MINA - A Tool for MSC-Based Performance Analysis and Simulation of Distributed Systems

Dissertation

zur Erlangung des Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

vorgelegt dem Fachbereich 5
Institut für Informatik und Wirtschaftsinformatik
der Universität Duisburg-Essen (Campus Essen)

von
Hesham Kamal Arafat Mohamed, geboren in Kalyoubia, Ägypten

Datum der mündlichen Prüfung: 15. April 2005

Gutachter:

Prof. Dr. Bruno Müller-Clostermann
(Universität Duisburg-Essen)

Prof. Dr. Micheal Goedicke
(Universität Duisburg-Essen)

To the memory of my fater

To my mother,

To my wife and my children

Acknowledgement

First, I would like to thank my supervisor Prof. Dr. Bruno Müller-Clostermann, for his inspiring and encouraging way to guide me to a deeper understanding of knowledge work, and his invaluable comments during the whole work with this dissertation. Without his encouragement and constant guidance, I could not have finished this dissertation. He was always there to meet and talk about my ideas, to proofread and mark up my papers and chapters, and to ask me good questions to help me think through my problems. Beside my supervisor, I would like to thank the rest of my thesis Committee: Prof. Dr. Klaus Echtle and Prof. Dr. Michael Geodicke

This work has been inspired by related work done in the IPonAir/MxRAN project. I would like to acknowledge the contributions of Peter Schefczik, Michael Söllner, Wilfried Speltacker (all with Lucent Technologies, Nuremberg), Andreas Mitschele-Thiel (Technical University of Ilmenau), Georgios Nikolaidis (University of Athens) and Anja Wiedemann (University of Duisburg-Essen).

Thanks also to all my colleagues at Institute for Computer Science and Business Information Systems for providing a good working atmosphere.

Thanks also to the Egyptian government for providing me with the financial support for my scholarship.

Last, but not least, I thank my family: my parents, my brother, my sisters, my beloved wife, my kids, who serve as an inspiration for me to move on against all odds on my way.

Abstract

Performance analysis can help to address quantitative system analysis from the early stages of the system development life cycle, e.g., to compare design alternatives or to identify system bottlenecks. This thesis addresses the problem of performance evaluation of distributed systems by employing a viewpoint where analytical and simulative evaluation techniques are unified in the MINA tool to make use of both techniques. We suggest a modelling tool chain to evaluate the performance of distributed systems like computer and communication systems based on an MSC description of the system.

MSC-based performance evaluation of distributed systems is an approach that uses performance models, which are based on an MSC description of a system to evaluate system performance measures. To determine the system performance, these descriptions can be extended by notions for time consumption and resource usage and afterwards be included in a system performance model. Based on this unique model specification, analytical as well as simulative techniques can be applied to achieve either quick mean value results by queueing networks analysis or confidence intervals or transient measures by simulation.

The applicability to real world systems and the advantages of the tool has been demonstrated by a large application example in the field of mobile communication systems, and its effectiveness has been evaluated by comparing it with other approaches. The experimental results show that the tool is scalable, the way it can model simple as well as complex systems. Moreover, it is straightforward and has the ability to find reasonable solutions in an efficient manner.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ALCAP	Access Link Control Application Part
API	Application Program Interfaces
ATS	Applied Telecommunication Symposium
BP	Basic Procedures
C/S	Client Server
CN	Core Network
CPU	Central Processing Unit
CS	Call Setup
DB	Data Base
E2E Delay	End-to-End delay
FCFS	First Come First Served
FE	Functional Entities
GUI	Graphical User Interface
HMSC	High level Message Sequence Charts
I/O	Input Output
IP	Instruction Pointer
IS	Infinite Server
ITU	International Telecommunication Union
JDBC	Java Data Base Connectivity
JDK	Java Development Kit
LAN	Local Area Network
LCFS	Last Come First Served

MSC	Message Sequence Charts
MTBF	Mean Time Between Failures
MVA	Mean Value Analysis
NBAP	Node B Application Part
ODBC	Open Data Base Connectivity
PMSC	Performance Message Sequence Chart
PS	Processor Sharing
QN	Queuing Network
QSDL	Queuing SDL
RNC	Radio Network Controller
RPC	Remote Procedure Call
RR	Round Robin
RRC	Radio Resource Control
SDL	Specification and Description Language
SF	System Functions
SIRO	Service in Random Order
SQL	Structured Query Language
TCP	Terminal Connection Point
TTCN	Tree and Tabular Combined Notation
UE	User Equipment
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
VBA	Visual Basic Application

Chapter 1

Introduction

1.1 Introduction

Functional correctness as well as performance behaviour are essential properties for the design and development of complex systems, such as communication protocols and real time applications. Hence, system developers often need predictions on quantitative measures like throughput and response time to decide on implementation design alternatives. On many occasions, guarantees on performance properties concerning behaviour over time measures are required before system implementation [68].

Since a formal specification has to be as implementation-independent as possible, specification languages like SDL (Specification and Description Language) and MSC (Message Sequence Chart) do not (and should not) cover performance aspects. To obtain performance measures and/or verified statements on the behaviour over time of the system in question, quantitative properties of the design have to be specified. In other words, one has to move towards the implementation to obtain some kind of performance information from the design. Performance aspects may cover issues like performance characteristics of hardware devices, concurrency due to shared resources, algorithms used for data manipulation and scheduling, processing speeds, bandwidths of communication channels, buffer sizes, timeout values, and last but not least workload and traffic characterization [29].

The standard MSC language does not support the modelling of quantitative aspects.

While the functional requirements of an MSC specification may be examined at an early stage during system design, the investigation of quantitative properties can only be done at a much later stage in the system development process, resulting in excessive costs for the correction of performance related design errors. Moreover, designers are confronted with a methodological gap between functional and quantitative analysis of a system, as existing methods for quantitative system analysis require the transformation of the MSC specification into a different model world. Such transformations are expensive and prone to error. Since the design of complex systems usually is an iterative process, the results of a performance analysis have to be retransformed back into the MSC model world, so that they can be integrated into the design of the system, leading to even more costs and errors [69].

As mentioned above, it is necessary to extend MSC by a number of features to incorporate performance modelling. In the following, the main concepts of these extensions are introduced. At first, an MSC is mapped into queueing models, which are a widespread paradigm for performance modelling. They are used to describe and analyze the congestion of multiple requests for restricted resources. In the queueing model, each system component (represented by an instance in the MSC context) is considered as a queueing station providing a service to the packet or signal or the request (represented by a message in the MSC context). The total time a request spends in a queueing station depends on the amount of service required, the speed of the server and additionally the wait time spent in the queue. The requested amount of service is normally described by a random variable, whereas the speed of the server is a real positive constant. The waiting time spent in the queue depends on the congestion due to concurrent usage of the system component. Associating these queueing models by performance parameters like server speeds and service amount, one can evaluate the system performance by analytical or simulative techniques. An important topic that highly affects system performance is the behaviour of the traffic sources. Hence, the characterization of the workload by suitable traffic parameters is part of each quantitatively assessable model.

1.2 Performance Evaluation of Computer and Communication Systems

Performance is defined as "quality of service, provided the system is correct" [48].

Performance modelling involves representing the probabilistic nature of user demands and predicting the system capacity to perform, under the assumption that the system structure remains constant.

Performance is a key criterion in the design, procurement, and use of computer and communication systems. As such, the goal of computer systems engineers, software engineers, scientists, analysts, and users is to get the highest performance for a given cost. To achieve that goal, computer systems professionals need, at least, a basic knowledge of performance evaluation terminology and techniques. Anyone associated with computer systems should be able to state the performance requirements of these systems and should be able to compare different alternatives to find the one that best meets their requirements [34].

Performance evaluation aims at forecasting system behaviour in a quantitative way. Whenever new systems are to be built or existing systems have to be reconfigured or adapted, performance evaluation can be employed to predict the impact of architectural or implementation changes on the system performance [26].

In the following, we discuss the goals of performance evaluation of computer and communication systems. Types of performance measures are also discussed. Finally, we give an overview of some performance evaluation techniques.

1.2.1 Goals of Performance Evaluation

Roughly speaking, the field of performance evaluation covers three related aspects:

- Determine certain performance measures for existing systems or for models of systems.
- Develop new analytical and methodological foundations of performance evaluation, e.g. seek for advances in queueing theory or time series analysis of measurement and simulation results.
- Find ways to apply theoretical approaches in creating and evaluating performance models.

Why are we interested in performance aspects of a system? In practice, one is often faced to certain performance-related problems, for which we discuss some examples, which have been partly taken from [34 and 77]:

- Some existing equipment is perceived to be too slow or not responsive enough. A performance evaluation study can reveal performance bottlenecks and provide hints on which system components should be improved.
- More general, it is often important to identify bottlenecks to guide the optimization of an existing or planned design.
- Capacity planning: given some anticipated load, how much resources should be assigned / bought to obtain some desired level of service quality? As an example: given some estimates for the load, how much memory should an internet router have in order to keep the packet loss due to congestion below 0.1%?
- Comparison of algorithms: Given two algorithms or protocols to solve the same problem, which is the better algorithm/protocol under which load situations? For example, when looking at an internet router carrying almost only TCP traffic, what is the best policy to drop packets, Drop-Tail or random early dropping? On the other hand, which scheduling algorithm provides the best responsiveness for a given system load?
- Given two offers for a computer system: how to find out which one serves my needs best?
- You made a contract with an Internet service provider, which guarantees you a minimum bandwidth. How can you find out if the service provider violates the contract?

Sometimes these questions arise out of pure interest or are part of a research study; however, more often the answers are needed to make business decisions, e.g. to decide on investments, directions of research and development.

1.2.2 Performance Measures

The performance measures of interest vary from application to application and from user to user. In this section, we discuss some classifications of performance measures and provide examples for typical measures used in many performance evaluation studies.

We can broadly distinguish between system-oriented performance measures, which can be assessed independently from applications, and application-oriented performance measures, which belong to a specific application and might depend in complex ways on system-oriented measures. Let us take video-conferencing as an example. The application-oriented performance measures might be:

- Frame rate, resolution, and colour depth.
- High signal-to-noise ratio and absence of distortions (dropouts, coding artefacts, etc.).
- Interactivity, i.e. round-trip times, response time.

The video frames have to be transported over a network, which can be characterized, e.g. by the following system-oriented performance measures:

- Throughput.
- Delay and jitter.
- Losses.

Even if we know the system-oriented measures, it is not obvious how to predict the application-oriented measures from these. For example, the video application might apply more or less clever error concealment techniques to combat packet losses and to increase the perceived video quality [34 and 77]. In computer networks like the Internet often the following performance measures are of interest:

- The delay is in general concerned with time. The following delay measures are often used:
  - ✓ The *end-to-end delay* measures the time needed by a packet to travel from the source via intermediate hops to the receiver.
  - ✓ The *round-trip-time* includes the acknowledgement, which has to travel back from the receiver to the transmitter. The transmitter measures the time between

issuing the packet and receiving the acknowledgement.

- ✓ When considering OSI service primitives, the *indication delay* might indicate the time between issuing a request primitive at the transmitter and the occurrence of the corresponding indication primitive at the receiver.
- ✓ The *confirmation delay* indicates the time at the receiver, which passes between issuing the request primitive and getting the confirmation primitive.

