give and receive, information and this helps in gaining information as well as providing explanatory knowledge to the participants. Structured interviews can be used to acquire specific quantitative and qualitative information, obtain general information relevant to specific issues, and gain a range of insights into specific issues[123]. Between the two interviews, the participants were involved in a workshop that discusses essential background knowledge of software performance engineering terminology and introduces the participants to the UML-JMT and XPRIT tools[124]. Afterwards, the participants were given the opportunity to use the UML-JMT tool to execute a scenario example which was explained in the workshop. After a participant executes this scenario, he/she will be asked to provide suggestions to improve the tool and evaluate its usability, using the standard IBM computer systems usability questioner (CSUQ)[125]. In this section, we will provide information regarding the experiment environment which includes information regarding the participants and the context of the experiment.

Table 9.1: *the business sectors the participants work in.*

Public service sector	King Saud University(KSU) Ministry of Finance(MoF), Ministry of Defence and Aviation(MoDA), Ministry of Water and Electricity(MoWE)
Banking sector	Saudi Arabian Monetary Agency Banking Technology Department(SAMA-BTD) Institute of Banking (IOB) Al-Tawiniya
Telecommunication sector	Mobily
Software Warehouses	Chip CS AlFisaliah ITS

Participants

The subjects chosen for this study represent software engineers with different academic and professional backgrounds. The participants were chosen from a range of sectors which heavily employ software systems. Table 9.1 shows the sectors and the organisations in which the participants work. The organisations were chosen to be in Saudi Arabia. We believe that the result of the study cannot be affected by chronological (age), geographical (location), or cultural factors. This comes down to the strong belief that software development cannot be affected by such factors. The study was conducted on 21 participants with an average experience of around 9 years, ranging from 2 to 27 years of experience. In a question to describe the magnitude of the largest project they were involved in, in terms of budget, number of components, time and man power. The participants were given a five scale measure to describe this project where 1 represents a project with less the 3 components, with a budget of < 10K\$, manpower of <3 personal and scheduled < 3 months, and 5 represent a project with > 15 components with a budget of >10M\$, manpower of >30 personal and scheduled > 24 months. The participants scored an average of (3.63) with scores ranging from 1 to 5. This indicates that the participants in this study represent an acceptable sample of software engineers with time and practical experience and who represent different sectors of the industry. Choosing the right sample size is essential for any qualitative usability study, as it determine the accuracy of generalising the outcome of the study. The recommended sample size defined in [125] can be calculated as following:

$$\text{Acceptable Sample Size} = 5 \times \text{number of scenarios} \times \text{number of tasks/scenario}$$

As we have a single scenario with 4 items, 20 participants are an acceptable sample size. The full information of the participants can be found in Appendix C, Table C1.

Scenario of the Experiment

As explained above, the experiment will involve a workshop that explains the terminology of software performance testing in the design stage. The workshop will use the example of the information retrieval system explained in Chapter 8 as an example of the two tools explained in the workshop. At the end of the workshop, the subjects are given the opportunity to evaluate the usability of the UML-JMT tool. They will be provided with two XMI files; one containing the UML diagrams for the information retrieval system, and the other containing an invalid UML diagram (there are three types of problematic XMI files). This file will be used to show the user the error reporting function available in the tool. The scenario in the experiment is to make a stress test for the architecture selected for the system. This task consists of five tasks:

Task 1: The participant will be asked to use the UML-JMT wizard to perform the model transformation task, as explained in the workshop.

Task 2: The user will be asked to open the resulting performance model in the JMT suite - JSIMgraph tool, and to select the objective performance indices. In this scenario, it will be the system's throughput.

Task 3: The user is asked to use the 'what-if' tool to inspect whether the increase in the number of users from 10 to 1000 will affect the system's throughput. The user will be asked to identify the saturation point on the throughput graph.

Task 4: The user will be asked to change service time on some of the service centres and investigate how this will affect the throughput.

9.1.3 Experimental Design

The actual study is composed of four stages. Two of them are structured interviews and one will consist of a workshop that will cover software performance engineering terminology and a number of methodologies similar to the UML-EQN methodology, together with the methodology under study. The interviews are designed to examine the participant's knowledge before and after providing the subjects with knowledge about software performance engineering and methodologies. The steps of the study and the activities conducted in each step are discussed in this section:

Step1 - Setting the par: In this step the subject will be asked a set of questions that will determine the level of knowledge that they have on the software performance engineering field, and how much experience (academically or professionally) they had before taking part in the workshop. The set of questions and the rationale for each of these questions is shown in Table B1 in Appendix B.

Step 2 - Providing the knowledge (workshop): The subjects will be provided with basic knowledge covering areas of software performance engineering terminology. This will help them to understand the importance of the software performance study and to be able to comprehend the methodology under study. The knowledge will be provided in a workshop consisting of three sections. The activities in each section are defined as follows:

Section 1: Explaining Performance Engineering

- Definition of software performance studies and their importance
- Functional requirement vs. non-functional requirement validation
- Performance studies: modelling vs. simulation
- Modelling paradigms
- Inputs and outputs of a performance study

Section 2: Explaining Model Transformation Methodologies

The subject will be given a brief description of three methodologies for conducting performance studies, two of which are UML based. The methodologies are chosen to have similarities because of the time limit and not to confuse subjects who are new to the area. Two of the methodologies, including the one under study (UML-EQN), adopted the SPE (Software Performance Engineering) framework. Therefore, the SPE methodology should be explained first. The second methodology is called PRIMA-UML, which is based on the SPE framework. The last methodology will be the one under study, UML-EQN. The criteria we explained in Chapter 4 for evaluating model transformation methodologies will be explained to the user.

Section 3: Explaining the Tools

XPRIT and UML-Tools will be explained using the information retrieval system example from 8.1.

Step 3 - Collecting Results: In this step, the subject will be interviewed again to ask them questions that will determine the level of knowledge that the subject has gained on the software performance engineering field, and which of the methodologies and tools is most convenient for the software performance test task. The questions to be asked are shown in Table B2 in Appendix B.

Step 4 - Testing the System: The participants will be asked to execute the scenario explained in the previous section. This will prepare them to answer the usability questionnaire in Step 5.

Step 5 - Evaluating UML-JMT Usability: The users will be asked to answer the IBM CSUQ questionnaire for evaluating the usability of the UML-JMT system.

9.2 General Methodology Effectiveness Analysis

One of the goals of this study is to investigate the causes of the infrequent deployment of non-deterministic software performance testing in the industry. A common claim for this in major publications comes back to the knowledge gap caused by the non-deterministic heavy-weighted mathematical and statistical terminology used in software performance evaluation. As discussed in the early chapters of this thesis, the introduction of model transformation methodologies aimed to bridge this knowledge gap by black-boxing the performance model building and analysis tasks which will make the performance testing process a semi-automated task. All of the literature discussing the lack of non-deterministic performance testing only speculated on the reasons for this problem. We have decided, as a part of this qualitative study, to make grounds for our claims by investigating the attitude of software engineers toward non-deterministic software performance testing. This will be done by asking the participants their opinion on non-deterministic performance testing, with and without model transformation tools, and before and after providing the participant with basic knowledge of performance testing.

The effectiveness of the model transformation based performance testing was explained to the participant through the use of the UML-JMT and XPRIT tools. The study was organised as a structured interview, as we explained in the previous section. This interview was divided into two parts. In the first interview, the participants were asked questions about their level of experience and their attitude towards performance testing in general, and non deterministic performance testing. The participants were then

introduced to model transformation performance testing with the UML-JMT and XPRIT tools. Next, the participants were asked about their knowledge and confidence level to conduct a performance study using the same methodology and if they would use it in future projects. The questions for the two parts of this structured interview are shown in tables B1, B2 of Appendix B. Most of the questions in this interview are based on a scale that measures the participant's perspective on articulated issues concerning performance testing and the tools offering the performance testing task. In this section, we will discuss the results of the interviews by analysing the participant response to each of the questions.

9.2.1 Pre-orientation Interview Analysis

In this section, we will analyse the interviews conducted before providing the participants with the orientation workshop discussed in the previous section. This interview is designed to investigate the participant's level of experience and knowledge in the context of UML and software performance testing. We previously used the level of experience in 9.1.2 to prove that the participant sample covers a broad spectrum of software engineers. We will analyse the questions in Table B2 as groups representing the experience, UML knowledge and performance engineering knowledge.

Experience

The experience level is defined in this interview by three main factors; the magnitude the participant is involved in software development, the nature of this participation from the context of development stage (analysis, design, development, test or all) and the size of the biggest project the participant has been involved. These factors are used to ensure that the sample involved in this study represents software engineers from different levels of experience and academic and industrial backgrounds. The first factor is gained from the participant's response to Q1 of the interview stating "have you been involved in software development (1 for very few times and 5 for majorly)" the average score gained for this question was 4.33, with a maximum score of 5 and a minimum score of 3. The participants who chose 3 are more involved in the support and maintenance of software systems. The second factor is represented in Q2 of the interview stating "In what stage are you usually involved? (1 - Analysis; 2 - design; 3 - development; 4 - test; 5 - all)". 76 % of the participants said that they are usually involved in all of the development stages, where the rest of them usually involved in analysis and testing. The

third factor is concerned with the size of the project they were involved in, as discussed already in 9.1.2. The results gained from these factors represent an indication that the participants chosen are a part of the targeted sample of software engineers, as we explained earlier.

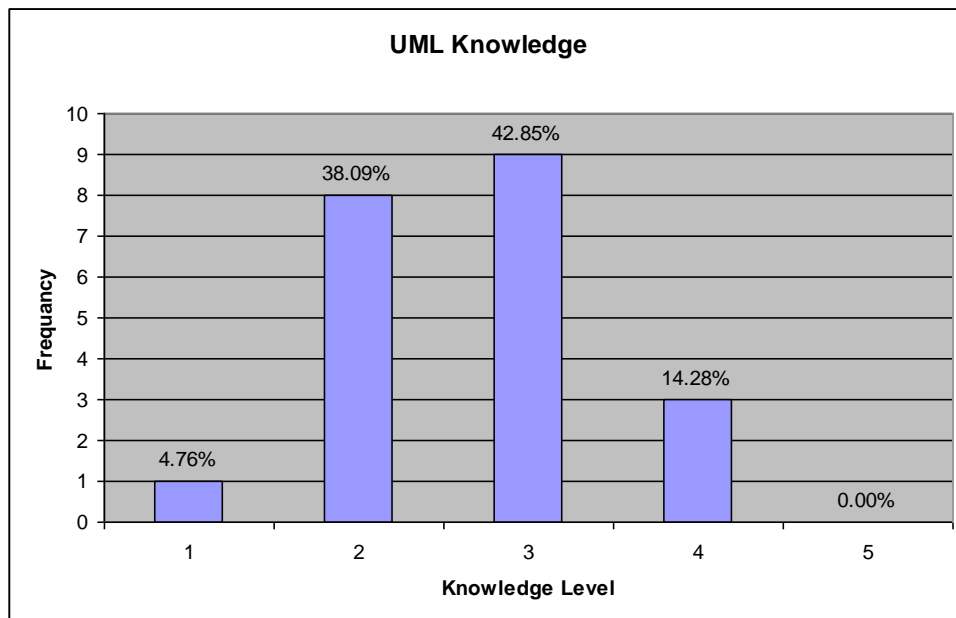


Figure 9.1: The UML knowledge frequency graph indicating the participant's level of knowledge and usage.

UML-Knowledge/Usage

The main model transformation methodologies for non-deterministic system's performance testing adopted UML modelling notation as the base model used to extract performance model used in the performance test. This was based on the suggestion that UML is the standard modelling notation widely used in the industry for development and documentation. As part of this study, we wanted to investigate this suggestion by asking the participant about their level of knowledge and usage of the UML standard. We asked the participants the question "Describe your knowledge/usage of UML (1 - no knowledge; 2 - learned it but never used it; 3 - use it occasionally; 4 - commonly use it; 5 - used it in all the projects I am involved in)" to investigate this factor. Figure 9.1 shows the frequency of each of the answers provided by the participants.

The result gained from this was unexpected as around 82% of the participants have either never used UML in development or documentation, or only used it occasionally. A number of the participants who had used the UML notations in some of the projects

they have participated in, responded by “they only use UML when the notation is specifically requested in the project specifications” [RUH03, 04, 17] and when asked about the reasons for not using the UML notation, most of the participants put this down to the time required to compose these models which represents an overhead in time resource. 14.28% (3) of the participants indicated that they use UML commonly in the project they are involved in. When we returned to these participants’ information they were found to work in the same organisation (MOF). This indicates that the organisation development policy is the main reason for these participants to deploy UML. This factor partly affects the effectiveness of the model transformation methodology which assumed that UML is a standard notation commonly used in software development. This effect was reflected in the usability effectiveness factor, as we will see in the next section.

Performance Engineering Knowledge/Usage

One of these study objectives is to investigate if one of the reasons for not deploying non-deterministic performance testing in the industry comes down to the lack of knowledge about this performance testing paradigm. We will investigate this assertion by asking the participants about their knowledge and experience in non-deterministic model based performance testing. This will take place before and after providing the user with information about this paradigm in a concentrated workshop. We asked the participants this question “Describe your knowledge of software performance engineering study (1- I’ve just heard about it; 2 - I heard about it but could not use it; 3 - I have heard about it but would not use it; 4 - I have used it several times; 5 - I commonly use it)” to clarify the knowledge and experience level they have for this performance testing paradigm. Figure 9.2 shows the distribution of the answers provided by the participants. 57% of the participants replied as they had never heard about this performance testing paradigm before, whereas 33% replied that they have prior-knowledge of this paradigm but have never used it before because of the difficulties they faced due to the complexity of the paradigm. 5% of participants said that they know this paradigm but they would not use it as “the high cost of this paradigm would not make it efficient with the projects they were involved in” [RUH12], another 5% said that they have used non-deterministic model based performance testing before but they would not use it again due to “the high cost, low accuracy factor” [RUH03]. The source of knowledge of the participants who have prior knowledge of non-deterministic model based performance testing is mostly from non-academic

resources, and only one participant said that he heard about this paradigm from his academic background.