- ➢ The *jitter* denotes delay variation. Roughly speaking, the jitter characterizes the deviation from a strictly isochronous service required by control applications or multimedia applications.
- ➢ The *throughput* denotes how many user bytes of data packets go through the network per time. The notion of *goodput* is similar, but explicitly excludes control information or control packets from the calculations.
- ➢ The *utilization* of a communications link denotes the fraction of time by which the link is actually used, i.e. where it is not idle. Typically, service providers are interested in high utilizations to justify investments.
- ➢ The *blocking probability* gives the probability that a service request is not served due to busy resources. For example, when you want to place a telephone call, you sometimes get no dial tone or a busy tone from the beginning on.

The performance measures for computer systems are in some parts similar, in other parts different from the typical measures used in computer networking. According to [27] we can roughly distinguish between *desktop systems*, *server systems* and *embedded systems:*

- ➢ In desktop systems, the most important measures reflect their principal use: such a system is often used only by a single user, who cares most about response times, not so much on throughput. The response times and the "interactiveness" are best served with high performance processors and graphics systems, while I/O bandwidth plays not such a big role. The response time denotes the time between issuing a request (e.g. hitting ENTER after typing a command at the console) and getting the answer.
- ➢ In server systems, the focus is not so much to serve a single user as fast as possible,

but to serve multiple users as fast as possible, i.e., throughput is of greater importance than the response time of a single user. Furthermore, since many enterprises depend critically on their servers (consider a web-shop as an example); they have to be *reliable* and *available.* Typical reliability measures are the mean time between failures (MTBF) and a typical availability measure is the system downtime per year.

- In embedded systems the following factors are often of importance: power consumption, since many embedded systems are battery driven; memory consumption, since embedded systems have to be low-cost; real-time performance measures (jitter for periodic traffic, frequency of deadline misses, interrupt latencies, etc.), since embedded computers often run control applications.

Finally, for measures like delay, throughput, error rates, etc., we might be interested in the following characteristics: mean value, variance (or more general: moments), minimum and maximum values, the whole distribution, certain quantiles and correlation between different samples.

1.2.3 Main Performance Evaluation Techniques

To assess the performance of one or multiple systems, we have to apply performance evaluation techniques. We can broadly distinguish three main techniques [26, 34 and 77]:

- When the system under study already exists and is accessible with reasonable effort, *measurement-based techniques* can be used.
- When the system does not exist or is too large (complex, unhandy, not available, etc.), a *performance model* has to be developed.

To develop a performance model, we should at least have an unambiguous *system description.* From this system description, we can then make an abstract *model.* In the context of performance evaluation, a model is an abstract description, based on (mathematically) well-defined concepts, of a system in terms of its components and their interactions, as well as its interactions with the environment. The environment part in the model describes how the system is being used, by humans or by other systems. Very often, this part of the model is called the system *workload model.*

The process of designing models is called modelling. This can be either an *analytical model,* which uses mathematical concepts and mathematical notations to describe the model. In contrast, a *simulation model* is a computer program, which mimics the important aspects of the system under study. We will briefly discuss each technique in turn.

1.2.3.1 Measurement-based

In a measurement, the system under study (which can be a single computer or multiple computers, network elements, etc.) is first *instrumented* with so-called *probes.* A probe is a piece of hardware or software, which captures certain system states and stores them in a buffer. The system is then subjected to a specified *workload* and the measurement starts. A *monitor system* collects the data from the probes and computes performance measures, analyzes and displays them.

As stated before, measurements are only possible when the system under study already exists and is accessible. Furthermore, it must be possible to install the probes. For example, when the probes are pieces of software, which need to be part of the operating system kernel, then you are restricted to open-source software.

It is often not easy to interpret measurement results, since many factors influence the results. For example, when measuring the delay of voice-over-IP packets between a local host and a remote host, several factors influence the measured delays, amongst them are:

- Speech coder latency.
- Operating system and networking stack at the transmitter.
- The network in between (bandwidths, queueing delays due to cross-traffic, etc.).
- Operating system and networking stack at the receiver.
- Size of play out buffer.
- Speech decoder latency.

It is not obvious how much each part influences the observed delays. However, the same example illustrates one key advantage of measurements: you can just take the measured values and declare them “your true numbers”. If you have to build an analytical or simulation model, you have to come up with "reasonable delay numbers" for each of the elements in the transmission chain. In general, these "reasonable numbers" are often hotly debated and the problems in finding them lead to some feeling of mistrust in model-based techniques. In measurements, it is mainly the selection of the workload, which is questioned by others. The time needed to set up a measurement varies, but should not be underestimated.

A serious measurement study can take a lot of time. Furthermore, many details have to be considered, e.g. the specific configuration of a computer system. Sometimes the measurement results can change dramatically after modifying the configuration data.

1.2.3.2 Analytical Modelling

Analytical modelling consists of setting up mathematical models and equations, which describe certain aspects of the system. Specifically for modelling of computer systems and communication networks, *probabilistic models* are often used to describe the evolution of systems. This choice accounts for the fact that the workloads observed in reality are often unpredictable, and hence are assumed random. In this text, we focus entirely on stochastic models of discrete-state systems.

The big advantage of analytical modelling is that it requires a thorough understanding of the system. The biggest disadvantage is that many systems are too complex to be in reach of analytical methods. Therefore, a lot of simplifications and approximations have to be made during the modelling process, which lessens the accuracy of the results. However, analytical models can often give a rough feeling for the influence of certain factors on the performance measures. A second disadvantage is that the analyst needs to know the necessary mathematics very well, including the respective abilities to model certain technical phenomena.

1.2.3.3 Simulation Modelling

A simulation model is a computer program written in a general-purpose language or in a special simulation-oriented language. A simulation implements the most important aspects of the original, often in a simplified and abstract manner. However, the advantage of simulation modelling over analytical modelling is that it allows for a greater level of detail and it allows avoiding too many simplifications. In analytical models, the level of detail is often restricted by the limited expressiveness of the analytical method, while for simulation models the available time and resources are the only restriction. If the simulations use stochastic input data, great care must be taken to achieve a desired level of statistical accuracy for the simulation results. In fact, one important question is for how long a simulation has to run, and this is often not trivial to decide and may need a long time.

A big advantage of simulations over measurement-based techniques (which can be exploited even for existing systems) is that simulations are much better reproducible than measurements. For example, when doing measurements of the error rates on a wireless channel, these are not reproducible, since the errors depend very much on the propagation environment found by transmitted waves. Small movements of obstacles (turn around a monitor, close a door, etc.) can change the error behaviour dramatically. It is close to impossible to control the environment and to reproduce it elsewhere. Another example: if you measure the time needed for a certain program to run on a UNIX computer, it is greatly influenced by the mix of other user programs and demons currently running on the system. In contrast, in a simulation you have tight control over all the input to the model.

1.2.3.4 Analytical vs. Simulation Modelling

If the model at hand fulfils a number of requirements, we can directly calculate important performance measures from the model by using analytical techniques. Analytical techniques are of course very convenient, but not many real systems can be modelled in such a way that the requirements are fulfilled. However, we will spend quite some time on deriving and applying analytical techniques. The reasons for this are, among others, that they can give a good insight into the operation of the systems under study at low cost, and that they can be used for "quick engineering" purposes in system design.

Within the class of analytical techniques, a sub-classification is often made. First, there are the so-called closed-form analytical techniques. With these, the performance measure of interest is given as an explicit expression in terms of the model structure and parameters. Such techniques are only available for the simplest models. A broader class of techniques are analytic numerical techniques, or numerical techniques, for short. With these, we are able to obtain (systems of) equations of which the solution can be obtained by employing techniques known from numerical analysis, e.g. by iterative procedures. Although such numerical techniques do not give us closed-form formulae, we still can obtain exact results from them, of course within the error tolerance of the computer, which is used for the numerical calculations.

For the widest class of models that can be imagined, analytical techniques do not exist to obtain model solutions. In these cases we have to resort to simulation techniques in order to solve the model, i.e., in order to obtain the measures of interest. With simulation, we mimic the system behaviour, generally by executing an appropriate simulation program. When doing so, we take time stamps, tabulate events, etc. After having simulated for some time, we use the time stamps to derive statistical estimates of the measures of interest.

It is also possible to combine the above modelling approaches. This is called hybrid modelling. In such an approach, parts of the model are solved with one technique and the obtained results are used in combination with the other model parts and solved by another technique.

The presented classification of solution techniques is not unique, nor beyond debate. Very often also, the performance models are classified after the techniques that can be used to solve them, i.e., one then speaks of analytical models or of simulation models. It is difficult to state in general terms which of the three solution techniques is best. Each has its own merits and drawbacks. Analytical techniques tend to be the least expensive and give the modeller deep insight into the main characteristics of the system. Unfortunately, real systems often cannot be adequately modelled by analytically tractable models. Approximate analytical models can be an outcome; however, their validity is often limited to a restricted range of parameters. Numerical techniques, as an intermediate between pure analytical and simulative techniques, can be applied in very many cases. Using simulation, the modeller is tempted to make the models too complex since the model solution technique itself does not bring about any restrictions in the modelling process. This might easily lead to very large and expensive simulation models.

1.3 Background and Related Work

Several approaches for integrating performance evaluation and formal specification techniques have been reported in the literature. A good survey on these methods is reported in [53]. Mitschele-Thiel et al. [50 and 51] described a toolbox called DO-IT toolbox to support performance engineering of SDL/MSC-based systems including model derivation, model-based performance evaluation and optimization. The performance evaluation within the DO-IT toolbox is based on MSC rather than on SDL. An annotated extension of MSC is used to define the performance requirements including the workload, and the resource requirements for specific execution of the system. The performance evaluation techniques provided by the DO-IT toolbox are rather simple and based on deterministic service times. The proposed techniques include bottleneck analysis, critical path analysis and deterministic simulation.

In [43], the Performance Message Sequence Chart (PMSC) language extends MSC-96 by annotations to integrate performance aspects. Annotations have semantical meanings for performance evaluation tools as developed, e.g. at the University of Erlangen-Nuremberg [23]; annotations are comments in the original language to allow standard tools to process the specification. PMSC is described in earlier versions in [19 and 20]. PMSC introduces a concept of time for an executed MSC by interpreting MSC events as actions that are executed by tasks, which need some time to complete. Every task has a start and completion events that occur at some point in time. In PMSC, a system model is used that has two separate sub-models, namely the load model and the machine model. The load model includes the MSC, which describes the functional dependencies between load units, the machine requirements, which are annotated with every load unit (action), and the traffic sources, that specify the intensity of the load. The machine model consists of queueing stations that model processors or channels between processors. To complete the system model a mapping from instances on modelled processors and communication paths on modelled channels must be obtained. To allow flexibility, the concepts are separated in different documents.

Many approaches do exist to enhance formal description techniques by non-functional information on time and resources. In the field of SDL and MSC, an overview on the role of performance aspects is given in [49, 52 and 54]. Examples for tools combining the SDL and/or MSC methodology and performance evaluation are QUEST, SPEET, and SPECS [25, 28, 43, 52 and 54]. Much work has also been especially done with respect to Timed MSCs [52 and 54] and Performance MSC [43].

Researchers at the University of Essen developed a queuing SDL tool, called QUEST [16 and 17]. QUEST is based on the adjunction of time-consuming machines that model the congestion of processes due to limited resources. By adding workload models after defining a mapping of workload to machines, an assessable performance model is automatically generated. The language QSDL (Queuing SDL) and the tool QUEST support the description and construction of performance models and their evaluation. The language QSDL provides means for the specification of load, machines and their binding. QSDL processes model load by issuing time-consuming requests that are referred for execution to adjunct machines given by queuing stations. QSDL processes are bound to the machines via links and pipes. Processes and machines within the same block are connected with a link. The translation of the QSDL description to an executable simulation program is done automatically.

There are some approaches to integrate time and performance into MSC. [70] extends MSC-92 (MSC-Real Time) by language constructs rather than by annotations. [72] introduces an extension of MSC-96, called Timed MSC, to support performance testing. Performance simulation based on formalized use cases with a language similar to MSC-96 is reported in [14]. A tool that uses MSC-96 for deriving performance models in early phases of the object-oriented software engineering process is described in [73]. In [40] a formal timed semantical model based on term rewriting rules is introduced for MSC-92. Most approaches to support specification based performance evaluation of systems in the SDL/MSC context extend SDL itself (e.g. the approach described in [17]). Since SDL and MSC are often combined in one project SDL-based and MSC-based performance prediction should be integrated and share common documents to support consistency between both specifications.

Here, in this dissertation, we follow the ideas sketched above; in particular, we will use MSCs notions, which are extended by annotations to describe required resource consumptions. The instances are assumed to run on resources, which have a certain processing speed. This way a performance model is established which could be quantitatively evaluated, either by discrete event simulation or by queueing network algorithms. Here we mainly follow the latter approach to calculate efficiently mean values for end-to-end delays and resource utilizations. Moreover, a simulation tool has been developed which allows evaluating models, which do not satisfy the necessary assumptions to obtain analytical solutions. Additionally to the evaluation of the steady state behaviour, simulation can also be used to study the dynamic performance behaviour.

1.4 Thesis Organization

In this work, we developed a tool called MINA to evaluate the performance of distributed systems by analytic as well as by simulative techniques. The rest of the thesis is organized in a way that describes step by step the tool chain described in Figure 1.

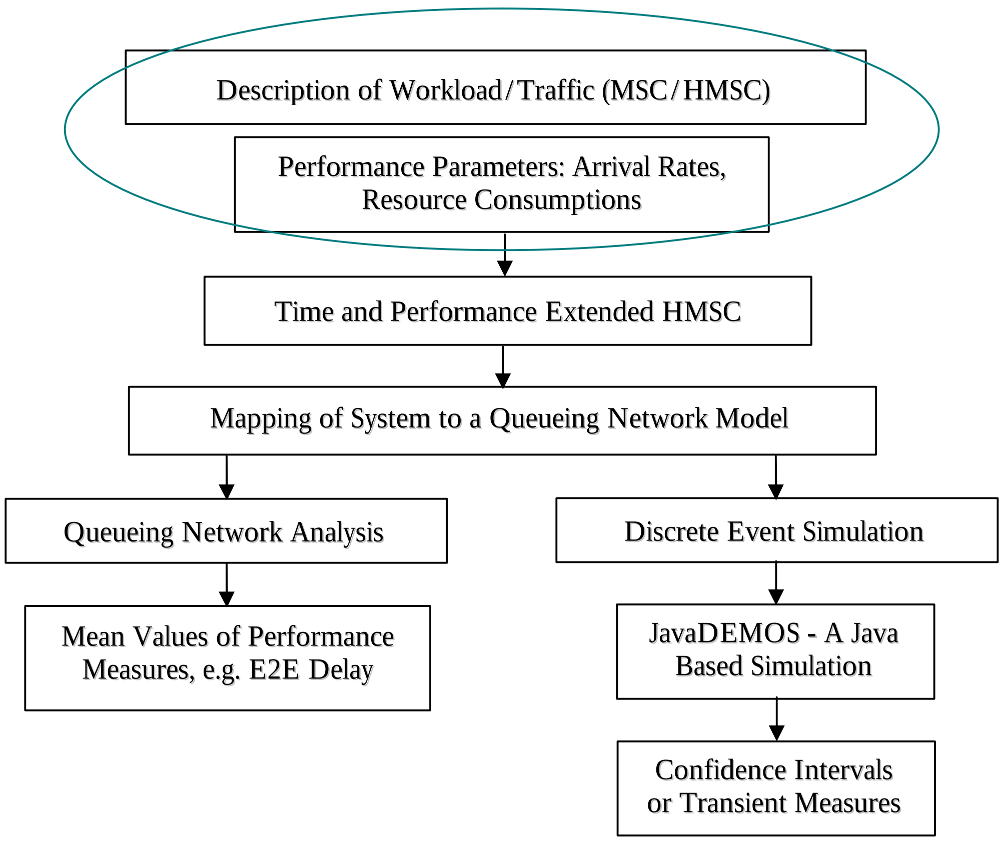

Description of Workload/Traffic (MSC/HMSC)
Performance Parameters: Arrival Rates, Resource Consumptions
Time and Performance Extended HMSC
Mapping of System to a Queueing Network Model
Queueing Network Analysis
Discrete Event Simulation
Mean Values of Performance Measures, e.g. E2E Delay
JavaDEMOS - A Java Based Simulation
Confidence Intervals or Transient Measures

Figure 1: An overview of the tool chain

The tool chain starts with describing the system workload by MSCs. Then, notions for time consumption and resources are added in order to extend MSCs. This produces the so-called performance extended MSC. The “performance extended MSC” is included in a system performance model (Queueing Network Model). Based on this model the performance evaluation of the system under consideration can be done by analytical techniques or by discrete event simulation. Analytical techniques are used to obtain steady state performance measures like resource utilizations, throughput, and end-to-end delays. Additionally, simulation allows for the investigation of dynamic performance behaviour.

According to this tool chain the rest of the thesis is organised as follows. Chapter 2 discusses the approach of describing communication between distributed instances by MSCs and how these descriptions can be extended by notions for time consumption and resource usage and can be afterwards included in a system performance model. Chapter 3 describes how such models can be evaluated under reasonable assumptions by analytical queueing network algorithms and how steady state performance measures like resource utilizations and end-to-end delays can be calculated. Chapter 4 describes how the simulation uses the same input like the analytical formulas and how it allows for the investigation of dynamic performance behaviour. Chapter 5 illustrates the basic ideas by simplified case studies examples taken from the field of computer systems. Modelling the applicability of the complete tool to model and characterize the performance of real industrial systems is shown by a real world example in Chapter 6. Finally, a summary of contributions of the work presented in this thesis is summarized in Chapter 7 and suggestions for future work are given. Appendix A presents in brief the JavaDEMOS package used to build the simulator. For the sake of completeness, Appendix B introduces the basics of client/server systems. A summary of the MINA tool chain is given in Appendix c.

Chapter 2

Describing Systems by MSCs

2.1 Introduction

The purpose of Message Sequence Charts (MSCs) is to provide a trace language for the specification and description of the communication behaviour of system components and their environment by means of message interchange. Communication between distributed instances can be described by MSCs. To determine the system performance, these descriptions can be extended by notions for time consumption and resources. Afterwards they may be included in a system performance model. Such models can be evaluated by discrete event simulation or under reasonable assumptions alternatively with analytical queueing network algorithms.

In section 2.2, we introduce Message Sequence Chart (MSC) including the definition of Message Sequence Charts, the main characteristics of Message Sequence Charts, the basic constituents of Message Sequence Charts, the MSC language notations (the graphical and the textual notations) and the High Level MSC (HMSC). We also show that HMSCs provide a mean to graphically define how a set of MSCs can be combined to express scenarios that are more complicated. Section 2.3 shows how MSCs are extended with time and resource parameters. The mapping of the extended MSCs into a queueing performance model is described in section 2.4.

Finally, in section 2.5 we show how the performance parameters together with the messages flow between system components are used by analytical models and simulation models to evaluate the system performance measures.

2.2 MSC (Message Sequence Charts)

2.2.1 Introduction to MSCs

A message sequence chart (MSC) is a high-level description of the message interaction between system components and their environment. A major advantage of the MSC language is its clear and unambiguous graphical layout, which immediately gives an intuitive understanding of the described system behaviour.

The syntax and semantics of MSCs are standardized by ITU-T, as recommendation Z.120. Message Sequence Charts (MSC) is a language to describe the interaction between a set of independent message-passing instances. The main characteristics of the MSC language are the following, cf. [58 and 59]:

- MSC is a scenario language. An MSC describes the order in which communications and other events take place. Additionally, it allows for expressing restrictions on transmitted data values and on the timing of events.
- MSC is a graphical language. The two-dimensional diagrams give overview of the behaviour of communicating instances.
- MSC is a formal language. The definition of the language is given in natural language as well as in a formal notation.
- MSC is a practical language, which is applicable. MSC is used throughout the engineering process.
- MSC supports structured design. Simple scenarios can be combined to form specifications that are more complete by means of High-Level Message Sequence Charts.
- MSC is often used in conjunction with other methods and languages. Its formal definition enables formal and automated validation of an MSC with respect to a model described in a different language. MSC can be used, for example, in combination with SDL (Specification and Description Language) and TTCN (Tree

and Tabular Combined Notation).

The basic constituents of Message Sequence Charts are instance, message, general ordering, condition, timer, action, instance creation and termination.

Here our focus is on MSCs consisting only of instances and messages. The most fundamental language constructs of MSC, are *instances* (e.g. entities of SDL systems, blocks, processes and services).

Instances are reactive entities whose communication behaviour is described by the MSCs. The message exchange is the only mean of communication among instances. Within the instance body, the ordering of events is specified. A message can be as simple as a signal or as complex as a sophisticated data packet. Each message is associated with a send and a receive event. To illustrate the basic ideas, a simple MSC-example, which has four instances that exchange five messages between each other, is shown in Figure 2.

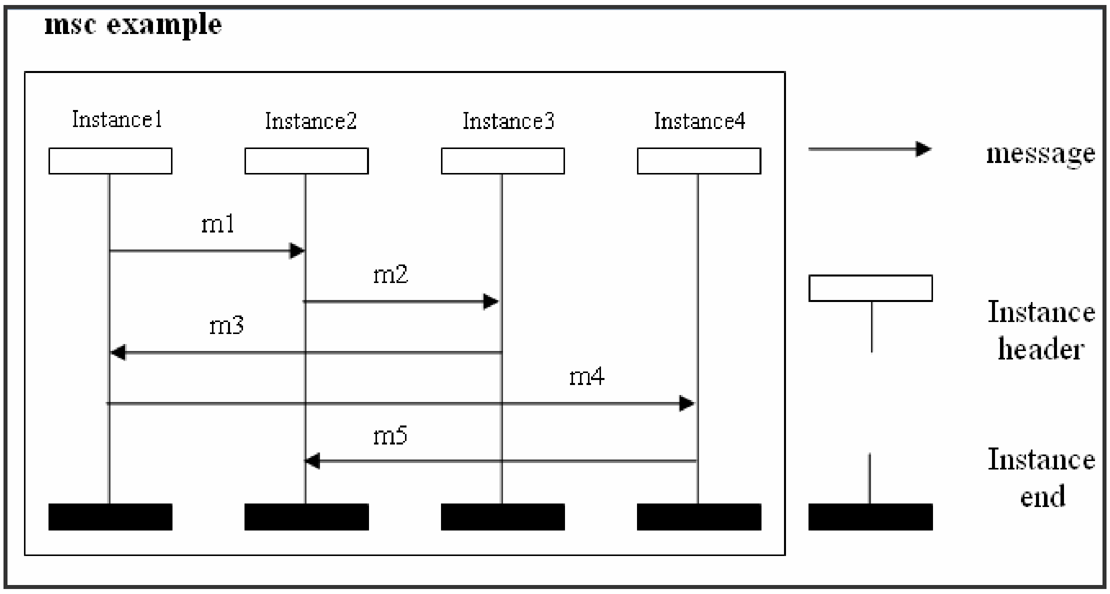


Figure 2: An example of an MSC (Graphical notation)

Message Sequence Charts have both a graphical and a textual representation. The language is best illustrated by the graphical representation, but where the definition of a formal semantics is concerned, the textual representation is preferred.