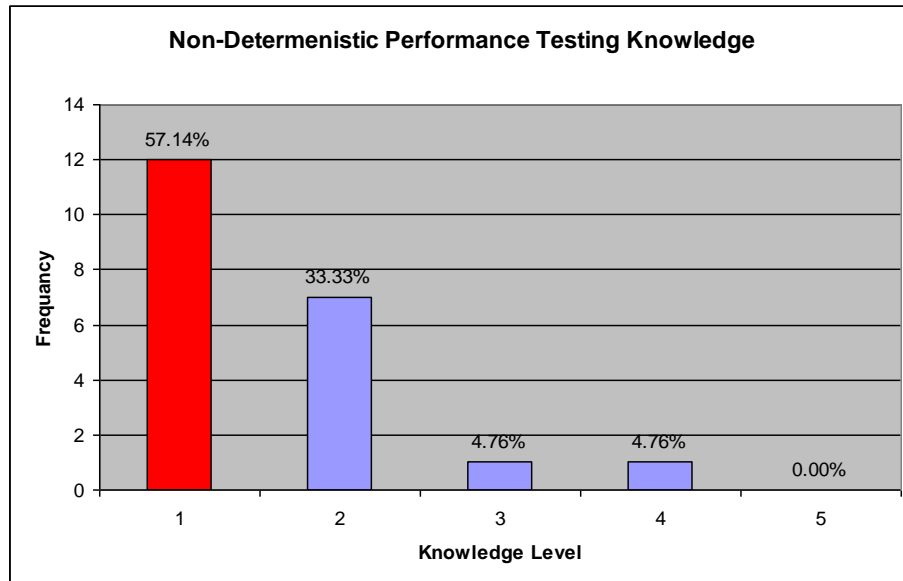


Figure 9.2 *The non-deterministic performance testing knowledge/experience frequency graph indicating the participant's level of knowledge and usage.*

This gives an indication that one reason, which may cause the lack of knowledge about this paradigm, comes down to the shortage of academic programs covering this paradigm. The participants were asked “Based on your knowledge, how important do you deem software performance engineering study to be? (1 - not important; 2 - it can be included in the testing phase; 3 - good practice for some projects; 4 - important for some projects; 5 - essential for all projects)”, to measure their attitude toward testing performance during the development of a software system. Figure 9.3 shows that 76% of the participants agreed that software performance should be included in the software development process, whereas 14% of them thought that it is a good practice, but is a low priority. One participant thought that the current method of including performance testing in the testing phase is the best practice. One participant thought that performance testing is an overhead in any software project budget. The participants were asked the current approach used to test the performance, if any. The question was “In any of the previous projects you have participated in, has a performance study been conducted in this project? (1 – none; 2 - real system test; 3 - spreadsheet; 4 - simulation; 5 - benchmarking)”. Figure 9.4 illustrates the percentage of techniques used for performance testing, as per the last question. .

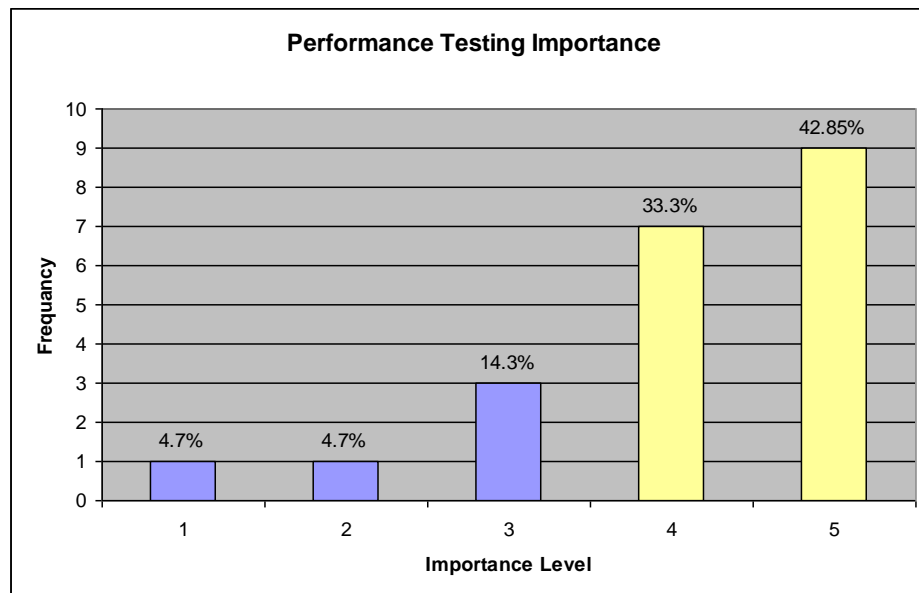


Figure 9.3: Performance testing importance frequency graph indicating the participant's level of importance

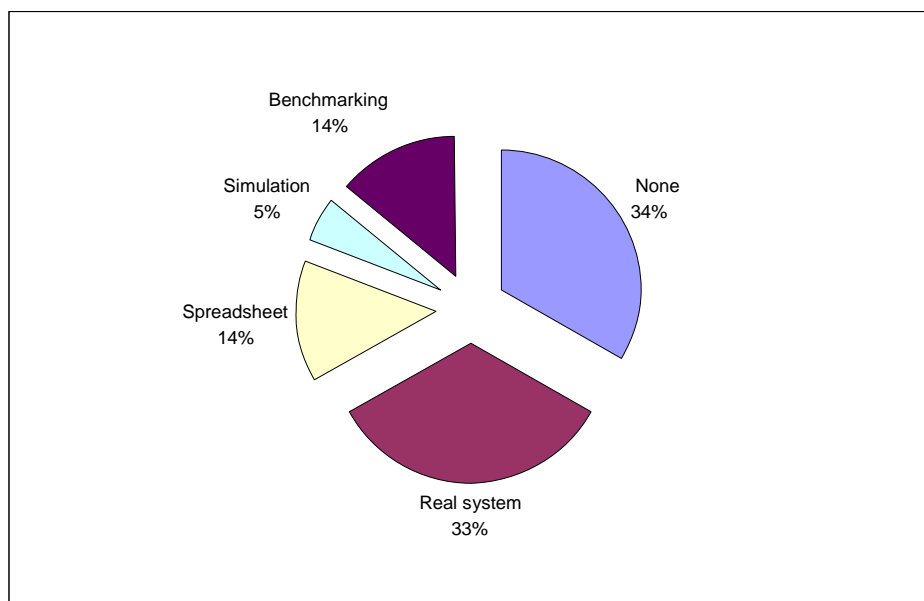


Figure 9.4: Performance testing techniques used by the participants

9.2.2 Post-orientation Interview Analysis

In this section, we will analyse the interviews conducted after providing the participants with the orientation workshop. This interview is designed to investigate the participant's level of knowledge and confidence with respect to conducting a model based software performance test. We will also discuss the participant's reaction to the two tools presented in the workshop and whether they think they could be utilised in future projects. Finally, we will investigate the importance of using standard modelling notation as an input in performance testing tools. We will analyse the questions in Table

B2 as groups representing the performance engineering knowledge and confidence, tools evaluation and importance of standard notation.

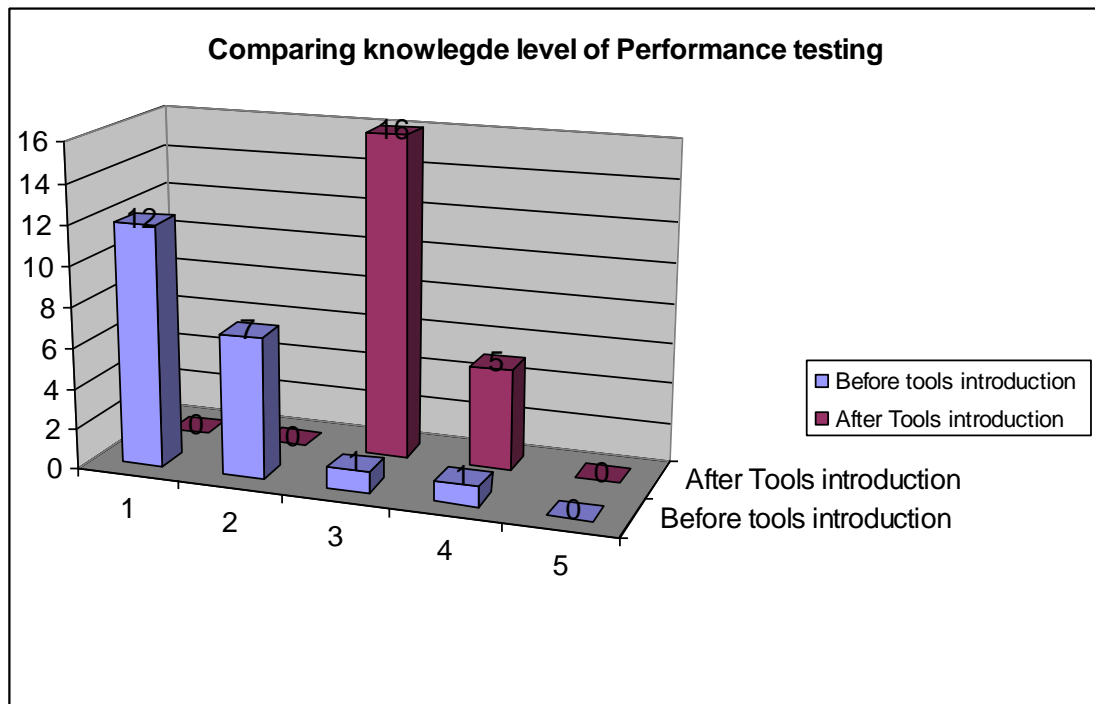


Figure 9.5: Comparing knowledge levels before and after the participants are provided with performance testing workshop

Knowledge increase and confidence

As we discussed earlier, we want to compare the knowledge level of the participants before and after introducing the basic terminology of non-deterministic model-based performance testing and the tools designed to simplify the testing process. At the end of the workshop, we asked the participants the question, “Describe your knowledge/experience in software performance engineering study? (1 - I still do not understand this paradigm; 2 - I understand it but could not use it; 3 - I think I have the basic knowledge to conduct a study; 4 - I think I have the necessary knowledge to conduct any performance test; 5 - I knew it already)”. Figure 9.5 shows a comparison of the participants’ replies to the corresponding question asked before the workshop. 76% of the participants thought that they have the necessary knowledge required to conduct a performance test and the system design level, whereas the other 24% thought that they had all the necessary terminology they need to conduct a performance test. To test the confidence of the participants when using the tools for conducting performance tests, we asked them, “How confident are you in your current knowledge of software performance study with the provided tools? (1 - not confident; 2 - need more background knowledge; 3 - I can conduct simple performance tests with assistance; 4 - I

can conduct simple performance tests without assistance; 5 - I can conduct any performance tests)”. Figure 9.6 shows the results gained from this question. The results showed that 38.1 % of the participants needed more background information to conduct the performance testing task. When asked about the information required, they needed one or more of the following:

- 1- Information about the UML use-case, sequence and deployment diagrams and the tools used to model and extract the XMI format for them.
- 2- Information regarding analysing and interpretation the performance results from design decisions.
- 3- Information about queuing networks.

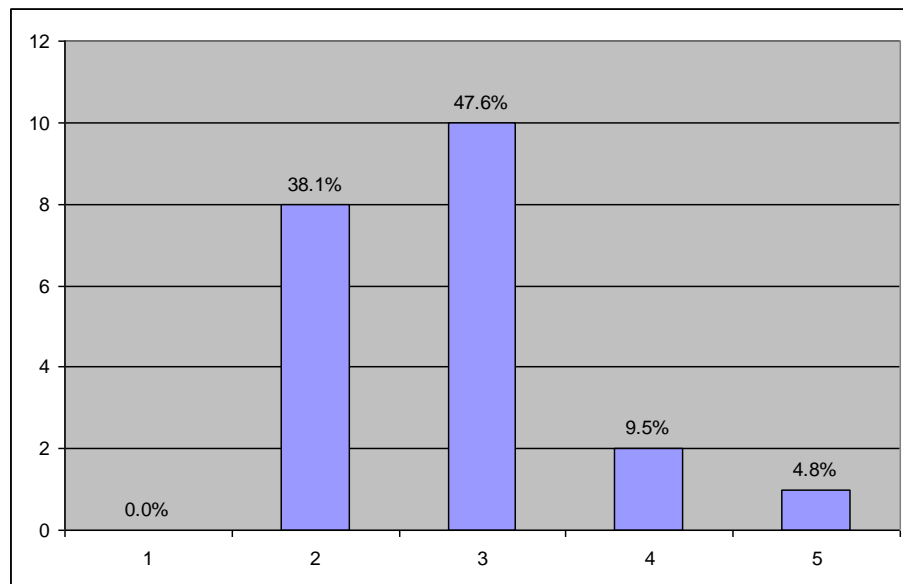


Figure 9.6: Confidence level of the participants to conduct a performance testing task using the suggested tools

47.6 % of the participants thought that they could conduct a performance study but with assistance, particularly in the area of interrupting the performance indices. 9.5 % of the participants were confident enough to conduct the study and utilise its results to make design decisions. We noticed that the participants who said that they needed more details were the ones who did not have any background knowledge about model based performance testing, although some of them chose the third answer. The participants who were confident to conduct the study tended to have some background in the discussed paradigm. This gives an indication that if we provided the participants with

more knowledge regarding performance indices, analysis and their relation to design, then they could have had more confidence to use the model based performance testing paradigm.