The textual notation MSC-PR can be expressed in two forms, *event-oriented* which describes the MSC using the order in which the events or *instance-oriented* which describes the MSC on an instance-by-instance basis. The syntax of both the *instance-oriented* form and the *event-oriented* form for the MSC mentioned above in Figure 2 is shown in Figure 3 (a) and Figure 3 (b) respectively.

```
msc Example
 inst instance1, instance2, instance3, instance4;
  instance instance1:
    out m1 to instance2;
    in m3 from instance3;
    out m4 to instance4;
  endinstance;
  instance instance2:
    in m1 from instance1;
    out m2 to instance3;
    in m5 from instance4;
  endinstance;
  instance instance3:
    in m2 from instance2;
    out m3 to instance1;
  endinstance;
  instance instance4:
    in m4 from instance1;
    out m5 to instance2;
  endinstance;
endmsc:
```

```
msc Example
 inst instance1, instance2, instance3, instance4;
   instance1: out m1 to instance2;
   instance2:  in m1 from instance1;
   instance2: out m2 to instance3;
   instance3:  in m2 from instance2;
   instance3: out m3 to instance1;
   instance3:  endinstance;
   instance1: out m3 to instance3;
   instance1: out m4 to instance4;
   instance1:  endinstance;
   instance4:  in m4 from instance1;
   instance4:  out m5 to instance2;
   instance4:  endinstance;
   instance2: out m5 to instance4;
   instance2: endinstance;
 endmsc;
```

Figure 3: MSC-PR: (a) Instance-oriented form, (b) Event-oriented form

2.2.2 Basic MSC vs. HMSC (High level MSC)

The core language of Message Sequence Charts is called *Basic Message Sequence Charts*. A Basic Message Sequence Chart concentrates on communications and local actions only. The body of a Basic Message Sequence Chart is formed by a finite collection of instances. An instance is an abstract entity on which message outputs, message inputs and local actions may be specified.

To define more complex scenarios, the HMSC provides a mean to graphically define how a set of MSCs can be combined. An HMSC is a directed graph where different types of nodes can be found. Each node could be one of the following [58, 59 and 61]:

- ➢ An HMSC reference (a component) consists of a frame with rounded corners enclosing the name of the referenced HMSC.
- ➢ Every component has exactly one start node, indicated by an upside-down triangle. In addition, it may contain a number of end nodes depicted by a triangle and several

HMSC references.

- Every node including the end-nodes within a component is reachable from the start node.
- An arrow between two HMSC references implies that they are composed vertically.
- Splitting of an arrow denotes that the successors are alternatives.
- A cycle connecting a number of HMSC references expresses a repetition. In this case, infinite behaviour can be described. Connectors (indicated by a circle) are also used for combining incoming and outgoing edges. The various compositional operators of HMSC are sketched below.

2.2.3 HMSC Composition

MSCs can be composed via operators into HMSCs. MSCs are identified in HMSCs by so-called MSC-references. To gain flexibility these HMSCs can be MSC-references themselves. The various compositional operators of an HMSC are described below; cf. [44 and 61]:

- **Sequencing:** Whenever two MSCs are sequenced or concatenated, it is interpreted to be vertically composed (Figure 4). Two variants of sequence operators are described below:
    - ✓ *Strong sequencing:* Ml and M2 are in strong sequencing if the transfer to M2 is possible only after the termination of all events in Ml.

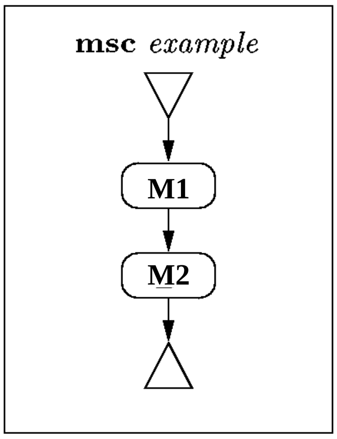


Figure 4: Vertical composition in HMSC

- ✓ *Weak sequencing:* Let Ml and M2 be weakly sequenced in that order. Let Ml and M2 share the instance I. Then weak sequencing means that all events on instance I of Ml will come before all events on instance I from M2. For events on instances, which are not shared, by Ml and M2 the order is arbitrary (just like for parallel merge). Moreover, if Ml and M2 share instances I, and J, may be there still events in M2 on J occurring before events on I in M1.

- ➢ **Alternatives:** If a node has more than one outgoing arrow this indicates a number of alternatives with which this node can be composed vertically. The HMSC given in Figure 5 shows that M1 is composed vertically with either M2 or M3.

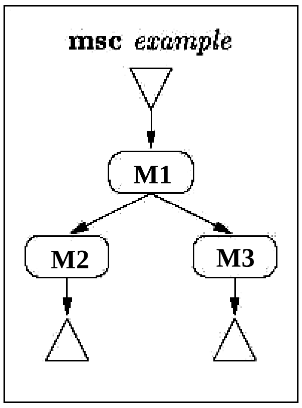


Figure 5: Alternatives in an HMSC

- ➢ **Parallel composition:** this is also called horizontal composition, and it means that multiple MSCs run in parallel. There is no restriction among multiple MSCs. Figure 6 illustrates the graphical representation of parallel composition in an HMSC.

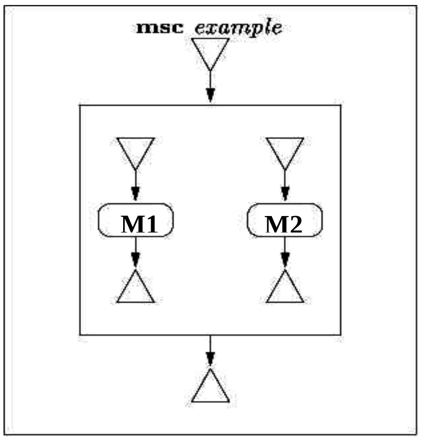


Figure 6: Parallel composition in HMSC

- **Loops**: A "Loop" is used to represent the possible execution of an MSC an arbitrary number of times with the possibility of termination. A loop results by the vertical composition of the last node with the first node, creating the loop. Figure 7 illustrates the graphical representation of loops in an HMSC.

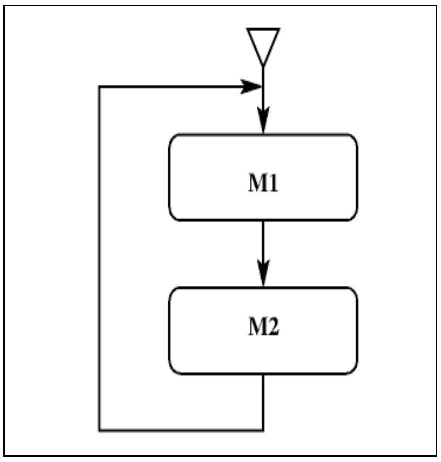


Figure 7: Loop in HMSC

Other HMSC operators are repetition, option, and exception. The operators, option and exception are only abbreviations that can be encoded using (delayed) choice. Similarly, finite repetition can be encoded using (delayed) choice and (weak) sequencing essentially by unfolding of the loop. Through the partial order of MSC events, a set of (totally ordered) traces is specified by one plain MSC.

An HMSC with only finite loops can be seen as the definition of a set of plain MSCs where, the sequential composition glues MSCs together, choice is a set of all possible branches and parallel composition is a set of all possible combinations of free merges where the precedence between MSC events in each MSC is preserved.

2.3 Extending the MSC by Performance Parameters

In order to construct quantitatively assessable models based on the MSC-notion we extend MSCs by performance parameters. This can be done in a rather straightforward way. Each message is associated with a service amount a_i to be executed at the receiving instance (resource or station) *i*. Each instance has a speed g_i, such that the service time is simply calculated by $s_i = a_i / g_i$. Of course we can group messages into classes and distinguish them, say by *index r*, *r = 1, ..., R*; hence we get the notion $s_{ir} = a_{ir} / g_i$, describing the service time of a message of class *r* at station *i*, where a_{ir} is the service amount of a message of class *r* at station *i*. Furthermore, we consider the instances to behave like queueing stations, i.e. messages arriving at a busy instance are stored in a queue and will have to wait for service.

Figure 8 displays the execution of a timed MSC; each message has to spend some wait time at arrival at an instance (including the case of zero wait time) followed by a service time which depends on the speed of the instance and the required service amount.

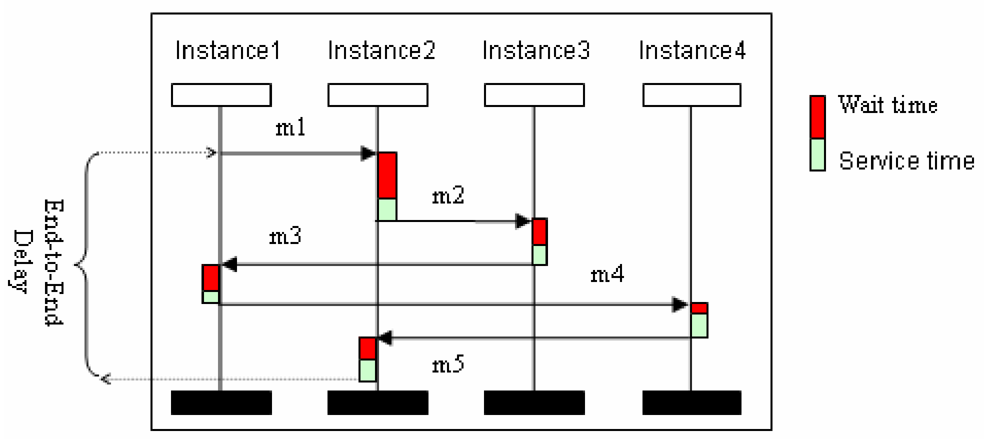


Figure 8: Wait and service times during execution of a timed MSC

Moreover we consider MSCs to be “open“, i.e. the start of an MSC is triggered from the environment according to some interarrival distribution. Since we will employ analytical mean value formulas based on queueing network theory the interarrival distribution is assumed to be negative exponential. The same assumption is made for the distribution of service amounts. By combining MSCs using the HMSC operators of composition, traces that are more complex can be defined. Therefore, we can define end-to-end delays also for HMSCs; this is done in Chapter 6.

2.4 Mapping the Extended MSC to a Performance Evaluation Model

Here we describe the derivation of a model, which can be quantitatively assessed by means of analytical or simulative techniques. Since instances are queueing stations and messages can be considered as customers, we obtain a queueing network. Each queueing station consists of a wait queue and a server. Messages are generated according to an arrival rate ?

and they are served at the stations 1 through 4, and finally they leave into a sink.

Depending on the interarrival distribution of messages, the service time distribution of the messages and the service disciplines of the stations such a network has the so called product form property and can be solved analytically, i.e. performance measures, like utilization of stations or response time can be derived very fast.

Theory and algorithms are well established; indeed, in this scenario we have a queueing network of Jackson type [32 and 33]. More discussion about queueing networks algorithms is introduced in Chapter 3.

Note that the numbering of messages defines their order of execution, here "source → m1→ m2 → m3 → m4 → m5 → sink". On the other hand, the queueing network formulas[1] to be applied here do neglect the correct order of visits. What really matters when deriving mean performance measures, is the number of visits (not their order) and the amount of requested service at the stations.

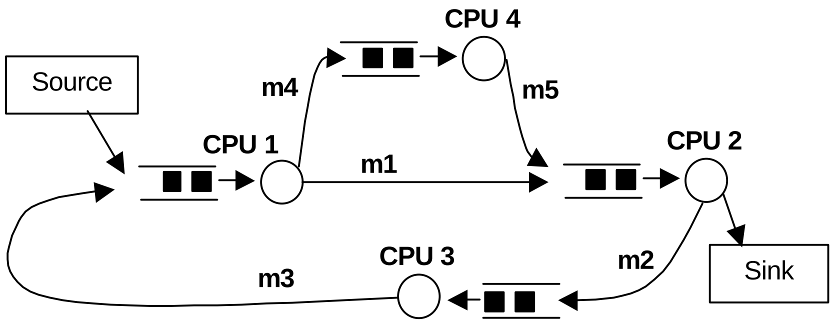


Figure 9: The example MSC transformed into a queueing network model

Here we assume that each station is of type -/M/1-FCFS and MSC arrivals occur according to a Poisson stream and the service times of the messages are also negative exponentially distributed.

In general, we assume that a system consists of *n* stations and *m* different MSCs classes, which arrive with an overall MSC arrival rate *?* [MSCs/sec]. Each MSC consists of a certain number of messages, which are to be served by the different stations. The arrival rate at

[1] In case any of these assumptions is not valid, we have to use approximation algorithms or discrete event simulation.

station *i* is h_i *?* [messages/sec] where, h_i is the number of messages to be served by station *i, i* = 1, 2, …, *N* for all MSCs of the classes *r, r* = 1, 2, …, *R*. Let c_{ir} be the number of messages of an MSC of class *r* which are served at station *i*, then we can define h_i, the total number of messages received and served by station *i,* as follows:

$$h_i = \sum_{r=1}^{R} c_{ir}, \quad i = 1, 2, \mathrm{K}\ , N, \ \textit{for all stations i} \tag{1}$$

The same way, we can define the message arrival rates $?_{i,r}$ [messages/sec] at station *i* for all messages of MSC class *r* as follows:

$$l_{ir} = c_{ir}\, l, \quad i = 1, 2, \mathrm{K}\ , N, \quad r = 1, 2, \mathrm{K}\ , R \tag{2}$$

Hence, the overall arrival rate of message $?_i$ at station *i* for all messages of all MSC classes is:

$$l_i = h_i\, l, \quad i = 1, 2, \mathrm{K}\ , N \tag{3}$$

Let μ_{ir} be the service rate (messages of MSC class *r* / sec) at station *i, i = 1, 2, …, N*. The associated service rates are defined by the following equation:

$$\mu_{ir} = 1/s_{ir} = g_i\ /a_{ir} \text{ [messages/sec]} \tag{4}$$

In Chapter 6, we have a more complex scenario that some MSCs are composed in parallel and the messages of these MSCs belong to different classes.

2.5 Input Parameters

Input parameters are the parameters used as input to the simulation model as well as input to the queueing network model. These input parameters are as follows:

- ➢ *The performance parameters*:
  - ✓ *The resource parameters:* Like the resource speed, this is used to determine the service times for messages of a certain MSC.
  - ✓ *Complexity class:* As shown in the previous section messages of the same MSC and/or different MSCs are belonging to classes to distinguish between them. Each class has a different service amount and therefore a different service time.

  - ✓ *The service time:* It is the time needed to serve a message at a resource, e.g. like a CPU. Note that the service time does not include the queueing time, which arises when the resource is busy and accordingly the message waits for service in a queue until the resource becomes free. It depends on the previous two parameters, i.e. the service time is dependent on the complexity class of the message to be served and on the speed of the resource that will serve this message.

- ➢ *The interarrival time*: The interarrival time is the time between two successive arrivals of a certain MSC. In the case that we have more than one MSC, which are composed, each MSC has its own interarrival time.

- ➢ *Visit counts*: This is the number of messages that visit the resource to be served in the time unit. To compute the visit counts, we count the number of messages of each complexity class at each resource. The visit count is used to determine the utilizations of the resources and the response times for different MSCs in the case of the queueing network model.

- ➢ *Message flow parameters*:
  - ✓ *Sender resource:* This is defined as the resource that sends the message. It is important to know the resource that sends a message to free it after sending the message to be available for serving other messages.
  - ✓ *Receiver resource:* When a resource receives a message, it spends some time to serve it. Therefore, it is important to keep these parameters in tables to use it to schedule the messages in the right order during the simulation and to get the right numbers of visit count to be used in the queueing network model.

As discussed before the performance parameters together with the interarrival time parameters and the visit count parameters are used for calculating the system performance measures like the resources utilization and the response time for all MSCs in the case of queueing network analysis.

All parameters are saved as text in files except the visit counts. The complexity parameters and the flow parameters are saved in one file and using this table, the visit count parameters are calculated. The interarrival parameters are saved in another excel sheet and the service times parameters are saved in a third one. Parts of these sheets are shown in the examples of section 4.5 and Chapter 6.

The way of reading the messages in the correct order to be scheduled in the simulation model will be discussed in details in the following chapters.

2.6 Output Parameters

Output parameters are the parameters obtained from the simulation as well as the analytical queueing network analysis. The output parameters of interest are as follow:

- *The average response time*: The response time is the time needed to complete an MSC. This is the time between the start of the first message of an MSC until the end of the last message of the same MSC.
- *The end-to-end delay*: The end-to-end delay is the time needed to complete all MSCs of the system, in case we have a large system described by more than one MSC.
- *The system throughput*: The system throughput is defined as the number of MSCs that complete per unit of time.
- *The resource utilization*: The resource utilization is defined as the fraction of time that the resource is busy.

Chapter 3

Queueing Network Analysis of MSC-based Models

3.1 Introduction

Queueing network models have been extensively applied to represent and analyze resource sharing systems such as communication and computer systems and they are powerful and versatile tool for system performance evaluation and prediction [6].

Queueing network models are used as performance evaluation models of congestion systems, such as production, communication and computer systems. They provide a simple model at a high level of abstraction, intuitively understandable and they can clearly represent resource contention. System performance evaluation with queueing network models consists in the definition and parameterization of the model to evaluate a set of figures of merit that are performance indices, such as resource utilization, system throughput and customers' response time. Analytical techniques are of course very convenient. The big advantage of analytical modelling is that it requires a thorough understanding of the system.

Analytical models can give a rough feeling for the influence of certain factors on the performance measures. Analytical queueing network algorithms results can be obtained very quickly, e.g. mostly in some seconds whereas in the case of simulation we may have runs for long hours or sometimes even for days. So using queueing networks algorithms in early design stages has a great advantage, e.g. system developers can investigate the scope of possible parameter settings, e.g. traffic intensity, and allow a better planning of simulation scenarios which include more details and are closer to reality. Of course this is difficult to be done using simulation which needs more CPU time than the queueing networks algorithms to give the similar results.

The rest of this chapter is organized as follows, sections 3.2 and 3.3 describes in brief queueing systems including both single station and queueing networks. The analysis of queueing stations as well as queueing networks to calculate performance measures like utilization of queueing stations and response times for each MSC is described in sections 3.4-3.6. Material of sections 3.2-3.6 follows closely the literature in the queueing networks textbooks, cf. [1, 13 and 24].

Section 3.6 describes the queueing network model we deal with. It is an open queueing network, which consists of a set of service stations. Each service station is either a multiclass station or a single class station with First Come First Served (FCFS) or Infinite Server (IS) queueing disciplines, Poisson arrival process and exponential service time distribution. In the case of multiclass stations, the mean service times for different customer classes may have different values. Each service station may have one or more servers.

Section 3.6 also describes how to calculate the utilization at each queueing station in the queueing network model we deal with and the response times for each MSC. To do these calculations many methods are used. *Jackson* method [32 and 33] is used in the case that the queueing stations have a FCFS queueing disciplines, have a single class of customers and have a single server. An extension to the method of Jackson, the *BCMP* method [7], is used for networks that have queueing stations of queueing disciplines rather than FCFS like for example IS (Infinite Server) and PS (Processor Sharing).

The open queueing networks with M/M/m queueing stations, FCFS queueing discipline, multiclass of customers with different service rates for different classes of customers do not satisfy the conditions of Jackson's method and the BCMP method and we can not use these methods to analyze such queueing networks. For this reason, a non-product form approximate method called *decomposition* method [12, 15, 24, 41, 67, 74 and 75] is used.

Some remarks on how to apply the queueing networks formulas in some special cases are discussed in section 3.7. A remark on how to deal with the queueing networks formulas in the case that messages can be distinguished not only by the complexity class of the MSC it belongs to but also if the message itself belongs to a certain complexity class as well.

A discussion about the use of the SHRINK approach [2, 2, 63, 64 and 66] with the analytical formulas in case of the so called slow down models is described. Another remark about branching in an MSC and about how to calculate the response time for a certain predefined branch is also discussed. Finally, an overview of how to calculate the end-to-end delay in the case of systems described by HMSCs is given in section 3.8.

3.2 Single Station Queueing Systems

A single station model is described by an arrival process of incoming customers, a service process, a buffer space (queue) for holding the waiting customers, a scheduling algorithm for the queue and one server (see Figure 10) or more servers (see Figure 11) that provide the service to customers.

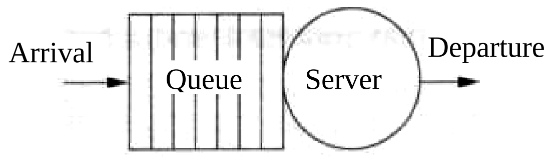


Figure 10: Graphical notations for a resource and its queue

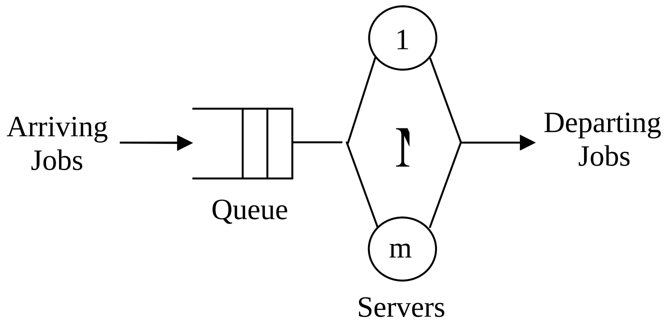


Figure 11: Service station with m servers (a multiple server station)

Figure 11 illustrates a single station with multiple servers. A server can only serve one customer at a time and hence, it is either in a "busy" or an "idle" state. If all servers are busy upon the arrival of a customer, the newly arriving customer is buffered, assuming that buffer space is available, and waits for its turn. When the customer currently in service departs, one of the waiting customers is selected for service according to a queueing (or scheduling) discipline.

Queueing stations are described by the so-called *Kendall notation* A/B/m/K-scheduling discipline. The interarrival and service time distributions are given by A and B respectively, m denotes the number of servers, and K is the capacity, i.e. the numbers of customers a queueing station can hold. The following symbols are normally used for A and B:

- **M:** Exponential distribution (memoryless property).
- **E_k:** Erlang distribution with k phases.
- **H_k:** Hyperexponential distribution with k phases.
- **C_k:** Cox distribution with k phases.
- **D:** Deterministic distribution, i.e., the interarrival time or service time is constant.
- **G:** General distribution.
- **GI:** General distribution with independent interarrival times.

Additionally a scheduling discipline may be specified. Some commonly used queueing disciplines are:

- **FCFS** (First-Come-First-Served): If no queueing discipline is given in the Kendall notation, then the default is assumed to be the FCFS discipline. The customers are served in the order of their arrival.
- **LCFS** (Last-Come-First-Served): The customer that arrived last is served next.
- **SIRO** (Service-In-Random-Order): The customer to be served next is selected at random.
- **RR** (Round Robin): If the servicing of a customer is not completed at the end of a time slice of specified length, the customer is preempted and returns to the queue, which is served according to FCFS. This action is repeated until the customer service is completed.
- **PS** (Processor Sharing): This strategy corresponds to round robin with infinitesimally small time slices. It is as if all customers are served simultaneously

and the service time is increased correspondingly.