Importance of practice and subsequent utilisation of the tools

We asked the participants the same question about their opinion on non-deterministic performance testing, taking into account the existence of tools like UML-JMT and XPRIT. We asked the participants the same question that we asked them in the first interview, which was, “Based on your current knowledge, how important do you consider software performance engineering study to be (1 - not important; 2 - it can be included in the testing phase; 3 - good practice for some projects; 4 - important for some projects; 5 - essential for all projects)”. We found that the percentage of participants agreeing that performance evaluation is an important task to be included in most of the software projects, increased from 76.15% to 85.71%, and the participants who thought that the performance test is important in some projects changed to only 9.5%. Only one participant thought that it is a good practice. We noticed that none of the participants thought that non-deterministic software performance testing is not a good practice after they were introduced to the assisting tools. We asked the participants about the possibility of utilising the tools demonstrated to them in future projects and all of them agreed that these tools can be utilised for performance evaluation of software designs, but only on large scale and complex projects.

Comparing the performance assistant tools

As discussed previously, the introductory workshop provided for the participants included the demonstration of two tools, which provide assistance to software engineers in conducting performance testing. These are UML-JMT and XPRIT tools. Both of these tools were demonstrated in the same case study. Part of this qualitative study was to compare the UML-JMT tool to the nearest tool related to it, in terms of functionality and methodology deployed. This is the XPRIT tool. We asked the participants their opinion on these two tools by giving a score on the degree that they think they will be using or recommending any of the tools in their future projects. They were asked to provide a score out of 5 (1 - will not endorse or use, and 5 - will definitely recommend or use this tool). The scores gained for these two questions showed that the average score for XPRIT was 2.33 (46.7%), whereas the participants gave the UML-JMT tool an average score of 3.87 (77.3%). This is an indication that the participants preferred the

UML-JMT tool. They justified their decision for recommending the UML-JMT tool because it provided them with an easier approach for providing the performance characterisation data required for the test using the performance data card gathering wizard.

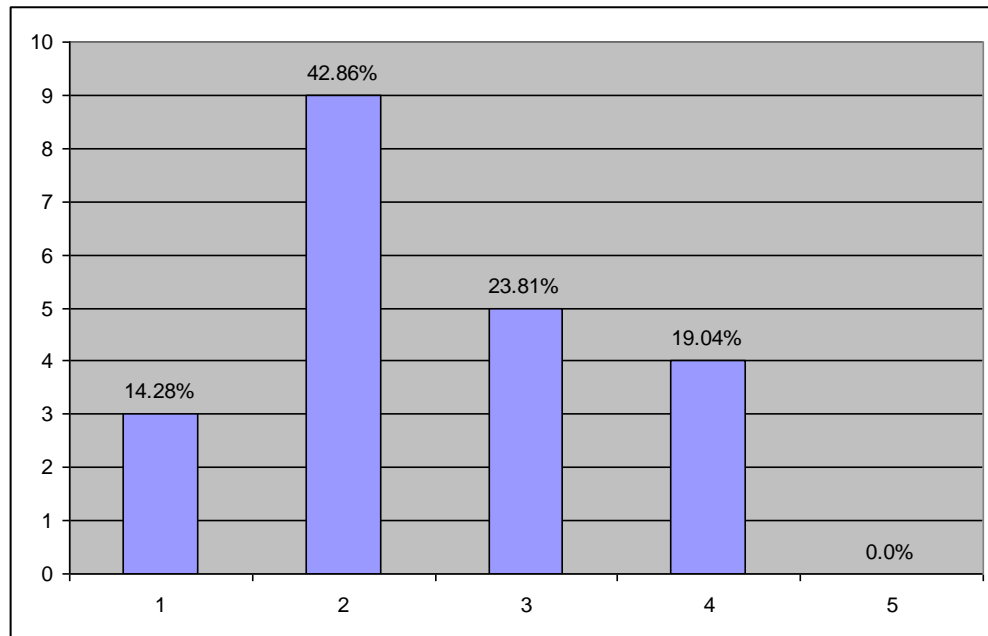


Figure 9.7: Participants' attitude toward learning new software modelling paradigms to be used in performance evaluation or other NFR verification tasks

Importance of Standard Notation

To investigate whether the users had any prior knowledge in the UML standard and the adequacy of updating the UML standard or defining a new modelling notation for a tool capable of providing assistant in performance or other NFR verification tests, we asked the participants the following question, “Are you willing to learn new software modelling paradigms to be used in performance evaluation or other NFR verification tasks”? (1 – no; 2 - yes, if it will provide accurate results and other NFR verification tests; 3 - yes, if it provide more readable verification tests; 4 - yes, if it provides more accurate performance test indices; 5 - yes, if it is part of a large CASE tool).

As Figure 9.7 shows, most of the participants (42.86%) indicated that they are willing to learn a new modelling notation if it was supported by a tool that will provide a verification test for all or most of the NFRs. 23.81% of the participants concentrated on the readability of the results gained from the tools, even if the tool is not comprehensive for all the NFRs. 19.04% concentrated on the accuracy of the results gained from the

tool. 14.28% of the participants did not agree to use a tool with a modelling notation which requires learning. This indicates that the participant's order of requirement for an assisting NFR verification tool is firstly, to be comprehensive, secondly, for it to have readable results and thirdly, to provide more accurate NFR indices.

9.3 Usability of the UML-JMT tool

According to the ISO 9241-11 standard [126], the usability of a software system is defined as “the context to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”[126]. *Effectiveness* defines the degree of accuracy and completeness that the system users achieve from the use of the system. It relates to the ease of use of the tested system and how the required tasks were achieved[126]. *Efficiency* relates to the level of effectiveness achieved in relation to the quantity of resources expended. This is usually measured by the mean time required by a user to perform a task[126]. Satisfaction is defined as the freedom of discomfort and positive attitudes toward the use of the product. *Satisfaction* describes the user's subjective response when using the product. The key factors that should be taken into consideration when testing the satisfaction of a system are the user acceptance of the product and the ease of use[126]. In this study, we will only concentrate on the effectiveness and satisfaction of the system, as the context of the study concentrates on the ability of the user to perform the task regardless of the time required to complete the task, this comes down to the nature of this task, which can be classified as infrequent.

9.3.1 Usability Metrics

The user satisfaction factor can be tested using a number of methods. The most common method for testing it is the use of standard usability questionnaires which are answered by system users to record their subjective reaction toward using the system. There are a number of standard usability questionnaires, such as ASQ[127], PSSUQ[125], QUSI[128], SUMI[129], and CSUQ[125]. In this user satisfaction test, we chose to use the IBM Computer System Usability Questionnaire (CSUQ) as it provides an overall satisfaction indicator, and the fact that CSUQ is recommended for non-laboratory setting tests[125]. The CSUQ is a 19-item questionnaire (See Table B3, Appendix B) designed for the purpose of assessing user satisfaction with the computer system under study. The items in CSUQ are 7-point *likert* scales. The likert scale is designed to measure a user's attitude or reaction by quantifying subjective information[126]. The CSUQ scale is anchored at the end points with; strongly disagree (1) and strongly agree (7). The CSUQ

has four score-metrics that consist of the average scores to responses in a group of questions which represent the metrics measured by these questions. These score metrics are; overall score, system use score, information quality score and interface quality score. The overall score reflects a comprehensive index of the degree of satisfaction for the system. The other scores indicate the degree of learnability, adoption and ease of use of the system. The effectiveness factor matrix is also covered in the CSUQ as item 4 and 5 of the questionnaire quoted, “I am able to complete the suggested work quickly using this system” and “I am able to efficiently complete the suggested work using this system” can be used to measure the user’s ability to complete the functionality of the system, along with the percentage of the participants completing the test scenario discussed in 9.1.2.

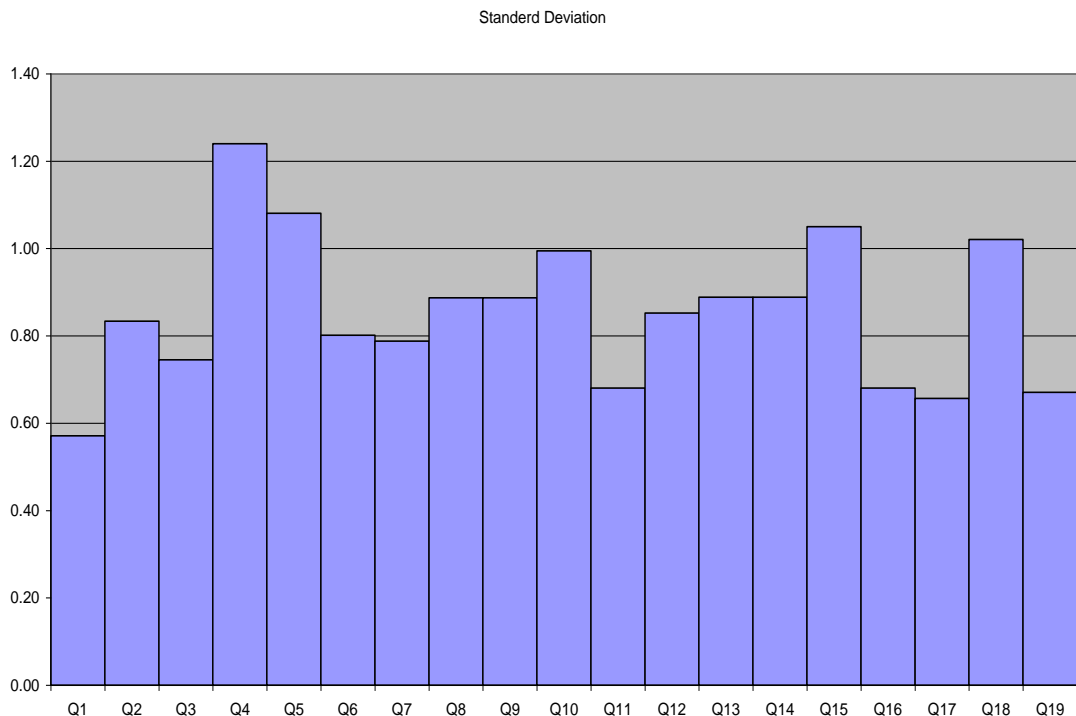


Figure 9.8: Standard deviation graph for the response of the participants in the usability study

9.3.2 Results

Table C2 in Appendix C shows the scores provided by the participants of the usability test to measure their satisfaction factor. As mentioned above, we have chosen the CSUQ test for this section of the usability test. Table C2 describes the individual scores given for each question in the questionnaire along with accumulative statistical results needed for the analysis of this questionnaire. The second last row of the table shows the mean score given for each of the questions, and the last row shows the percentage this score represents on the overall satisfaction scale of 7. We chose to call it a satisfaction scale

as the questions are designed as positive articles describing the user's attitude towards the system. We only can claim that the mean score for each question represents the general feeling that the users have for the system if there is no significant difference in the variance between individual scores for each question.

Table 9.2: Satisfaction metrics and rules for calculating the accumulative score for them.

Metrics	Description	Representing Questions
OVERALL	Overall subscale, indicating the overall satisfaction factor by averaging the satisfaction scores of all the questions.	Q1-Q19
SYSUSE	System use subscale, indicating the degree of satisfaction for using the system covering the user's attitude towards the overall satisfaction, ease of use, how easy it was to learn the use of the system and the efficiency in terms of time and productivity.	Q1-Q8
INFOQUAL	Information quality subscale, indicating the user satisfaction with the organisation and comprehensibility of information in the system. This information includes on-screen messages, error messages and documentation.	Q9-Q15
INTERQUAL	Interface quality subscale, indicating the user's satisfaction with the GUI in the context of use and appearance.	Q16-Q18

We have calculated the *Standard Deviation* for scores given for each of the questionnaire's items. The standard deviation is a measure to test the range of variation among data sets from the mean value[130]. Figure 9.8 Shows that the standard deviation ranges from 0.57 to 1.24, which means that the mean value of each of the questions represents the general score, with a difference ranging from 8-17%. This level of variance is acceptable for measuring the attitude of users towards a software system, bearing in mind that the satisfaction factor includes factors that depend mainly on the individuals, such as the user interface. The relatively small standard deviation results

allow us to use the mean value of the scores of each question as an indicator of the general attitude towards the question.

9.3.3 Analysis

The IBM CSUQ defined four main metrics to define satisfaction. These are; OVERALL, SYSUSE, INFOQUAL and INTERQUAL. These metrics can be measured by accumulatively averaging the scores representing the users' reaction to a set of questions to determine the satisfaction factors these questions represent. Table 9.2 explains these factors and the set of questions representing the accumulative score for these metrics. Figure 9.9 shows the average scores of each of the questions and the accumulative scores for each of the satisfaction factors, presented as a percentage of the 7 scale measure used in the questionnaire. Table 9.3 provides the individual usability sub-scale measures calculated for each of the participants.

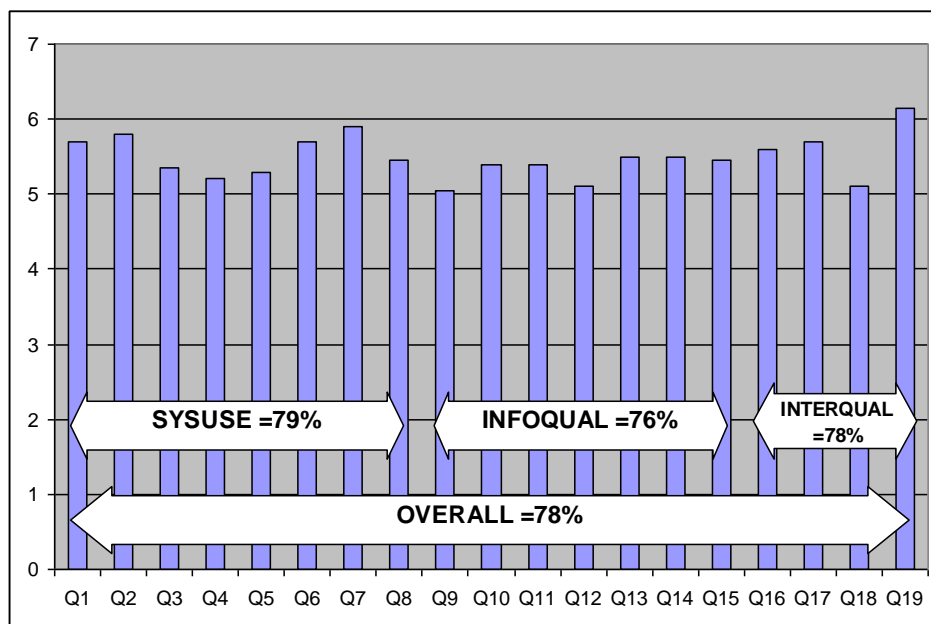


Figure 9.9: Average scores for all the questions in the questioner and the usability sub-scales results.