- **IS** (Infinite Server): There are an ample number of servers so that no queue does exist.
- **Static Priorities**: The selection depends on priorities that are permanently assigned to the customer. Within a class of customers with the same priority, FCFS is used to select the next customer to be processed.
- **Dynamic Priorities**: The selection depends on dynamic priorities that alter with the passing of time.
- **Preemption**: If priority or LCFS discipline is used, then the customer currently being processed is interrupted and preempted if there is a customer in the queue with a higher priority.

Depending on the type of parameters, it is possible to derive closed analytical formulas for utilization, wait and response time.

3.3 Queueing Networks

A queueing network model is a collection of service stations representing the system resources that provide service to a collection of customers that represent the users. The customers' competition for the resource service corresponds to queueing into the service stations.

A queueing network may be open, closed or mixed:

- **Open**: A queueing network is called open when customers can enter the network from outside and customers can also leave the network. Customers can arrive from outside the network at every node and depart from the network from any node.
- **Closed:** A queueing network is called closed when customers can neither enter nor leave the network. The number of customers in a closed network is constant. A network in which a new customer enters whenever a customer leaves the system can be considered as a closed one.
- **Mixed:** If a queueing network contains both open and closed classes, then it is said

to be a mixed network.

Informally, a queueing network is defined by:

- **Type of service stations:** Service station characteristics include the service time, the buffer space with its queueing scheduling and the number of servers.
- **Customers:** Customers are described by their number for closed models and by the arrival process to each service centre for open models, the service demand to each service centre and the types of customer.
- **Network topology:** Network topology models how the service stations are interconnected and how the customers move between them.

In other words, different types of customer in the queueing network model can model different behaviour of customers. This allows representing various types of external arrival of customers, different service demands and different types of network routing (or different visit counts). Customers may have a different service times and different routing probabilities. Hence, the network may have multiple customer classes (multiclass network).

If no customers of a particular class enter or leave the network, i.e., the number of customers of this class is constant, then the customer class is said to be closed. A customer class that is not closed is said to be open. If a queueing network contains both open and closed classes, then it is said to be a mixed network.

3.4 Performance Measures

The analysis of the queueing network models consists of evaluating a set of performance measures, such as resource utilization and throughput and customer response time. The different types of queueing systems are analyzed mathematically to determine performance measures from the description of the system. Because a queueing model represents a dynamic system, the values of the performance measures vary with time.

Normally, however, we are content with the results in the steady state. The system is said to be in steady state when all transient behaviour has ended, the system has settled down, and the values of the performance measures are independent of time. The system is then said to be in statistical equilibrium, i.e., the rate at which customers enter the system is equal to the rate at which customers leave the system. Such a system is also called a stable system. The most important performance measures are:

- **p_k:** The probability of the number of customers in the system
- **?:** The *utilization* of a queueing station and in the case of a single server it equals the fraction of the time in which the server is occupied. In case there is no limit on the number of customers in the single server queue, the server utilization is given by:

$$r = \frac{arrival\ rate}{service\ rate} = \frac{l}{m} \tag{5}$$

In case of multi server, it is defined as follows:

$$r = \frac{l}{mm} \tag{6}$$

where m is the number of servers. The previous formulas are hold only under the following condition (it is called the stability condition)

$$r < 1 \tag{7}$$

- **?:** The *throughput ?* is defined as the mean number of customers whose processing is completed in a single unit of time, i.e., the departure rate. Since the departure rate is equal to the arrival rate *?* for a queueing system in statistical equilibrium, the throughput is given by:

$$l = m \cdot r \cdot m \tag{8}$$

- ***T*:** The *response time T*, also known as the sojourn time, is the total time that a customer spends in the queueing system.
- ***W*:** The *waiting time W* is the time that a customer spends in a queue waiting to be served. Therefore, we have:

Response time = waiting time + service time.

The mean response time $\overline{T}$ is calculated by using the following formula:

$$\overline{T} = \overline{W} + \frac{1}{m} \tag{9}$$

- ***Q*****:** The *queue length* Q is the number of customers in the queue.
- ***K*****:** K represents the *number of customers* in the queueing system.

The mean number of customers in the queueing system K and the mean queue length Q can be calculated by Little's theorem:

$$\overline{K} = l\,\overline{T} \qquad (10)$$

and

$$\overline{Q} = l\,\overline{W} \qquad (11)$$

Little's theorem is valid for all queueing disciplines and arbitrary GI/G/m systems.

3.5 Single Queueing Station Analysis

In this section, we show the response times for some simple queueing stations in the steady state[2]. These results or solutions, which are called "stationary solutions[3]", are available in closed-form. We show here the response time for queueing stations of types M/M/1, M/M/m and M/M/8 which we use to model our stations in both communication and computer systems. Transient solutions for other queueing systems like M/M/l/K, M/G/l, GI/M/l, GI/M/m and GI/G/l are not discussed here.

3.5.1 Single Server Stations (M/M/1)

In the M/M/1 systems, the arrival process is Poisson, the service times are exponentially distributed, and there is a single server. Assuming that the arrival rate *?* and the service rate *μ* satisfy the stability condition *?* < *μ* then the mean response time is given by the following formula:

$$\overline{T} = \frac{1/m}{1-r} \qquad (12)$$

[2] The system is said to be in steady state when all transient behaviour has ended, the system has settled down, and the values of the performance measures are independent of time.

[3] Solutions obtained when a system is in a steady state are called Stationary solutions.

3.5.2 Multi Server Stations (M/M/m)

In the M/M/m queueing system, we have m servers. Each server has service rate *µ* with arrival rate *?*. The condition for the queueing system to be stable is *? < mµ*. The individual server utilization, *?* = *?* / (*mµ*).

The mean response time is given by the following formula:

$$\overline{T} = \frac{1}{l}\left[mr + \frac{r}{1-r}\cdot P_m\right] \quad , \qquad (13)$$

where the steady-state probability that an arriving customer has to wait in the queue is given by:

$$P_m = \frac{(mr)^m}{m!(1-r)}\cdot p_0 \qquad (14)$$

and the steady-state probability of no customers in the system is given by:

$$p_0 = \left[\sum_{k=0}^{m-1}\frac{(mr)^k}{k!} + \frac{(mr)^m}{m!(1-r)}\right]^{-1} \qquad (15)$$

3.5.3 Infinite Server Stations (M/M/8)

In an M/M/8 queueing system, we have a Poisson arrival process with arrival rate *?* and an infinite number of servers with service rate *µ*. The mean response time is given by the following formula:

$$\overline{T} = \frac{1}{m} \qquad (16)$$

3.6 Performance Measures of Queueing Network Models

3.6.1 Description of Queueing Network Models

Our queueing network model is an open queueing network, which consists of a set of service stations. Each service station is either a multiclass or a single class with First Come First Served (FCFS) or Infinite Server (IS) queueing disciplines, Poisson arrival process and exponential service time distribution. In the case of multiclass stations, the mean service times for different customers' classes have different values. Each service station may have one or more servers.

In the following, we will show how to calculate the utilizations of each queueing station and the response times for different MSCs.

3.6.2 Computing the Utilization

Under the definitions of the arrival, rate and the service time given by equations 1-4 (see section 2.4) we can easily compute mean values for stationary performance measures like station utilizations and response times.

The utilization *?*$_{ir}$ of station *i* with respect to MSC class *r* in the case of single server queueing stations, is defined as:

$$r_{ir} = \frac{l_{ir}}{m_{ir}} = \frac{c_{ir}\, l}{m_{ir}}, \quad i = 1, 2, \mathrm{K}\ , N, \quad r = 1, 2, \mathrm{K}\ , R \qquad (17)$$

and for a queueing station with multi-servers m_i, it is given by the following formula:

$$r_{ir} = \frac{c_{ir}\, l}{m_i m_{ir}}, \quad i = 1, 2, \mathrm{K}\ , N, \quad r = 1, 2, \mathrm{K}\ , R \qquad , \qquad (18)$$

where *?* is the overall MSC arrival rate and c_{ir} be the number of messages of an MSC of class *r* which are served at station *i*.

The overall utilization *?*$_i$ for station *i* with respect to all MSC classes is given by the following equation:

$$r_i = \sum_{r=1}^{R} r_{ir}, \quad i = 1, 2, \mathrm{K}\ , N \qquad (19)$$

3.6.3 Computing the Response Times

The response times for each MSC class and the overall end-to-end delay (*E2E*) for the execution of all MSCs can also be computed but it depends on the queueing disciplines and whether the queueing network is a single class or multiclass network.

Note that we calculate performance measures for the execution of MSCs, i.e. response time refers to the mean execution time of one MSC including wait times as well as service times. Link delays are only included if links are explicitly modelled as instances. Performance measures for single messages are not considered; moreover, not all messages belonging to the same MSC are distinguished. An extension to distinguish between messages would lead to a three-indexed service amount, say a_{irk}. This will be shown in section 3.7.

In the following, we will show how to calculate the response time using a product form method like Jackson's method and the BCMP methods, which are constrained to certain networks. In the case of multi class of messages (customers), service rates are different according to different message classes, in this case, our problem does not fit to any product form method and we have to use some approximate formulas to get the results.

3.6.3.1 Jackson's Method (A Product Form Method)

Jackson's theorem [32 and 33] is concerned only with networks of single-server queues having exponentially distributed service times. The theorem states that the steady state queue occupancy distribution is the product of the individual queue distributions when each queue is treated as an independent M/M/1 queue with the appropriate arrival rate. For this reason, networks of single server queues with exponential service times and Poisson arrival rates from the "outside world" are called *product form networks*. Jackson examined open queueing networks and found product-form solutions [32 and 33]. The networks examined fulfil the following assumptions [13]:

- There is only one customer class in the network.
- The overall number of customers in the network is unlimited.
- Each of the N nodes in the network can have Poisson arrivals from external sources. A customer can leave the network from any node.
- All service times are exponentially distributed.
- The service discipline at all nodes is FCFS.
- The i^{th} node consists of $m_i = 1$ identical service stations with the service rates μ_i, $i =$

1, …, *N*. The arrival rates *?*$_{0i}$, as well as the service rates, can depend on the number k_i of customers at the node. In this case, we have load-dependent service rates and load-dependent arrival rates.

Under the previous conditions, the nodes of the network can be considered as independent M/M/m queues with arrival rate ?$_i$ and service rate μ_i. According to [32 and 33] the overall mean response time is given by the following formula:

$$\overline{T} = \frac{1}{l}\left[m_i r_i + \frac{r_i}{1-r_i}\cdot P_{m_i}\right] \qquad (20)$$

Where the steady-state probability that an arriving customer has to wait in the queue of node *i* is given by:

$$P_{m_i} = \frac{(m_i r_i)^{m_i}}{m_i!(1-r_i)}\cdot p_i(0) \qquad (21)$$

and the steady-state probability of no customers at node *i* is given by:

$$p_i(0) = \left(\sum_{k_i=0}^{m_i-1}\frac{(m_i r_i)^{k_i}}{k_i!} + \frac{(m_i r_i)^{m_i}}{m_i!(1-r_i)}\right)^{-1} \qquad (22)$$

and ?$_i$ is given by the following equation:

$$l_i = l_{0i} + \sum_{j=1}^{N} l_j p_{ji}, \quad for\ i = 1, K\ , N \qquad (23)$$

In our model, each customer leaves the system after being served and hence the probability p_{ji} that it visits the station *j* after service at station *i* is zero and hence the overall arrival rate from outside to an open network is:

$$l = \sum_{i=1}^{N} l_i = \sum_{i=1}^{N} l_{0i} \qquad (24)$$

3.6.3.2 BCMP Method (A modified Jackson's Method)

The BCMP theorem takes the Jackson's idea much farther. It proves a similar result for a much larger class of queueing networks with several customer classes, different service strategies, interarrival and service time distributions and to mixed networks that contain open and closed classes. The networks considered by BCMP [7] must fulfil the following assumptions:

- The queueing disciplines FCFS, PS, LCFS-PR, IS are allowed at network nodes.
- The service times of an FCFS node must be exponentially distributed and class-independent, while PS, LCFS-PR and IS nodes can have any kind of service time distribution with a rational Laplace transform. For the latter three cases of queueing disciplines, the mean service time for different customer classes can be different.
- The service rate of an FCFS node is only allowed to depend on the number of customers at this node, whereas in a PS, LCFS-PR and IS node the service rate for a particular customer class can also depend on the number of customers of that class at the node but not on the number of customers in another class.

The BCMP theorem says that networks with the characteristics just described have product-form solution.

For an open queueing network fulfilling the assumptions of the BCMP theorem and load-independent arrival and service rates, the response time $\overline{T}_{ir}$ for an MSC of class *r* that is served by station, *i* is calculated by applying the M/M/1 formula as follows:

$$\overline{T}_{ir} = \frac{1}{l_{ir}} \cdot \frac{r_{ir}}{1 - r_i} \quad , \tag{25}$$

where l_{ir} is given by the following formula:

$$l_{i,r} = l_{0i,r} + \sum_{j=1}^{N} l_j p_{ji,r}, \quad for\ i = 1,K\ ,N\ and\ r = 1,K\ ,R \tag{26}$$

Again, as explained before, in our model each customer leaves the system after being served and hence the probability $p_{ji,r}$ that the customer of class visits the station *j* after served at station *i* is zero and hence:

$$l_{ir} = l_{0i,\,r} \tag{27}$$

Let $\overline{T}_r$ denotes the overall response time for MSC of class *r* that is served by all stations, and then $\overline{T}_r$ is as follows:

$$\overline{T}_r = \sum_{i=1}^{N} \overline{T}_{ir} = \sum_{i=1}^{N} \frac{1}{l_{ir}} \cdot \frac{r_{ir}}{1 - r_i} \tag{28}$$

,

$$r_i = \sum_{r=1}^{R} r_{ir} \tag{29}$$

and

$$r_{ir} = \begin{cases} l_r \dfrac{e_{ir}}{m_i}, & Type-1\,(m_i = 1) \\ l_r \dfrac{e_{ir}}{m_{ir}}, & Type-2,3,4 \end{cases} \quad , \tag{30}$$

[4]

where the mean number of visits e_{ir} of a customer of the r^{th} class at the i^{th} node of an open network can be determined from the routing probabilities as follows:

$$e_{ir} = p_{0i,r} + \sum_{j=1}^{N} \sum_{s=1}^{R} e_{js} p_{js,ir}, \quad for\ i = 1, \mathrm{K}\ , N\ and\ r = 1, \mathrm{K}\ , R \quad , \tag{31}$$

where $p_{0,ir}$ is the probability in an open network that a customer from outside the network enters the i^{th} node as a customer of the r^{th} class, but the $p_{js,ir}$ are equal to zero because after a message (customer) is served it leaves the network and hence

$$e_{ir} = p_{0,ir}, \quad for\ \ i = 1,...,N\ and\ r = 1,...,R \tag{32}$$

and

$$l_r = \sum_{i=1}^{N} l_{0,ir} \tag{33}$$

For Type-l nodes with more than one service unit (m_i > 1), the response time for an MSC of class *r* that is served by station *i*, is given by the following formula:

[4] Type-l: M/M/m-FCFS.
Type-2: M/G/l-PS.
Type-3: M/G/8 (infinite server).
Type-4: M/G/l-LCFS PR.

$$\overline{T}_{ir} = \frac{1}{l_{ir}}\left[m_i r_{ir} + \frac{r_{ir}}{1 - r_i} \cdot P_{m_i} \right] \quad , \tag{34}$$

where the steady-state probability that an arriving customer has to wait in the queue of node *i* is given by:

$$P_{m_i} = \frac{(m_i r_{ir})^{m_i}}{m_i!(1 - r_i)} \cdot p_i(0) \tag{35}$$

and the steady-state probability of no customers at node *i* is given by:

$$p_i(0) = \left(\sum_{k_i=0}^{m_i-1} \frac{(m_i r_{ir})^{k_i}}{k_i!} + \frac{(m_i r_{ir})^{m_i}}{m_i!(1 - r_i)} \right)^{-1} \tag{36}$$

3.6.3.3 Approximate Solution Methods

As we mentioned in previous sections, the open queueing networks with M/M/m queueing stations, FCFS queueing discipline, multiclass of customers with different service rates for different classes of customers are not one of the product form queueing networks and hence product form methods like Jackson's method and BCMP method cannot be used. To calculate the performance measures for such networks, we need to use approximate solutions.

In this section, we show how to deal with approximate performance analysis of these open non-product form queueing networks, based on the method of decomposition. Different methods, which are based on the method of decomposition, are due to Kühn [12 and 42], Chylla [15], Pujolle [67], Whitt [74 and 75] and Gelenbe [24].

Open networks to be analyzed by the method of decomposition must have the following properties:

- The interarrival times and service times are arbitrarily distributed and are given by the first and second moments.
- The queueing discipline is FCFS and there is no restriction on the length of the queue.

- The network can have several classes of customers.
- The nodes of the network can be of single or multiple server type.
- Class switching is not allowed.

With these prerequisites, the method works as follows:

- Calculate the arrival rates and the utilizations of each node as follows:

$$l_{i,r} = l_{0i,r} + \sum_{j=1}^{N} l_{j,r} p_{ji,r}, \quad for\ i = 1,\mathrm{K}\ ,N \tag{37}$$

and due to the reason that each customer leaves the system after being served and hence the probability $p_{ji,r}$ that the customer of class *r* visits the station *j* after served at station *i* is zero, then:

$$l_{i,r} = l_{0i,r}, \quad for\ i = 1,\mathrm{K}\ ,N \tag{38}$$

,

$$l_i = \sum_{i=1}^{N} l_{i,r} \tag{39}$$

,

$$r_{ir} = \frac{l_{i,r}}{m_i m_{ir}} \tag{40}$$

and

$$r_i = \sum_{r=1}^{R} r_{ir} \tag{41}$$

respectively.

- Calculate the mean service rate μ_i of node *i*:

$$m_i = \frac{1}{\sum_{r=1}^{R} \frac{l_{i,r}}{l_i} \cdot \frac{1}{m_i m_{i,r}}} \tag{42}$$

- Compute the coefficient of variation c_{Bi} of the service time of node *i*, as follows:

$$c_{Bi}^2 = -1 + \sum_{r=1}^{R} \frac{l_{i,r}}{l_i} \cdot \left(\frac{m_i}{m_i \cdot m_{i,r}} \right)^2 \cdot \left(c_{i,r}^2 + 1 \right) \quad , \tag{43}$$

where $c_{i,r}$ is the coefficient of variation for the service time of customers of class *r* at node *i*.

- ➢ The method is then an iterative method with the following steps:
  - ✓ Compute the coefficient of variation of the interarrival times at each node, using the following equations:

$$c_{ij,r}^2 = 1 + p_{ij,r} \cdot \left(c_{Di}^2 - 1 \right) \tag{44}$$

The initial values of $c_{ij,r}$ equal to 1. The coefficient of variation $c_{Aj,r}$ of the interarrival times at node *i* for class *r* is given by:

$$c_{Ai,r}^2 = \frac{1}{l_{i,r}} \cdot \left(\sum_{j=1}^{N} c_{ji,r}^2 \cdot l_{j,r} \cdot p_{ji,r} + c_{0i,r}^2 \cdot l_{0,r} \cdot p_{0i,r} \right) \tag{45}$$

and hence the coefficient of variation c_{Ai} of the interarrival times at node *i* is:

$$c_{Ai}^2 = \frac{1}{l_i} \cdot \sum_{r=1}^{R} c_{Ai,r}^2 \cdot l_{i,r} \tag{46}$$

- ✓ Compute the coefficient of variation of the interdeparture times at each node, using the following equations:

$$c_{Di}^2 = r_i^2 \cdot (c_{Bi}^2 + 1) + (1 - r_i) \cdot c_{Ai}^2 + r_i \cdot (1 - 2r_i) \tag{47}$$

- ➢ Compute the mean queue length for the M/M/m-FCFS:

$$\overline{Q}_{i,rM/M/m} = \frac{r_{i,r}}{1 - r_i} \cdot p_{m_i} \tag{48}$$

and hence the response time is:

$$\overline{T}_{ir} = \frac{1}{l_{ir}} \left[m_i r_{ir} + \overline{Q}_{i,rM/M/m} \right] \tag{49}$$

To get better approximations, many authors have modified the mean queue length described by equation 48 and here are some examples:

- Allen-Cunneen [1] :

$$\overline{Q}_{i,rAC} \approx \overline{Q}_{i,rM/M/m} \cdot \frac{c^2_{Ai,r} + c^2_{Bi,r}}{2} \tag{50}$$

- Kramer/Langenbach-Belz [41]:

$$\overline{Q}_{i,rKLB} \approx \overline{Q}_{i,rAC} \cdot G_{KLB} \tag{51}$$

with the correction factor:

$$G_{KLB} = \begin{cases} e^{\left(-\frac{2}{3}\cdot\frac{(1-r_i)}{r_i}\cdot\frac{(1-c^2_{Ai,r})^2}{(c^2_{Ai,r}+c^2_{Bi,r})}\right)}, & c^2_{Ai,r} \le 1 \\ e^{-\left((1-r_i)\cdot\frac{(c^2_{Ai,r}-1)}{(c^2_{Ai,r}+c^2_{Bi,r})}\right)}, & c^2_{Ai,r} > 1 \end{cases} \tag{52}$$

Note that due to the fact that we use interarrival time and service time distributions of type negative exponential, the coefficients of variation for the service time and interarrival time $c^2_{Ai,r}$ and $c^2_{Bi,r}$ are equal to 1 and hence modifications of the mean queue length in equations 50 and 51 do not improve the value of $\overline{Q}_{i,rM/M/m}$ given by equation 48. These approximations of the mean queue length $\overline{Q}_{i,rM/M/m}$ are very valuable and of course give better results in the case of using non-exponentially distributed service times.

3.7 Remarks on Applying the Queueing Network Formulas for Some Special Cases

3.7.1 Class of Messages Factor

In the previous sections, we assumed that the service rate is affected by two factors; *i*, which indicates the node, and *r* which indicates the MSC class, i.e. the service rate at different nodes for different MSCs classes are different. However, in practice, messages of different MSCs are also classified into complexity classes and hence the service rates are different according to different messages complexity classes.

To deal with this case, we should use a three indices service rate μ_{isk}, where *s* indicates the MSC class and *k* indicates the message class. Instead of using the three indices service rate μ_{isk}, we simply consider two indices service rate μ_{ir}, where *i* to point to the node and *r* to point to the class of the message where $r = s \cdot k$. According to this modification, the class index in all formulas of the previous section should have an upper-bound equals to the multiplication of the MSC classes number and the message classes number.