The satisfaction sub-scales show acceptable scales as Figure 9.9 shows that the system overall sub-scale scored 78% of the scale of satisfaction. For the system use, the system scored 79% which indicates that the system provides an acceptable degree of learnability and ease-of-use. The information quality scored 76% and the interface quality scored 78%. These results provide an indication that the users of the system were satisfied with the use, look and feel of the UML-JMT system. The participants pin pointed some areas in the interface that need to be updated and provided some suggestions to improve the usability of the UML-JMT tool, some of which we will discuss in Section 9.4.

Table 9.3: Results of the satisfaction metrics for the UML-JMT system.

Participant	OVERALL	SYSUSE	INFOQUAL	INTERQUAL
RUH01	5.68	5.63	5.57	6.00
RUH02	5.68	5.88	5.71	5.00
RUH03	5.47	5.25	5.71	5.67
RUH04	5.89	6.00	5.86	5.33
RUH05	5.37	5.38	5.43	5.33
RUH06	5.37	5.38	5.29	5.33
RUH07	5.79	5.38	6.00	6.00
RUH08	5.53	5.75	5.71	4.33
RUH09	4.68	4.13	5.00	5.00
RUH10	5.37	5.75	5.14	4.67
RUH11	4.47	4.75	4.00	4.33
RUH12	5.68	5.75	5.43	6.00
RUH13	5.32	5.50	5.43	4.67
RUH14	6.21	6.50	5.86	6.00
RUH15	5.63	6.00	4.71	6.33
RUH16	5.05	4.88	4.57	6.33
RUH17	5.63	5.88	5.43	5.33
RUH18	5.68	5.75	5.43	6.00
RUH19	5.74	5.88	5.29	6.00
RUH20	5.58	5.63	5.29	5.67
Average	5.49	5.55	5.34	5.47
% of an overall scale of 7	78.46%	79.29%	76.33%	78.10%

The percentage of participants who were able to complete the whole scenario explained in 9.1.2 was 100%. The time of completion and the degree of assistance varied from one to another, however, as discussed earlier, we will be ignoring the time factor as it does not affect the usability of the system, as the nature of the system is not time dependant. As previously discussed, we will use the scores of items 4 and 5 from the questionnaire to analyse the effectiveness of the system. As Figure 9 shows, the participants' average score for item 4 was 5.2 (74%) and for item 5 it was 5.3 (76%).

Although these scores provide an acceptable indicator of effectiveness, the SD of these two items was highest among the other items of the questionnaire, as shown in Figure 9.8. This indicates that the scores provided by the participants ranged over a wider spectrum than the other items. If we return to the previous interviews analysed in 9.2 results we can find the reason for this difference in the scores provided. The dependability of the UML-JMT tool on UML standard modelling notations, which were not commonly deployed by, or known to (at least to some of) the participants in the projects they were involved in, affected the effectiveness of this tool. Another reason for this range in the scores became apparent when deploying the last step of the performance study scenario which includes stress testing the IR system. The knowledge gap between software and performance engineering became visible again as some participants faced difficulties interpreting and analysing the throughput results and graph.

9.4 Conclusion

We have conducted this study to investigate the attitude of software engineers working in the software development industry towards model based non-deterministic performance evaluation methodologies and tools, in the context of their adequacy to be deployed in an engineered style. We were concerned with analysing the reasons that may cause the low appreciation of this performance testing paradigm. This section will summarise the main outcomes of the study. We will also suggest some improvements to the UML-JMT tool that might increase its effectiveness and usability.

9.4.1 Study Outcomes

We noticed from the results of the interview that a large percentage of the participants did not have any prior knowledge of the model based non-deterministic performance testing and its role in non-functional requirements verification. This affected their judgment of its importance in the software development practice. The cause of this lack of knowledge mainly comes down to the basic training provided to the software engineers on an academic level, and only less than 5 % of the participants have come across this performance testing paradigm at the academic training level. We noticed that, as the participant's level of knowledge of this specific paradigm increased, their attitude towards the importance and level of deployment this paradigm can take in the software development process, have increased. This indicates that one of the main reasons for unpopularity of this verification paradigm, besides its complexity, is the absence of knowledge about the paradigm. As the complexity factor was predominantly

solved by the introduction of the model-assistant building methodologies which convert the architectural models to equivalent performance models, the main cause of complexity became the analysis and interpretation of the performance indices gained from solving the generated performance models. This can be solved by providing functionalities in the performance testing tools which describe the performance indices gained from the performance studies in software engineering and design terminology.

The main model transformation methodologies for non-deterministic system performance testing adopted UML modelling notation as the base model to generate the performance model used in the performance test. This was based on the suggestion that UML is the standard modelling notation widely used in the industry for development and documentation. As a part of this study, we wanted to investigate this suggestion by asking the participants about their level of knowledge and usage of the UML standard. The result gained in this study showed that the assumption set by all of the model transformation methodologies is not always true. We noticed that the organisation development policy is the main reason for these participants to deploy UML. This factor partly affects the effectiveness of the model transformation methodology which assumed that UML is a standard notation commonly used in software development. This is also one of the factors leading to model transformation based performance testing methodology being less frequently used in the industry. Although we found that UML notations were not widely used, we noticed that more than 80% of the participants were willing to use standard or non standard modelling notations dedicated for NFR verification if it will assist them in producing reliable tests. Consequently, we can assume that model transformation methodologies still provide valuable assistance methods to simplify the performance model building task.

The system usability study concentrated on the level of satisfaction and effectiveness scales. The satisfaction sub-scales show acceptable scores, as discussed earlier in 9.3.3. Also, the percentage of completion and the sub-scales dedicated to effectiveness showed a high acceptance rate in the context of effectiveness. Although these scores provide an acceptable indication of effectiveness, the SD of these two items was the highest among the other items of the questionnaire, as shown in Figure 9.8. This is due to the dependability of the UML-JMT tool on UML standard modelling notations which were not commonly deployed by the participants. Another factor that may affect the efficiency is the output format of the performance studies. The knowledge gap between

software and performance engineering became visible again as some participants faced difficulties interpreting and analysing the throughput results and graph.

9.4.2 Suggestions

During the study, the participants were asked to provide suggestions for the UML-JMT tool. Here, we list some of the suggestions provided by the participants:

- 1- The UML-JMT tool needs to be part of a larger CASE tool which provides more functionalities including other NFR verification tests.
- 2- The tool needs to provide automatic analysis of the performance data by identifying problematic design areas and providing suggestions for them. This can be done by deploying anti-patterns deduction algorithms.
- 3- The results need to be more in the software engineering context as they are still explained in performance engineering terminology. One suggestion is adding an intelligent report generator capable of reading the results and providing the performance study results and suggestions to amend the design.
- 4- The tool performance data card wizard needs to provide more assistance to the user by providing him/her with more information about the requested data and the source of this data. Also, the wizard needs to provide the performance data for off-the-shelf components commonly used, such as web servers and DBMS.
- 5- The adaptation of hardware related modelling notation instead of a component-based modelling view of the system architecture.

We will explain the possible modifications to the system in the next chapter when we discuss the future work.

9.5 Summary

One of the main contraptions of this thesis is the validation of the resulting methodology and tool from qualitative point of view in the software industry. This validation was undertaken by conducting a qualitative study that involved demonstration the tool to software professionals and investigating their attitude toward it specifically and toward the use of nondeterministic model transformation based performance requirement validation methodology in general. The main objective of this qualitative study is to investigate the effectiveness of the method transformation methodology by studying the level of knowledge that members of the software engineering community support for this specific paradigm of performance requirements validation, and the reasons for the lack of utilisation for this paradigm in the software development industry. Our hypothesis state that, the lack of deployment of this

performance requirements validation paradigm returns mainly to the knowledge gap between software and performance engineering domains. The introduction of the UML-EQN tool was aiming to bridge this knowledge gap by introducing methods for assisting the performance study initiation, starting at gathering the performance data required for the performance study and ending with efficient, easy to use experimentation functionalities available in the JMT tool. As a part of the objectives, we need to investigate if the cause of this knowledge gap returns to the absence of knowledge about the paradigm itself or does it return to problems in the model transformation methodology.

The study involved interviewing a group of software engineers from different sectors of the software development sectors. The study involved demonstrating two performance requirement validating tools based on the model transformation methodology one of them is the UML-EQN. The reason for demonstrating the second performance validation tool is to investigate the user's acceptance and attitude toward the method usually adopted in collecting the performance study data (UML-SPT) compared to the method adopted in the UML-EQN (PDC). The study was aiming to assess the participants' level of acceptance and their attitude toward the non-deterministic performance validation as a design aid in general and the model transformation methodology for deploying this methodology specifically. These satisfaction metrics were measured before and after the introduction of a treatment represented as workshop explaining the validation paradigm deployment using the demonstrated tools.

The main results of this study showed a complacency of the participants' views with the hypothesis set before the study. Our hypothesis about the absence of knowledge about the non-deterministic performance requirement validation was realised as more than half of the participants did not hear about this paradigm before, and none of them came across it at the academic level. The introduction of the method transformation methodology changed the attitude of the participants toward the importance of validation at the design level, and increased their level of confidence on conducting this type of study. This gives an indication that our hypothesis of the knowledge gap causing the lack of utilisation of this paradigm in the industry is true. The only hypothesis that was contradicted by the study's results was the one declared by most of the performance evaluation tools in the availability of the UML models as an artefact that could be utilised in the process of performance evaluation is not always true as majority of the

participants declared that UML is not a used standard during development. Although the participants declared that UML modelling is an overhead in the development process, they declared that it will be useful with the availability of tools similar to the tools demonstrated in the study.

The final chapter of this thesis will summarise the achievements gained in the field of software system performance engineering, and will provide some of the conclusions and suggestions for future work in this field. Section 10.1 will summarise the novelties, improvements and extensions provided by the work discussed, compared to the original work, and the extent to which the work in this thesis meets the requirements of software performance engineering. Section 10.2 will outline and discuss some of the improvements and open problems related to the domain of this work. Finally, Section 10.3 will outline the relevant articles published during the preparation of this thesis.

10.1 Contributions and Achievements

In this thesis, we have considered performance evaluation of software systems. The performance evaluation of software systems is a hugely valuable task, especially in the early stages of a software project. The goal of performance evaluation is to provide performance related design guidance during the system development. Literature reports many methodologies for integrating performance analysis into the software development process. These methodologies work by utilising the software architectural and behavioural models known in the software engineering field, by transforming these models into performance models that can be analysed to attain the expected performance characteristics of the projected system. We discussed in the early chapters that the utilisation of non-deterministic model transformation methodologies faces a challenge caused by its own terminology. This is caused by the knowledge gap between software and performance engineering. The work of this thesis aims to bridge this knowledge gap by introducing a semi-automated transformation methodology which was designed from the beginning to be generic, in order for it to be integrated into any of the leading software engineering development processes. The first work of the author was to determine the key criteria that should be covered in the model transformation methodology so that it can provide the user with the black-box effect which can support the bridging of the two knowledge domains. These criteria were discussed in 4.3.1.

The first attempt to develop a model transformation methodology was the extension of the state marking methodology discussed in Chapter 5. We showed that the automation criterion can be applied to the original methodology; as we explained an algorithm for systematically constructing a Markov chain model from a UML sequence diagram. Although the method we presented in Chapter 5 automates the generation of Markov chain performance model, we can only describe this method as an assisted method, as the modeller is required to identify the appropriate model for the system architecture in hand, and furthermore, know the type of system performance variables to annotate the sequence diagram model with, in order to generate the required performance data. We believe that a fully automated method will only be true if it provides the modeller with assistance in gathering the required performance variables needed for the performance evaluation process, as well as providing the user with the required performance indices.

The need for a methodology that will assist the user in choosing the performance study, capture the required performance variable and simplify the build and analysis of the performance model inspired the author to come up with another methodology, which was the UML-EQN methodology discussed in Chapter 6. This methodology adopts the SPE framework in dividing the architectural model of the system under study into two meta-models, which are called the software and machine models. This will give the designer the benefit of testing different alternatives of structural behavioural configurations. This performance study would help the designer to decide an initial design for the projected system. The UML-EQN methodology takes advantage of the use-case and sequence diagrams to build the software model and deployment diagram for structuring the machine model. The resulting performance model is an EQN performance model. We introduced the performance data card, a data sheet used for supporting the capture of the performance variables used in the build and analysis of the performance model. With the help of an automatic design model to performance model algorithms introduced in the UML-EQN methodology, a software engineer with basic knowledge of performance modelling paradigm can conduct a performance study on a software system design. This was proved in a qualitative study where the methodology and the tool deploying this methodology were tested by software engineers with different levels of background, experience and from different sectors of the software development industry. The study results that we explained in Chapter 9 showed an acceptance for this methodology and the UML-JMT tool which deploys this methodology from these participants.

In Chapter 7, we discussed the design and implementation of the UML-JMT tool. This tool is based on the UML-EQN methodology. The UML-JMT tool formats the output model so that it can be solved and analysed using a non-product form queuing network simulation engine available in the JMT suite[10]. Although the JMT suite provided a variety of performance model building and solving and analysis tools, it lacked the ability to adopt software design models as the starting point for the performance study, a requirement seen in literature as the solution to close the gap between software engineering and performance engineering. UML-JMT comes as a bridge to fulfil this requirement. The UML-JMT tool provides the user with abilities to conduct different types of performance studies that will assist in the system design task. The UML-JMT is designed to be used as an automatic testing tool for the verification of performance non-functional requirements. This functionality is essential in incremental and agile software engineering processes. In software developed using these development processes, continuous verification of the requirements is a fundamental process. This comes down to the fact that these software development paradigms will allow continuous change in the system's requirements. These changes may have effects on the overall performance of the system. The author has suggested the CPASA framework (discussed in 4.2.1) for the assessment of a system performance during the development of these systems, using incremental and agile development paradigms. The UML-JMT was designed to implement the performance evaluation tests specified in this framework. Continuous assessment of software performance requires a comprehensible tool that provides the user with performance characteristics of a design. This tool is designed to fulfil the needs of software engineers with minimal knowledge of performance engineering theory, as it introduces a fully automated model building and analysis approach provided by the UML-JMT tool and the analysis tools available in JMT suite. The UML-EQN methodology and the UML-JMT tool were validated quantitatively by comparing the results gained by the UML-JMT tool and by comparing the results provided by similar performance model transformation tools and other performance evaluation paradigms, as the case study discussed in Chapter 8 showed.

10.2 Open Problems and Future Work

The work presented in this thesis was aiming to bridge the performance engineering process for software systems by introducing model transformation methodologies and methods for deploying these methodologies in different software engineering

paradigms. This section will discuss some of the open areas and future work in the same field of research, in both the theoretical and practical parts of this work.

10.2.1 Model Transformation Methodologies

The results gained from the qualitative study discussed in Chapter 9 reviewed some of the grey areas that the research community have taken for granted, which in return, caused the lack of utilisation of the non-deterministic performance evaluation practice in the real software development world. One of these areas is the assumption, made in all of the method transformation methodologies, that UML is a standard modelling tool used in the development of the majority of software systems. We found that most of the software engineers consider the UML modelling an overhead. A large percentage of the software engineers interviewed agreed that the UML models would be useful if the performance tools provided results with an acceptable degree of accuracy. This gives an indication that the real requirements of the software engineering community are to have freedom in the type of software architecture format provided to the performance evaluation tool. This means that we require extensions of the model transformation methodologies that would transform different UML notations (i.e. activity or state chart diagrams for the scenario), or even take advantage of the re-engineering approaches used to build UML-models from source code, which can be used to generate the performance model.

Another open area in the UML-EQN methodology is in the performance parameters capturing support method. We have introduced an uncomplicated approach represented by the PDC which only introduces the user of the methodology to the name and type of the performance parameter required. This support method requires additional effort in terms of how it can be included in the requirement gathering task, and in providing users of the methodology with methods for acquiring these parameters.

10.2.2 CPASA Framework

Agile software development methodologies are the latest trends in the software development industry. These methodologies focus on increasing the business values of the software system and decreasing the potential risks in the development process. One of the likely risks in any software development is the system not meeting the potential performance expectations. The main factor for such a risk is caused by improperly designed architecture. In Chapter 4, we introduced the CPASA framework, an extension to the PASA method which was designed primarily for the conventional software

development methodology. The CPASA framework has been extended to be deployed on agile developed projects. The primary philosophy of CPASA is continuous change in the initial plans which require continuous assessment of the architecture's performance. We have introduced the various steps of the CPASA method and explained how to employ this method using the UML-JMT tool. The CPASA is a generic method that was suggested for agile development methodologies. Future work for this framework includes customising it for specific agile development methodologies (i.e. XP, scrum ... etc.) that would include performance testing as one of the development practices for these agile development methodologies, and furthermore, building specialised CASE tools for continuous testing based development which will include the UML-JMT tool as one of the tools used in the deployment of these development methodologies.

10.2.3 Improving the UML-JMT Tool

During the qualitative study discussed in Chapter 9, the participants were asked to suggest services that they expect from a performance evaluation tool. Some of the participants' suggestions provided ideas for improvements and extensions that can increase the acceptance and assistance required from the methodology and the tool. One of the main suggestions was to represent the tool in a software engineering context. This is essential as the expected users faced some problems trying to cope with the performance engineering terminology. We partly solved this problem by re-designing the PDC wizard to eliminate any pure performance engineering terminology. The results and benefit of these results still faces a considerable challenge. The tool needs to provide automatic analysis of the performance data by identifying problematic design areas and providing suggestions for these problematic designs. This can be done by deploying anti-patterns deduction algorithms that can be used to identify anti patterns, which could cause performance problems. The results need to be more in the software engineering context as they are still explained in performance engineering terminology. One suggestion is adding an intelligent report generator capable of reading the results and providing the performance study results and suggestions, to amend the design. Another suggestion is to provide readymade performance tests which are known in the field of software engineering (i.e. stress test, bottleneck ... etc.), which can be selected by the user at the PDC wizard. Also, a full report is generated at the end of the test in software engineering terminology. This can be done as the JMT suite stores the test type and the performance results in the same XML file that contains the performance model.

Currently, the UML-JMT tool requires a full manual written in the context of software engineering knowledge domain. This manual will include the necessary background knowledge, which is essential for typical software engineer in order for his/her to perform a full performance elevation study using this tool. Moreover, the tool needs enhancements on the interface to include hints, and help files that would facilitate the use of the tool, and that will assist the user in finding the source of the entries required to carry out the performance study.

10.3 Relevant Publications

The work described in this thesis has appeared in some publications. These are listed here:

- The state marking methodology was published in various versions in [109] and [131]. The latest version, which was discussed in Chapter 5, was published in [33].
- The criteria used to evaluate the model transformation methodologies discussed in 4.6 were published in [33].
- The UML-EQN methodology discussed in Chapter 6 was published in [9]
- The realisation of the UML-EQN methodology represented by the UML-JMT tool was published in [11]. This paper also included the quantitative validation discussed in Chapter 8.
- The deployment of the performance engineering in agile development context represented by the CPASA approach, discussed in Chapter 4, was published in [12].

USDX Parser Documentation

The USDX parser (Use-case, Sequence and Deployment diagrams XMI) parser is a Java library developed specially for the UML-JMT tool. It provides classes and operations that will help the analysis of UML models represented in XMI document. It is built on top of the javax DOM XML parser. The Class diagram of the USDX parser package is shown in Figure 7.1. This Appendix contains the java documentation for this parser.

class UMLModel

This class represents the main container for the UML model extracted from the XMI file. This class will invoke the extraction methods for all the UML components searched in the XMI File. In addition, it will store the extracted components in containers named with the same name as the UML notation they represent.

Functions

UMLModel (Document)	Constructor, expect the document object of the XML (XMI) file, and it will invokes the different extraction functions for all the UML notations being extracted.
void FindActors (Document)	Traverse the XMI file and extract the Use-Case entries.
Void FindUseCase (Document)	Traverse the XMI file and extract the UseCase entries.
void getScenarios (Document)	traverse the XMI file and extract the Scenarios entries
public DeploymentDiagram getDeploymentDiagram ()	Returns deployment diagram representation
public SequenceDiagram getSequenceDiagram (String Name)	Returns the sequence diagram named "Name"

class SequenceDiagram

This Class represents the Sequence Diagram of the System under study. It will contain a set of Components and a Set of Messages or connections.

the operations in this class include a function that will parse the XMI file and collect all information regarding the Sequence diagram and store it in the Components list

Functions

SequenceDiagram(String name, Document doc)	Constructor, expect the document object of the XML (XMI) file, and the name of scenario this sequence diagram represents. The constructor will invoke the sequence diagram extraction function
private void ExtractSD(Document doc)	Traverse the XMI file and extract the sequence diagram represented by the scenario
String getName()	Returns the scenarios name.
public String getId()	Returns the XMI ID of the scenario,
public ArrayList<Message> getMessages()	Returns the sequence diagram set of messages in an array list of messages

class DeploymentDiagram

This Class represents the Deployment Diagram of the System under study. It will contain a set of nodes. the operations in this class includes function that will parse the XMI file and collect all information regarding the deployment diagram and store it in the nodes list.

Functions

DeploymentDiagram(Document doc)	Constructor, expect the document object of the XML (XMI) file, a. The constructor will invoke the deployment diagram extraction function.
private void ExtractDD(Document doc)	Traverse the XMI file and extract the deployment diagram.
public boolean Rinthesamenode(Component a, Component b)	Returns true if the two components sent are in the same node.
public ArrayList<String> getComponentsNames()	Returns the list of components

class DDNode

The node Class will represent all the nodes representing

the Deployment Diagram Representing the Hardware of the system each of the nodes will have its name, set of components and the nodes connected to it. when created the node will be given a name (the name will be extracted from the XMI file)

the class provide a set of operations for:

- returning the name of the node
- adding a node to be connected
- adding a component to be the set of components
- Checking if a given node is connected to a node
- checking if a given component exist in this node
- and overwriting the equal function

Functions

public DDNode(String name)	Constructor, takes the node's name and creates the connection and components lists
public boolean isComponent(Component C)	Returns true if the component sent is a member of the components list of this node
public void AddComponent(Component C)	Adds a new component to this node
public boolean IsConnected(DDNode N)	Return true if this node is set to be connected to the node in the parameter list
public void setConnection(DDNode N)	Sets a connection between this node and the node N in the parameter list
public String getName()	Returns the nodes name
public boolean equals(DDNode N)	Overwrite the equals function by defining the equality between two nodes objects
public String getId()	Returns the XMI id of the node

class Component

The Component Class will represent all the components representing the system under study each of the components will have its name and set of components connected to it. when created the component will be given a name (the name will be extracted from the XMI file)the class provide a set of operations for:

- returning the name of the component
- adding a component to be connected
- Checking if a given component is connected to a component
- and overwriting the equal function

Functions

boolean equals (Component obj)
--

Overwrite the equals function by defining the equality between two components objects.

String **getId()**

this function will returns the XMI ID of the component

String **getName()**

this function will return the name of the component

Returns:

the name of the component

boolean **IsConnected**(Component C)

Parameters:

C - the Component to be Searched in the Connection list

Returns:

true if the Components are connected

void **Print()**

Void **setConnection**(Component C)

this Function will add a new Component to the list of connected components

Parameters:

C - is the component to be added to the list

Qualitative study Questions

The experiment used to validate the qualitative aspects of the methodology and the tool was composed of four phases. In two of these phases, the participants were involved in a structured interview. A structured interview is conducted with a moderately open framework which allows for focused, conversational, two-way communication. The questions of the interviews and the rationale for each question are shown in Tables B1 and B2.

Afterwards, the participants were given the opportunity to use the UML-JMT tool to execute a scenario example explained in the workshop. After a participant executes this scenario, he/she will be asked to offer suggestions to improve the tool and evaluate the usability of the tool using the standard IBM computer system usability questionnaire (CSUQ)[125]. This questionnaire is shown in Table B3.

Table B1: Pre-orientation questions asked for the participants of the qualitative study on the structured interview.

Question	Rationale
1 Have you been involved in software development? (1 for very few times, 5 for extensively)	To know the frequency the participant is involved in software development.
2 In which part of the process are you usually involved? (1 – analysis; 2 – design; 3 - development; 4 – test; 5 – all)	To clarify which participants are more involved in the analysis and design phases, as they are the more likely to come across performance engineering.
3 Describe the biggest project you were involved in (1 represents a project with less than 3 components, with a budget of < 10K\$, man power of <3 personal and scheduled < 3	To define the scale of experience the subject has by the finding out the size of projects he/she was involved in.

<p>months. And 5 represents a project with > 15 components with a budget of >10M\$, manpower of >30 personal and scheduled > 24 months.)</p>	
<p>4 Describe your knowledge/usage of UML (1 - no knowledge; 2 - learned it at university but never used it; 3 - use it occasionally; 4 - use it regularly; 5 – use it in all the projects I have been involved in)</p>	<p>Test the subject knowledge and experience with UML as it represents a main part of the methodology.</p>
<p>5 Describe your knowledge of software performance engineering study? (1 – I’ve only just heard about it; 2 – I’ve heard about it but could not use it; 3 – I’ve heard about it but wouldn’t use it; 4 – I’ve used it several times; 5 - I use it regularly)</p>	<p>Test the subject’s previous knowledge of performance engineering.</p>
<p>6 If you have knowledge and experience in performance engineering, what is the source of your knowledge?</p>	<p>Determine how many subjects know and have used performance engineering study in the market.</p>
<p>7 Based on your knowledge, how important do you consider software performance engineering study to be? (1 - not important; 2 - it can be included in the testing phase; 3 - good practice for some projects; 4 - important for some projects; 5 - essential for all projects).”</p>	<p>Test the subject’s opinion on the importance of performance engineering.</p>
<p>8 Has a performance study been conducted in any of the previous projects that you have participated in? (1 – none; 2 - real system test; 3 - spreadsheet; 4 - simulation; 5 - benchmarking)</p>	<p>Test how common it is for performance engineering studies to be undertaken in projects.</p>

Table B2: Post orientation questions asked for the participants of the qualitative study on the structured interview.

Question	Rationale
Describe your knowledge/experience software performance engineering study. (1 - I still don't know this paradigm; 2 - I know of it but couldn't use it; 3 - I think I have the basic knowledge to conduct a study; 4 - I think I have the necessary knowledge to conduct any performance test; 5 - I knew it before).	Test the subject's current knowledge of performance engineering.
How good is your current knowledge of software performance study with the provided tools? (1 - not confident; 2 - need more background knowledge; 3 - I can conduct simple performance tests with assistance; 4 - I can conduct simple performance tests without assistance; 5 - I can conduct any performance test).	Determine the level of the subject's knowledge of performance engineering.
Based on your knowledge, how important do you consider software performance engineering study to be? (1 - not important; 2 - it can be included in testing phase; 3 - good practice for some projects; 4 - important for some projects; 5 - essential for all projects)."	Test the subjects' opinion on the importance of performance engineering.
In any of the previous projects you have participated in, has a performance study, like the one described here, been conducted? (1 none 5 all of them)	Test how often the performance engineering methodologies are undertaken in projects.
Based on the knowledge you received earlier, will you be using or recommending software performance studies in your future projects?	Test the subject's confidence in using software performance study.

Based on the knowledge you received earlier, will you be using or recommending PRIMA-UML methodology in your future projects?	Test the subject confidence of using PRIMA-UML.
Based on the knowledge you received earlier, will you be using or recommending UML_EQN methodology in your future projects?	Test the subject confidence of using UML-EQN.
Are you willing to learn new software modelling paradigms to use them in performance evaluation or other NFR validation tasks? (1 - no; 2 - yes, if it will provide accurate results and other NFR verification tests; 3 - yes, if it provides more readable validation tests; 4 - yes, if it provides more accurate performance test indices; 5 - yes, if it is part of a large CASE tool).	To evaluate the willingness of the subjects to learn new modelling notations.
What are the best features that you found in the UML-JMT tool?	To find the subject's view on the features that he/she will recommend the UML-JMT for.
What would you suggest to improve the UML-JMT tool?	To locate future developments in the UML-JMT tool.
What are the best features that you found in the JMT suite?	To find the subject's view on the features that he will recommend the JMT for.

Table B3: IBM Computer System Usability Questioner (CSUQ)

Based on your use of the UML-JMT system		strongly disagree			strongly agree			
		1	2	3	4	5	6	7
Q1	Overall, I am satisfied with how easy it is to use this system.							
Q2	It was simple to use this system.							
Q3	I can effectively complete my work using this system.							
Q4	I am able to complete the suggested work quickly using this system.							
Q5	I am able to efficiently complete the suggested work using this system.							
Q6	I feel comfortable using this system.							
Q7	It was easy to learn to use this system.							
Q8	I believe I became productive quickly using this system.							
Q9	The system gives error messages that clearly tell me how to fix problems.							
Q10	Whenever I make a mistake using the system, I recover easily and quickly.							
Q11	The information (such as on-screen messages, and other documentation) provided with this system is clear.							
Q12	It was easy to find the information I needed.							
Q13	The information provided for the system is easy to understand.							
Q14	The information is effective in helping me complete the tasks and scenarios.							
Q15	The organisation of information on the system screens is clear.							
Q16	The interface of this system is pleasant.							
Q17	I like using the interface of this system.							
Q18	This system has all the functions and capabilities I would expect it to have.							
Q19	Overall, I am satisfied with this system.							

Qualitative Study Results

This appendix contains some of the relevant results from the qualitative study in Chapter 9. Table C1 has the information of the participants involved in the qualitative study. It contains the reference number, name, place and nature of work, and the experience measured in number of years. Table C2 contains the results of the *CSUQ* questionnaire with the standard deviation, mean and median calculated for the results.

Table C1: Participants in the Qualitative Study

Number	Name	Place of Work	Position	Experience	Nature of work
RUH01	Nader Almarzouki	SAMA BTD	System Analyst	2	Application support and system analysis
RUH02	Zyad AlBisa	SAMA BTD	System Analyst	2.5	Development and support of SPAN II and SAREI systems
RUH03	Simon Ainsworth	SAMA BTD	Freelance Consultant	27	Support, consultancy and enhancement for SARIE system
RUH04	Fisal ALHarbi	Chip CS	Project Manager	10	Project manager in different scale systems
RUH05	Abdulraman Alkhanifer	KSU	Vice Director of Portal and E-Services	8	Project management
RUH06	Hussain ALHaddad	KSU	Portal and E-Services Dep.	9	Team leader of design and development
RUH07	Abdulaziz ALOraiji	KSU	Director of Portal and E-Services	10	Software design and analysis
RUH08	Abdullah AlSaleh	KSU	Portal and E-Services Dep.	1.8	System analyst in KSU Portal
RUH09	Omar S. ALAbdullatif	AITawiniya	Project Manager	27	Project manager
RUH10	Hussain ALMutere	MODA-CERT	Director of CERT	5	Project design and management
RUH11	Fawaz Abdulrahaman	Ministry of Water and Electricity	System Analyst	5	Project design and development
RUH12	Abdullah ALMubarak	IOB	IT Manager	15	Finding solutions to support the business

Number	Name	Place of Work	Position	Experience	Nature of work
					activities
RUH13	Ahmad ALHaddab	SAMA BTD	Head of Application Development and Support	7	Design and development of the business applications
RUH14	Abdulaziz ALNadari	SAMA BTD	System Analyst	7	System design and development
RUH15	Mohammed AlRowaijeh	Ministry of Finance	Assistant Project Manager	3.5	System design and development
RUH16	Ali ALEssa	MOF	Project Coordinator	2	System design and development
RUH17	Abdulaziz AlDahmash	SAMA BTD	System Analyst	7	Application support and system analysis
RUH18	Khalid Alangari	Mobily	Manager of Mobily Programs	10	Managing and tracing corporate systems
RUH19	Bader Mohammed	MOF	Project Coordinator	5	System design and development
RUH20	Mohammed Massoud	Chip CS	Project Manager	15	System design and development
RUH21	Ahmad AlSharqi	AlFisaliah ITS	SAB Consultant	8	Design, customisation and support of SAB systems

Table C2: Results of the *CSUQ* questionnaire gained from the qualitative study for the *UML-JMT* tool usability

Participant	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19
RUH01	6	6	5	6	6	5	5	6	5	6	6	5	6	6	5	6	6	6	6
RUH02	6	6	6	6	5	7	6	5	6	7	6	6	5	6	4	5	5	5	6
RUH03	6	6	4	6	4	6	6	4	5	6	6	4	6	6	7	6	7	4	5
RUH04	6	6	6	6	7	5	6	6	5	6	6	6	7	6	5	5	6	5	7
RUH05	6	6	5	4	5	6	7	4	5	6	5	5	5	5	7	6	6	4	5
RUH06	5	5	5	5	6	6	6	5	6	5	5	5	5	5	6	6	6	4	6
RUH07	6	6	4	4	4	6	6	7	6	7	6	4	6	6	7	6	6	6	7
RUH08	5	5	6	6	6	7	5	6	5	6	5	6	7	7	4	4	5	4	6
RUH09	4	4	5	2	3	4	5	6	6	3	5	5	4	6	6	5	5	5	6
RUH10	6	6	6	5	6	5	6	6	6	5	6	5	5	5	4	5	4	5	6
RUH11	6	6	4	4	4	5	5	4	3	5	5	3	4	3	5	5	5	3	6
RUH12	6	6	5	6	6	6	6	5	6	5	6	5	5	6	5	6	6	6	6
RUH13	6	7	5	4	4	6	7	5	5	6	6	5	6	5	5	5	5	4	5
RUH14	5	7	6	7	6	7	7	7	6	5	6	6	6	6	6	6	6	6	7
RUH15	6	6	6	6	6	6	6	6	4	4	5	4	5	5	6	6	6	7	7
RUH16	5	4	6	4	4	6	4	6	4	4	4	6	4	6	4	7	6	6	6
RUH17	6	6	6	7	6	5	6	5	4	6	5	5	6	5	7	5	6	5	6
RUH18	6	6	5	6	6	5	7	5	5	6	6	5	6	4	6	6	6	6	6

Appendix C

RUH19	6	7	6	5	6	6	6	5	4	5	5	6	6	6	5	6	6	6	7
RUH20	6	5	6	5	6	5	6	6	5	5	4	6	6	6	5	6	6	5	7
RUH21	6	6	5	6	6	6	6	6	6	5	5	5	5	6	5	6	5	6	6
Statistical Results	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19
Standard Deviation	0.57	0.83	0.75	1.24	1.08	0.80	0.79	0.89	0.89	0.99	0.68	0.85	0.89	0.89	1.05	0.68	0.66	1.02	0.67
Median	6	6	5.5	5.5	6	6	6	5.5	5	5.5	5.5	5	6	6	5	6	6	5	6
Mean	5.7	5.8	5.35	5.2	5.3	5.7	5.9	5.45	5.05	5.4	5.4	5.1	5.5	5.5	5.45	5.6	5.7	5.1	6.15
% of 7 point Scale	81	83	76	74	76	81	84	78	72	77	77	73	79	79	78	80	81	73	88

Model Transformation Methodologies Review

In this appendix we provide a survey of some of the work done in the field of generating performance models from design models. We will discuss the methodologies in the context of the criteria we discussed in section 4.3.1. For each of the methodologies discussed, we will provide the input UML models, the output performance model, and will summarise the performance model generation process. Then we will classify these methodologies' compliance with the criteria defined in 4.3.1 as *High*, *Mid* (medium) or *Low*. High compliance reflects that the methodologies comply with the entire factor defining a criterion or have extra features that cover the missing factors. Medium compliance refers to the methodology complying with some of the factors and low refers to the methodology not complying with any of the factors which define the criterion.

For simplicity, the survey is formatted in a table form. Table D1 contains a survey of some of the methodologies for performance evaluation. We have constricted it to methodologies that adopt the SPE method of separating structural and behavioural aspects. This is a return to the use of this method in the main performance model building methodology, as discussed in Chapter 6 of this thesis. For the same reason, we also concentrated on methodologies generating QN models. We have included in the survey selected methodologies that produce other types of performance models (i.e. Petri Nets, Process algebra and simulation). There are many papers which review other methodologies that the reader can refer to(e.g.[132; 133]).

Table D1: A survey of Software model to performance model transformation methodologies.

Methodology	Input	Output	Methodology Summary	Criteria			
				<i>Time Efficiency</i>	<i>Generality</i>	<i>Transparency</i>	<i>Automation</i>
SPE [6; 92] Williams and Smith	SD,DD, EG	EQN	The original SPE methodology was described in sec.4.1.2, in [92], the SPE methodology was extended to utilise UML models in creating EG used in the creation of the end EQN model.	(Mid) - The SPE uses a non-standard meta-model represented by the EG. - EQN are easy to solve and tools to simulate and analyse are available	(High) - The EQN performance model produced by SPE is capable of representing any class of system architecture[62] (See 3.5).	(High) - QNs preserve the structure of the components representing the system.	(High) - SPE was automated by a tool called SPEED[91].
PRIMA-UML Cortellessa, Mirandola [93]	UC,SD, DD	EQN	This methodology was an extension of the original SPE methodology. It provided algorithms for building the system and machine models specified in the SPE methodology form UC, SD and DD. This methodology uses an intermediate execution graph generated from the SD and feeds the results of it along	(Mid) - Like SPE, this methodology uses a non-standard meta-model represented by the EG.	(High) - The EQN performance model produced is capable of representing any class of system architecture.	(High) - QNs preserve the structure of the components representing the system.[82]	(High) - A tool based on this methodology was described in [134]

Appendix D

Methodology	Input	Output	Methodology Summary	Criteria			
				Time Efficiency	Generality	Transparency	Automation
			with information from both UC and DD to generate an EQN.	- EQN are easy to solve and tools to simulate and analyse are available			
UML-LQN[103] Cortellessa, et al	CD, SD	LQN	The methodology extends the SPE methodology, as it uses an EG to build a LQN, it differs from the above methodology in its utilisation of CD and SD in the process of building the EG. The methodology defines a complete approach for collecting the necessary performance characteristics of the system under study.	(Mid) - This methodology uses a non-standard meta-model represented by the EG. - LQN are easy to solve and tools to simulate and analyse are available.	(High) - The LQN performance model produced by SPE is capable of representing any class of system architecture.	(High) - QNs preserve the structure of the components representing the system.	(Low) - Although the methodology provided a systematic way for producing a QN model, there was no tool to automate it.
Architectural patterns Petriu and Wang	CoD, SD	LQN	The methodology considers a significant set of architectural patterns; these patterns are specified by CoD and SD. The methodology suggests corresponding performance	(Mid) - This methodology uses a non-standard meta-	(High) - The LQN performance model produced	(High) - QNs preserve the structure of the components	(High) - This methodology was automated by two

Appendix D

Methodology	Input	Output	Methodology Summary	Criteria			
				Time Efficiency	Generality	Transparency	Automation
[104]			model based on LQN for each of these patterns. The methodology suggests that more complex SA models can be constructed by combining set of patterns that compose the system. The methodology depends on an SPE approach in which the LQN is built using a meta execution model.	model represented by the EG. - LQN are easy to solve and tools to simulate and analyse are available	is capable of representing any class of system architecture.	representing the system.	tools [104; 135] these two tools provide that same functionality but differ in the method of representing the performance characterisation of the system.
UML-QN [105] Pooley, King	DD	PFQN	The methodology suggested that QN can model UML DD, mapping the resources in the deployment diagrams to service centres and the communication links to the queues themselves. The methodology introduced a method to add performance data in UML diagrams in the form of performance tags (time labels).	(High) - No meta-model - PFQN have efficient algorithms and tools to solve and analyse.	(Low) - The methodology output model is PFQN which have limits in the solving algorithms for a range of system architectures.	(High) - QNs preserve the structure of the components representing the system.	(Low) - Although the methodology provided a systematic way for producing a QN model, there was no tool to automate it.
UML-QNE (UML Queuing	UC, AD ,DD	PFQN	Te methodology takes advantage of the hardware/software model	(High)	(Low)	(High)	(High)

Appendix D

Methodology	Input	Output	Methodology Summary	Criteria			
				Time Efficiency	Generality	Transparency	Automation
Network Evaluator) Balsamo <i>et.al.</i> [102]	UML-SPT[1]		separation found in SPE. The hardware baseline is extracted from the DD, where the AD and UC are used to define the behaviour of the routing between the service centres defined by the components of the DD. The methodology uses the UML-SPT that is performance characterisation is annotated in the input UML models.	- Direct mapping no meta-model - PFQN has a range of efficient algorithms and tools to solve and analyse.	- The methodology output model is PFQN which have limits in the solving algorithms for a range of system architectures. - Writer suggested the use of EQN in the case of fork/join in ADs.	- QNs preserve the structure of the components representing the system.	- A tool is implementing this methodology (UML Queuing Network Evaluator) [102] The tool depended on translating these UML diagrams represented as an XML.
ArgoSPE. Martinez and Merseguer [107]	SC, AD, ID UML-SPT[1]	GSPN	The methodology involves translating UML-SPT model to a GSPN model. The methodology take advantage of behaviour UML models represented as SC, AD and ID to directly map them to an equivalent GSPN model, performance characterisation is annotated in the input UML models.	(High) - Direct mapping no meta-model - The ArgoSPE provide a query based system where the user will query the performance	(Mid) - The methodology output model is GSPN which is limited for behavioural modelling[82] See 3.5.	(Low) - GPSN does not preserve the architecture of the modelled system.[82]	(High) - ArgoSPE as a plug-in in ArgoUML[43]

Appendix D

Methodology	Input	Output	Methodology Summary	Criteria			
				Time Efficiency	Generality	Transparency	Automation
				aspects of the UML model directly proving a black box effect.			
UML to GSPN Bernardi et al. [108]	SC,SD	GSPN	The methodology uses the state diagram to provide information about the <i>single objects of a system</i> . The sequence diagrams provide information about inter-object communication. The information gathered will construct the wanted model.	(Mid) - Direct mapping no meta-model - GSPN models are complicated in the context of representation.	(Mid) - The methodology output model is GSPN which is limited for behavioural modelling.	(Low) - GSPN does not preserve the architecture of the modelled system.	(Low) - Although the methodology provided a systematic way for producing a GSPN model, there was no tool to automate it.
State Marking methodology [97]	CoD, SC	GSPN	Method for deriving performance models based on (GSPN) from UML collaboration-state chart diagrams. The suggested methodology takes advantage of the idea of marking, used in GSPN modelling as the state of the system at each step in the model's execution. The overall marking of the system will form the	(Mid) - Direct mapping no meta-model - GSPN models are complicated in the context of representation.	(Mid) - The methodology output model is GSPN which is limited for behavioural modelling.	(Low) - GSPN does not preserve the architecture of the modelled system.	(Low) - Although the methodology provided a systematic way for producing a QN model, there was no tool to

Appendix D

Methodology	Input	Output	Methodology Summary	Criteria			
				Time Efficiency	Generality	Transparency	Automation
			required performance model.				automate it.
PEPA to UML Canevet et al. [100]	SC, CoD	PEPA	The methodology associates exponentially distributed random variables to actions, The methodology work by extracting information related to PEPA from SC and CoD to capture information related to state machines and their components; and how these components collaborate. This information will be used by a Cooperation to generate the PEPA system equation.	(Mid) - The clarity of PEPA provide a challenge to the user See 3.5.	(High) - The PEPA performance model produced is capable of representing any class of system architecture.	(Low) - PEPA does not preserve the architecture of the modelled system.	(High) - The methodology is automated as [100] described the integration of ArgoUML with PEPA workbench
Simulation SimML Arif and Speirs [106]	SD, CD	Simulation	The methodology work by Transforming the UML diagrams into a simulation model described as an XML document. The XML notation used to describe the simulation model has been called SimML (Simulation Modelling Language). This model is then translated into a simulation program, which can be executed and provides	(Low) - The analysis of simulation model especially for complex and large systems can be a complicated process.	(High) - The use of simulation allow the method to include all system architectures.	(High) - The use of simulation will allow reverse engineering.	(High) - The tool implementing this methodology known as SimML[106].

Appendix D

Methodology	Input	Output	Methodology Summary	Criteria			
				<i>Time Efficiency</i>	<i>Generality</i>	<i>Transparency</i>	<i>Automation</i>
			performance results.				



References

- [1] OMG, UML Profile for Schedulability, Performance, and Time Specification, Object Management Group, 2005.
- [2] G.Serazzi, Java Modelling Tools - System Manual, Dipartimento di Elettronica e Informazione Milano - Italy, v.0.1, Jan. 2009.
- [3] R. Pooley, Using UML to Derive Stochastic Process Algebra Models. in: N. Davies, and J. Bradley, (Eds.), Proceedings of the Fifteenth UK Performance Engineering Workshop, University of Bristol, Bristol, UK, 1999, pp. 23-33.
- [4] I. Sommerville, and P. Sawyer, Requirements engineering: a good practice guide, John Wiley & Sons, Inc. New York, NY, USA, 1997.
- [5] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of Software Engineering, Prentice Hall Englewood Cliffs, NJ, 1991.
- [6] C.U. Smith, Performance Engineering of Software Systems, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1990.
- [7] R. Pooley, Software Engineering and Performance: a Roadmap, Proceedings of the Conference on The Future of Software Engineering International Conference on Software Engineering, , ACM Press New York, NY, USA, Limerick, Ireland 2000, pp. 189-199.
- [8] C.U. Smith, and M. Woodside, Performance Validation at Early Stages of Software Development., Performance 99, Istanbul, Turkey, 1999.
- [9] A.A. Abdullatif, and R. Pooley, From UML to EQN: Studying System Performance from an Early Stage of Systems Life Cycle. in: K. Djemame, (Ed.), Proceedings of the 25th UK Performance Engineering Workshop, University of Leeds, Leeds, 2009, pp. 111-122.
- [10] M. Bertoli, G. Casale, and G. Serazzi, The JMT Simulator for Performance Evaluation of Non-Product-Form Queueing Networks, Proceedings of the 40th Annual Simulation Symposium IEEE Computer Society Washington, DC, USA Norfolk, Virginia, USA, 2007, pp. 3-10.
- [11] A.A.L. Abdullatif, and R.J. Pooley, UML-JMT: A Tool for Evaluating Performance Requirements, Proceedings of the 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, IEEE, Oxford, England 2010, pp. 215-225.
- [12] R.J. Pooley, and A.A.L. Abdullatif, CPASA: Continuous Performance Assessment of Software Architecture, Proceedings of the 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, IEEE, Oxford, England 2010, pp. 79-87.
- [13] L.G. Williams, and C.U. Smith, PASA: a Method for the Performance Assessment of Software Architectures, Proceedings of the 3rd international workshop on Software and performance ACM New York, NY, USA, Rome, Italy 2002, pp. 179-189.

- [14] P. King, and R. Pooley, Derivation of Petri Net Performance Models from UML Specifications of Communications Software, Computer Performance Evaluation. Modelling Techniques and Tools: 11th International Conference, TOOLS 2000,, Springer, Schaumburg, IL, USA, , 2000, pp. 262-276.
- [15] M.A. Jackson, Principles of Program Design. Studies In Data Processing **12** (1975) 300-312.
- [16] OMGSpecification, UML 2.0 Specification, Object Management Group, 2005.
- [17] A. Abran, J.W. Moore, P. Bourque, and R. Dupuis, Guide to the software engineering body of knowledge: trial version, IEEE, 2001.
- [18] W.W. Royce, Managing the development of large software systems: concepts and techniques, Proceedings of the 9th international conference on Software Engineering, IEEE Computer Society Press, Monterey, California, United States, 1987.
- [19] B.W. Boehm, Software engineering, Classics in software engineering, Yourdon Press, 1979, pp. 323-361.
- [20] L.B.S. Raccoon, The Chaos Model and the Chaos Cycle. ACM SIGSOFT Software Engineering Notes **20** (1995) 55-66.
- [21] IBAG, V-Model Documentation, 2006.
- [22] I. Sommerville, Software Engineering. 7th Edition, Pearson/Addison-Wesley, 2004.
- [23] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.
- [24] P. Kruchten, The rational unified process: an introduction, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000.
- [25] J. Stapleton, DSDM, Dynamic Systems Development Method: the Method in Practice, Addison-Wesley Professional, 1997.
- [26] K. Schwaber, and M. Beedle, Agile Software Development With Scrum, Prentice Hall Upper Saddle River, NJ, 2001.
- [27] K. Beck, and C. Andres, Extreme Programming Explained: Embrace Change, Addison-Wesley Professional, 2004.
- [28] S.W. Ambler, and R. Jeffries, Agile Modeling: Effective Practices for Extreme Programming and the Unified Process, Wiley New York, 2002.
- [29] Sujoy Bose, M. Kurhekar, and J. Ghoshal, Agile Methodology in Requirements Engineering, SETLabs, 2008.
- [30] B. Boehm, and R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed Reading, MA: Addison-Wesley, 2004.
- [31] H.F. Hofmann, and F. Lehner, Requirements engineering as a success factor in software projects, IEEE Computer Society, 2001, pp. 58-66.
- [32] A. Podelko, Performance Requirements, 2006.
- [33] A.A. Abdullatif, and R. Pooley, A Computer Assisted State Marking Method For Extracting Performance Models From Design Models. International Journal of Simulation Systems, Science & Technology **8** (2008) 36-46.
- [34] E. Codd, A relational model for large shared data banks. Communications of the ACM **6** (1970).
- [35] P.S. Chen, entity relation models: toward a unified view of data. ACM transactions on Database systems(TODS) **1** (1976) 9-36.
- [36] T. DeMarco, J.D.W. Prize, and S. Prize, Structured Analysis and system specification, Yourdon Press, New York, 1979.
- [37] E. Yourdon, and L.L. Constantine, Structured design: fundamentals of a discipline of computer program and systems design, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1979.

- [38] C.H. Kung, and A. S Ivberg, Activity modeling and behavior modeling, North-Holland Publishing Co. Amsterdam, The Netherlands, The Netherlands, 1986, pp. 145-171.
- [39] M.L. Brodie, and E. Silva, Active and passive component modelling: ACM/PCM, North-Holland Publishing Co. Amsterdam, The Netherlands, The Netherlands, 1986, pp. 57-107.
- [40] G. Booch, J. Rumbaugh, and I. Jacobson, Unified Modeling Language User Guide, The (Addison-Wesley Object Technology Series), Addison-Wesley Professional, 2005.
- [41] I. Vessey, and A.P. Sravanapudi, CASE tools as collaborative support technologies, ACM New York, NY, USA, 1995.
- [42] K.E. Kendall, and J.E. Kendall, Systems analysis and design, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2002.
- [43] J. Robbins, Argo UML, University of California, open source UML modeling tool and includes support for all standard UML 1.4 diagrams.,v.0.81, 2001.
- [44] O.M. Group., XML Meta Data Interchange (XMI) – Proposal to the OMG OA&DTF RFP 3: Stream-based Model Interchange Format (SMIF), Framingham., October 1998.
- [45] OMGSpecification, XMI MOF 2.0/XMI Mapping Specification, v2. 1.1, Object Management Group, 2007.
- [46] A.S. Tools, ARTiSAN Studio 6.1 Features SysML Requirements Modelling, XMI 2.1, Embedded systems directory and blog, ARTiSAN Software Tools, 2006.
- [47] D. Megginson, Sax 2.0: The Simple API for XML, 2000.
- [48] B. McLaughlin, and J. Edelson, Java & XML, O'Reilly Media, Inc., 2006.
- [49] W.S. Means, and M.A. Bodie, The Book of SAX: the Simple API for XML, No Starch Press, 2002.
- [50] J. Banks, J.S. Carson, B.L. Nelson, and D.M. Nicol, Discrete-event System Simulation, Prentice Hall Budapest, Hungary, 2001.
- [51] O.J. Dahl, and K. Nygaard, SIMULA: an ALGOL-based Simulation Language. Communications of the ACM **9** (1966) 671-678.
- [52] P.A. Fishwick, SimPack: Getting Started With Simulation Programming in C and C++, Proceedings of the 1992 Winter Simulation Conference, ACM New York, NY, USA, 1992, pp. 154-162.
- [53] K. Salah, On the Accuracy of Two Analytical Models for Evaluating the Performance of Gigabit Ethernet Hosts. Information Sciences **176** (2006) 3735-3756.
- [54] B.R. Haverkort, Markovian Models for Performance and Dependability Evaluation. Lectures on Formal Methods and Performance Analysis, Lecture Notes in Computer Science **2090** (2002) 38-83.
- [55] J. Hillston, Modelling and Simulation, University of Edinburgh, 2005.
- [56] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press New York, NY, USA, 1996.
- [57] C.A. Petri, Kommunikation mit automaten. Schriften des iim nr. 2, Technical Report RADC-TR-65-377, Institut fur Instrumentelle Mathematic, Griffiths Air Base, New York 1962.
- [58] M.K. Molloy, Performance Analysis Using Stochastic Petri Nets. Computers, IEEE Transactions on **C-31** (1982) 913 - 917.
- [59] M.A. Marsan, G. Conte, and G. Balbo, a Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. ACM Transactions on Computer Systems (TOCS) . ACM New York, NY, USA **2** (1984) 93 - 122
- [60] P.J. Haas, Stochastic Petri Nets: Modelling, Stability, Simulation Springer, 2002.

- [61] L. Kleinrock, *Queueing Systems. Vol. 2, Computer Applications*, Wiley New York, 1976.
- [62] G. Haring, C. Lindemann, M. Reiser, and S. Balsamo, *Product Form Queueing Networks. Performance Evaluation: Origins and Directions, Lecture Notes in Computer Science* **1769** (2000) 377-402.
- [63] K.M. Chandy, U. Herzog, and L.S. Woo, *Parametric Analysis of Queueing Networks. IBM Journal of Research and Development* **19** (1975) 36-42.
- [64] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1984.
- [65] J.R. Jackson, *Jobshop-like Queueing Systems. Management Science* **50** (2004) 1796-1802.
- [66] P.J. Burke, *The Output of a Queueing System. Operations Research* **4** (1956) 699-704.
- [67] W.J. Gordon, and G.F. Newell, *Cyclic Queueing Networks with Exponential Servers. Operations Research* **15** (1967) 254-265.
- [68] J.P. Buzen, *Computational Algorithms for Closed Queueing Networks with Exponential Servers. Communications of the ACM* **16** (1973) 527 - 531.
- [69] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios, *Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. J. ACM* **22** (1975) 248-260.
- [70] W.J. Oates, *Manufacturing Modeling Using RESQ, Proceedings of the 16th conference on Winter simulation IEEE Press Piscataway, NJ, USA, Dallas, TX 1984, pp. 356-359.*
- [71] D. Potier, and M. Veran, *QNAP2: A Portable Environment for Queueing Network Modelling, the french national institute for research in computer science and control., Le Chesnay, Yvelines 1984.*
- [72] H. Beilner, J. Mater, and C. Wysocki, *The Hierarchical Evaluation Tool HIT. in: D. Potier, and R. Puigjaner, (Eds.), 7th Int. Conf. on Modelling Techniques and Tools for Computer Perf. Evaluation 1994.*
- [73] M. Reiser, *Mean-value Analysis of Closed Multichain Queueing Networks. Journal of the ACM (JACM)* **27** (1980) 313-322.
- [74] J.P. Buzen, *Fundamental Laws of Computer System Performance, Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation ACM New York, NY, USA Cambridge, Massachusetts, United States 1976, pp. 200-210.*
- [75] P.J. Denning, and J.P. Buzen, *Operational Analysis of Queueing Networks. ACM Computing Surveys (CSUR)* **10** (1978) 225-261.
- [76] H. Hermanns, U. Herzog, and J.P. Katoen, *Process Algebra for Performance Evaluation. Theor. Comput. Sci.* **274** (2002) 43-87.
- [77] N. Gotz, U. Herzog, and M. Rettelbach, *Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis using Stochastic Process Algebras Performance Evaluation of Computer and Communication Systems* **729** (1993) 121-146.
- [78] H. Hermanns, U. Herzog, and J.P. Katoen, *Process Algebra for Performance Evaluation. Theoretical Computer Science* **274** (2002) 43-87.
- [79] N. Halbwachs, D. Peled, H. Hermanns, V. Mertsiotakis, and M. Siegle, *TIPTool: Compositional Specification and Analysis of Markovian Performance Models. Computer Aided Verification* **1633** (1999) 683-683.
- [80] S. Gilmore, and J. Hillston, *The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling, Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer*

- Performance Evaluation, Springer-Verlag New York, Inc. Secaucus, NJ, USA, Vienna, Austria 1994, pp. 353-368.
- [81] M. Bertoli, G. Casale, and G. Serazzi, JMT: Performance Engineering Tools for System Modeling. *ACM SIGMETRICS Performance Evaluation Review* **36** (2009) 10-15.
- [82] V. Cortellessa, A. Di Marco, and P. Inverardi, Three Performance Models at Work: A Software Designer Perspective, *Proceedings of FOCLASA 2003, the Foundations of Coordination Languages and Software Architectures, Electronic Notes in Theoretical Computer Science*, Elsevier B.V., CONCUR 2003, Marseille, France, 2004, pp. 219-239.
- [83] B. Plateau, J.M. Fourneau, and K. Lee, PEPS: a Package for Solving Complex Markov Models of Parallel Systems. *Modelling techniques and tools for computer performance evaluation* **397** (1989) 291.
- [84] P. Hughes, and D. Potier, The Integrated Modelling Support Environment, ESPRIT II-IMSE Project 1989.
- [85] M. Bertoli, G. Casale, and G. Serazzi, The JMT Simulator for Performance Evaluation of Non-product-form Queueing Networks, *Proceedings of the 40th Annual Simulation Symposium (ANSS'07)*, IEEE Computer Society, Norfolk, Virginia 2007, pp. 3-10.
- [86] M. Bertoli, G. Casale, and G. Serazzi, An Overview of the JMT Queueing Network Simulator, Politecnico di Milano - DEI, 2007.
- [87] D.d.E.e. Informazione, Java Modelling Tools - Users manual, v.v.0.5, 2009.
- [88] M. Woodside, G. Franks, and D.C. Petriu, The Future of Software Performance Engineering. in: L.C. Briand, and A.L. Wolf, (Eds.), *Proceedings of the International Conference on Software Engineering, 2007 Future of Software Engineering IEEE Computer Society Washington, DC, USA 2007*, pp. 171-187.
- [89] R. Kazman, G. Abowd, L. Bass, and P. Clements, Scenario-Based Analysis of Software Architecture. *IEEE Software* **13** (1996) 47-55.
- [90] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, The Architecture Tradeoff Analysis Method, *Proceedings of the 21st international Conference on Software Engineering, ACM, New York, NY, Los Angeles, California, United States, , 1999*, pp. 54-63.
- [91] C.U. Smith, and L.G. Williams, Performance Engineering Evaluation of Object-Oriented Systems with SPE* ED. *Computer Performance Evaluation Modelling Techniques and Tools, Lecture Notes in Computer Science* **1245** (1997) 135-154.
- [92] L.G. Williams, and C.U. Smith, Performance Evaluation of Software Architectures, *Proceedings of the 1st international workshop on Software and performance ACM Press New York, NY, USA, Santa Fe, New Mexico, United States 1998*, pp. 164-177.
- [93] V. Cortellessa, and R. Mirandola, PRIMA-UML: a Performance Validation Incremental Methodology on Early UML Diagrams. *Special issue on unified modeling language (UML 2000)* **44** (2002) 101-129.
- [94] F. Paetsch, A. Eberlein, and F. Maurer, Requirements Engineering and Agile Software Development, *Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises IEEE, 2003*, pp. 308-313.
- [95] A. Eberlein, and J.C.S. do Prado Leite, Agile Requirements Definition: A View from Requirements Engineering, *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02)*, Germany, 2002.
- [96] D.d.E.e. Informazione, Java Modelling Tools, Dipartimento di Elettronica e Informazione, Performance Evaluation Modelling v.0.7.4, 2009.

- [97] P. King, and R. Pooley, Using UML to Derive Stochastic Petri Net Models, Proceeding of the 8th Int. Workshop on Petri Net and Performance Models PNPM '99, Zaragoza, Spain, , 1999, pp. 45-56.
- [98] J.P. López-Grao, J. Merseguer, and J. Campos, From UML Activity Diagrams to Stochastic Petri nets: Application to Software Performance Engineering, Proceedings of the 4th international workshop on Software and performance ACM Press New York, NY, USA, Redwood Shores, California 2004, pp. 25-36.
- [99] G.P. Gu, and D.C. Petriu, Early Evaluation of Software Performance Based on the UML Performance Profile, Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research IBM Press, Toronto, Ontario, Canada 2003, pp. 66-79.
- [100] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens, Performance Modelling With UML and Stochastic Process Algebras. IEE Proceedings: Computers and Digital Techniques **150** (2003) 107–120.
- [101] A.J. Bennett, and A.J. Field, Performance Engineering With the UML Profile for Schedulability, Performance and Time: a Case Study, IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), IEEE Computer Society Washington, DC, USA Volendam, NL, 2004, pp. 67-75.
- [102] S. Balsamo, R. Mamprin, and S. Marzolla, Performance Evaluation of Software Architectures With Queuing Network Models. in: C. Bobenau, (Ed.), Proceeding of the European Simulation and Modeling Conference (ESMc'04), Paris, France, 2004, pp. 206-213.
- [103] V. Cortellessa, A. D'Ambrogio, and G. Iazeolla, Automatic Derivation of Software Performance Models from CASE Documents. Performance Evaluation **45** (2001) 81-105.
- [104] D.C. Petriu, and X. Wang, From UML Descriptions of High-Level Software Architectures to LQN Performance Models. Applications of Graph Transformations with Industrial Relevance **1779** (1999) 217-221.
- [105] R. Pooley, and P. King, Unified Modelling Language and Performance Engineering. IEEE proceedings. Software **146** (1999) 1-10.
- [106] L.B. Arief, and N.A. Speirs, A UML Tool for an Automatic Generation of Simulation Programs, Proceedings of WOSP 2000, ACM Press New York, NY, USA, 2000, pp. 71-76.
- [107] E. Gomez-Martinez, and J. Merseguer, ArgoSPE: Model-Based Software Performance Engineering. Petri Nets and Other Models of Concurrency - ICATPN 2006, Lecture Notes in Computer Science **4024** (2006) 401-410.
- [108] S. Bernardi, S. Donatelli, and J. Merseguer, From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models, Proceedings of the 3rd international workshop on Software and performance ACM Press New York, NY, USA, Rome, Italy 2002, pp. 35-45.
- [109] A. Abdullatif, and R. Pooley, From UML Design to Markov Chain Performance Models, UK Performance Engineering Workshop (UKPEW06), Bournemouth University, Poole, UK., 2006.
- [110] U. Force, OMG UML Specification, Object Management Group, 1999.
- [111] F. Bause, and P. Buchholz, Protocol Analysis Using a Timed Version of SDL, Proceedings of the 3rd Int. Conf. on Formal Description Techniques (FORTE '90), Springer, 1991, pp. 269-285.
- [112] N. Rico, and G. von Bochmann, Performance Description and Analysis for Distributed Systems Using a Variant of LOTOS. in: B. Jonsson, J. Parrow, and B. Pehrson, (Eds.), Proceedings of the IFIP WG6.1 International Symposium on

- Protocol Specification, Testing and Verification XI North-Holland Publishing Co. Amsterdam, The Netherlands, The Netherlands, 1991, pp. 199-213.
- [113] P. Pukite, and J. Pukite, Markov Modeling for Reliability Analysis, Wiley-IEEE Press, 1998.
- [114] isograph-software, MKV- Markov Analysis Software v.3.0, 1993.
- [115] Kishor S. Trivedi, SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator, International Conference on Dependable Systems and Networks, Washington, D.C., USA 2002.
- [116] W.J. Stewart, MARCA: Markov Chain Analyzer. A Software Package for Markov Modelling,v.3.0, 1996.
- [117] J. Clark, XSL Transformations (XSLT) W3C 1999.
- [118] M. Bertoli, G. Casale, and G. Serazzi, Java Modelling Tools: an Open Source Suite for Queueing Network Modelling and Workload Analysis, QEST 2006, IEEE Press, Riverside, US, 2006, pp. 119-120.
- [119] L.G. Williams, and C.U. Smith, Information Requirements for Software Performance Engineering. Quantitative Evaluation of Computing and Communication Systems, Lecture Notes in Computer Science **977** (1995) 86-101.
- [120] R. Mordani, J.D. Davidson, and S. Boag, Java API for XML Processing, ,Sun Microsystems, V1.1, 2001.
- [121] ISO, Financial Transaction Card Originated Messages - Part 1: Messages, Data Elements and Code Values, International Organization for Standardization, 2003.
- [122] e.C. Corporation, IST/SWITCH, eFunds Canada Corporation, payment software suite,v.7.4, 2004.
- [123] T. Wengraf, Qualitative Research Interviewing: Biographic Narrative and Semi-Structured Methods, Sage Pubns Ltd, 2001.
- [124] V. Cortellessa, M. Gentile, and M. Pizzuti, Xpmit: An XMI-based Tool to Translate UML Diagrams Into Execution Graphs and Queueing Networks, Proceedings. First International Conference on the Quantitative Evaluation of Systems,. QEST 2004. , University of Twente, Enschede, The Netherlands., 2004, pp. 342-343.
- [125] J.R. Lewis, IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. International Journal of Human-Computer Interaction **7** (1995) 57-78.
- [126] ISO, Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Guidance on Usability 9241-11, International Organization for Standardization, 1998.
- [127] J.R. Lewis, Psychometric Evaluation of an After-scenario Questionnaire for Computer Usability Studies: the ASQ. ACM SIGCHI Bulletin **23** (1991) 78-81.
- [128] J.P. Chin, V.A. Diehl, and K.L. Norman, Development of an Instrument Measuring User Satisfaction of the Human-computer Interface, Proceedings of the SIGCHI conference on Human factors in computing systems ACM New York, NY, USA, Washington, D.C., United States 1988, pp. 213-218.
- [129] J. Kirakowski, and M. Corbett, SUMI: The software usability measurement inventory, John Wiley & Sons, 2006, pp. 210-212.
- [130] G.E.P. Box, J.S. Hunter, and W.G. Hunter, Statistics for Experimenters: Design, Innovation, and Discovery, Wiley-Interscience New York, 2005.
- [131] A. AlAbdullatif, and R.Pooley, Automating State Marking Methodology for Extracting Performance Model from Design Model, Saudi Innovation Conference 2007 Newcastle University Newcastle 2007.

- [132] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, Model-based Performance Prediction in Software Development: a Survey. *IEEE Trans. Softw. Eng.* **30** (2004) 295-310.
- [133] S. Balsamo, and M. Simeoni, Deriving Performance Models from Software Architecture Specifications. *ACM SIGSOFT Software Engineering Notes* **27** (2001) 6–9.
- [134] A. D’Ambrogio, and G. Iazeolla, Design Of XMI-Based Tools For Building EQN Models Of Software Systems. in: P. Kokol, (Ed.), *Proceeding Software Engineering - 2005*, ACTA Press, Innsbruck, Austria, 2005.
- [135] C.P. Dorina, and S. Hui, Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications, *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools, Lecture Notes in Computer Science*, Springer London, UK, , 2002.

End of Document