Let us take an example to make things more clear. In the real world example in Chapter 6, we have 14 MSCs, 3 message complexity classes, and 4 stations. Therefore, the service rate μ_{ir} is defined for 4 nodes and 42 classes.

The response time T_{ir} is then calculated for 42 classes. If we sum up T_{ir} for classes 1, 2 and 3, we will get the response time for MSC 1 at node *i* and if we do the same but for classes 4, 5 and 6, then we get the response time for MSC 2 at node *i* and so on until get the response time for each MSC.

3.7.2 The SHRINK Factor

In section 6.3.4, we have a real world mobile communication system, which has a lot of mobile users and the goal, is to evaluate some performance measures like for example the response time for some MSCs using simulation.

To do this, we need to gather data by evaluating some thousand observations, whereas during the necessary observation period some hundred millions of events may occur. The requirement is to evaluate these measures of interest without considering the plenty of mobile users located elsewhere. The SHRINK approach is applied to reduce the effort for large-scale simulations. It has been originally propagated and applied by K. Psounis, cf. [63, 64 and 66]. This approach can be applied in the field of simulation and measurement to evaluate the performance of large communication systems, for example, the MxRAN simulator models [2, 3 and 61]. When applying the queueing network analytical formulas to evaluate some performance measures some changes in the parameters should be done, to understand these changes we should at first explain the SHRINK approach.

The SHRINK approach is based on scaling down the complete model. Let us take an M/M/1 queueing station as an example, where the arrival rate l and the service rate m are multiplied by a factor a, $0<a<1$. This model is "shrinked" or "scaled down", because it works slower, i.e. arrivals are less frequent and service durations are longer, but the performance behaviour is identical with respect to utilization and mean number of population because the utilization $r = l/m$ or equivalently $r = a\,l/am$.. The mean response times $R = E[N]/X$, where the $E[N]$ "mean number of customers" equals $r/(1-r)$ and the arrival rate $X = l$ (for the fast model) and $X = a\,l$ (for the slow model) respectively, so the response time for the fast model is a times the slow model. According to this explanation, some changes should be done before applying the queueing networks formulas discussed in the previous sections. The arrival rates and the service rates should be multiplied by the SHRINK factor a After calculating the response times $\overline{T}_{ir}$ for the slow down system, it should multiplied by the SHRINK factor a to get the corresponding values for the fast model (original model) and then summed up to get the overall response time $\overline{T}_r$.

3.7.3 The Branching Factor

In some MSCs of the real world mobile communication system example in Chapter 6, the following situation happens, two messages are sent at the same time which leads to two branches in the same MSC but due to the fact that the last message of each MSC is predefined, there is only one path which leads to this last message and of course the response time for this MSC is calculated only for that path although messages of the other branch is served.

When we calculate the response times for different MSCs using the analytical queueing network formulas, a correction must be done, otherwise the calculated response time will be different from response times obtained from the simulator. The correction is based on subtracting the summation of the service times of messages that do not belong to the predefined branch from the calculated the response times $\overline{T}_{ir}$. The service times are calculated by multiplying the number of messages of a certain class, which does not belong to the predefined branch by its service time.

3.8 End-to-end Delay of HMSCs

3.8.1 Strong Sequencing Composition HMSC

In the previous sections, we assumed that T_r denotes the overall response time for MSC of class *r* that is served by all stations. In this section, we will derive a formula for calculating the *end-to-end delay E2E*. We can define the end-to-end delay to be the time between the arrival of the first message in the first MSC of the HMSC and the departure of the last message in the last MSC of that HMSC, i.e. the time needed to run a complete HMSC. In case that we have n MSCs that are strongly sequenced and that the calculated response times are $T_r, r = 1,2,\ldots,N$, then the average end-to-end delay E2E giving the duration of one execution of all MSCs is calculated by:

$$\overline{T}_r = \sum_{i=1}^{N} \overline{T}_{ir} \qquad (53)$$

Equation 53 gives a formula to calculate the end-to-end delay, which can be applied in the case that two or more MSCs are vertically composed (see Figure 4, section 2.2.3).

3.8.2 Other Types of Composition

Now, we will discuss the end-to-end delay for different possible cases of HMSC composition. Note that the response times T_i for MSC M_i do account for service times and wait times of all messages; since there may be additional delays due to the type of HMSC composition, the formulas given by equations 54 and 55 can not generally written as equations (instead of relations). Note that in the formula for strong sequencing composition, we assumed that there is no time gap between the MSCs.

- **Selection:** In the case of selection composition, consider that we have n MSCs (M_1, M_2,…, M_n). After all events of MSC M_1 are completed then one MSC among *n-1* choices (M_2, M_3, …, M_n) may be selected to start. If we assume that P_{12}, P_{13}, …, P_{1n} are the routing probabilities for different choices M_2, M_3, …, M_n then the end to end delay will take the form:

$$E2E \leq \overline{T}_1 + \sum_{i=2}^{n} P_{1i}\overline{T}_i \quad , \tag{54}$$

where T_1, T_2, …, T_n are the response times for M_1, M_2, …, M_n respectively.

- **Parallel Composition and Weak Sequencing:** In both cases, let T_1, T_2 be the response times for *M1* and *M2* then the end to end delay will take the form:

$$E2E \leq \overline{T}_1 + \overline{T}_2 \tag{55}$$

- **Loop:** The example in Figure 7 shows a loop for two MSCs denoted *M1* and *M2* which are vertically composed. Let T_1, T_2 be the response times for *M1* and *M2* respectively. Due to the fact that the loop is the vertical composition of the last node with the first node, then the end-to-end delay in that case equals the end-to-end delay in case that there is no loop multiplied by the repetition number, i.e., it equals to:

$$E2E = n \cdot (\overline{T}_1 + \overline{T}_2) \quad , \tag{56}$$

where *n* is the repetition number.

Chapter 4

Simulation of MSC-based Models

4.1 Introduction

Simulation is an alternative way to obtain performance measures of a system. Additionally to stationary measures, also transient measures can be determined. Using a discrete event simulation IDE (like JavaDEMOS, the one that we use, see Appendix A) the state of all objects, e.g. resources and entities, can be inspected at any time. For example, the values of the state variables of a certain resource, in particular the maximum queue length, the current queue length, the average queue length and the average wait time can be observed dynamically at any time during the simulation run. In addition, simulation can be used to evaluate models which do not satisfy the conditions necessary for analytical evaluations. In Sections 4.2, an introduction to simulation modelling including simulation definition, the purposes of using simulation and a survey of different simulation types is presented. In section 4.3, we concentrate on discrete event simulation and give a survey to its types and its structural components. The rest of this chapter describes the simulation model used to evaluate the performance of systems under study.

4.2 Simulation Modelling Overview

4.2.1 What is Simulation?

A simulation is the imitation of the operation of a real-world process or system over time.

Whether done by hand or on a computer, simulation involves the generation of an artificial history of a system, and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system [4].

The behaviour of a system as it evolves over time is studied by developing a simulation model. This model usually takes the form of a set of assumptions concerning the operation of the system. These assumptions are expressed in mathematical, logical, and symbolic relationships between the entities, or objects of interest, of the system. Once developed and validated, a model can be used to investigate a wide variety of "what if" questions about the real world system [5]. Potential changes to the system can first be simulated in order to predict their impact on system performance.

Simulation can also be used to study systems in the design stage, before such systems are built. Thus, simulation modelling can be used both as an analysis tool for predicting the effect of changes to existing systems, and as a design tool to predict the performance of new systems under varying sets of circumstances [5].

In some instances, a model can be developed which is simple enough to be "solved" by mathematical methods. Such solutions may be found by the use of differential calculus, probability theory, algebraic methods, or other mathematical techniques. The solution usually consists of one or more numerical parameters, which are called measures of performance of the system. However, many real-world systems are so complex that models of these systems are impossible to be solved mathematically. In these instances, numerical, computer-based simulation can be used to imitate the behaviour of the system over time. From the simulation, data are collected as if a real system was being observed. This simulation-generated data is used to estimate the measures of performance of the system [5].

4.2.2 Why Using Simulation

Simulation is a useful technique for computer and communication systems performance analysis. If the system to be characterized is not available, as is often the case during the design or procurement stage, a simulation model provides an easy way to predict the performance or compare several alternatives. Further, even if a system is available for measurement, a simulation model may be preferred over measurements because it allows the alternatives to be compared under a wider variety of workloads and environments [34]. While queueing theory is a powerful technique, many systems cannot be analyzed so easily. This can be for a number of reasons [65]:

- **Complicated distributions**: Most of the straightforward queueing results hold only for a limited number of distributions.
- **Complicated dynamics**: Analytical queueing network analysis has many stringent requirements in order to be applied.
- **More complicated queueing rules**: There are a number of aspects of queues that are not handled by the models. For example, what is the advantage of processing the fastest jobs first? What if there, are multiple lines leading to the servers? What if customers, renege after, say, 10 minutes waiting? These aspects add another level of complexity to already complicated systems.
- **Transient versus long-term behaviour**: The queueing results discuss long-term behaviour. How do systems act in the short term? Does it take a long time to reach steady state behaviour?

To handle these and other issues, we need another approach for modelling system behaviour. One approach to this is simulation. Simulation will easily be able to handle the issues above. This comes at a cost, however, that may make it inappropriate for some situations [65].

4.2.3 Types of Simulation Models

A model is defined as a representation of a system for the purpose of studying the system [4]. There are many ways to classify simulation models [4, 30, 34 and 65], as follows (see Figure 12):

- The first is whether the model has a stochastic (or random) aspect or not. According to this point simulation models are classified as follows:
  - ✓ *Deterministic simulations:* Deterministic simulations are completely defined by the model. A system is simulated under well-determined conditions. This kind of

simulation is useful to observe the behaviour of system in certain particular cases, to discover errors in the design or in the implementations, to build examples, etc. In this kind of simulations, only one run is needed and there is no truly random variable involved. To see the behaviour of the system we need to "trace" the output on a file and later to see and analyze it in a textual or in a graphical form.

- ✓ *Stochastic simulations:* Stochastic simulations include randomness. In a statistical simulation, we measure the system performance. This is useful to see if the system has good response time under average conditions, to compare different implementations of the same system, or very different systems that have the same output. Multiple runs of the same model may generate different values. This random element forces us to generate many outcomes to see the range of possibilities.

- ➢ Another aspect that is of interest is the difference between static and dynamic models:
  - ✓ *Static model:* In a static model, time plays no essential role. Most of these models are called *Monte Carlo* models. Many portfolio selection models in finance are Monte Carlo models. Given a portfolio, with different probabilistic (and correlated payouts), it is possible to generate a possible yield.
  - ✓ *Dynamic model:* Such a model might become a dynamic model if it incorporates changes in the portfolio over time, or if the model of payoff must be simulated over time. An example of a dynamic model is the analysis of a bank queue as it evolves over time.

- ➢ A third aspect that is important is that of *discrete* versus *continuous* models.
  - ✓ *Discrete event simulation*: A simulation using a discrete state model of the system is called a discrete event simulation.
  - ✓ *Continuous event simulation*: In continuous event simulations, the state of the system takes continuous values. The continuous state models are, e.g. used in chemical simulations where the state of the system is described by the concentration of a chemical substance.

A major subclass of simulation problems is concerned with the simulation of time varying systems, which are controlled, by a combination of either physical, chemical, biological or man made laws. Such systems can be categorized by the way, in which time is treated in the simulation. The variation in time may be considered continuous in some systems whilst in others the state of the system changes at discrete time intervals. This phenomenon gives rise to two branches of simulation: Discrete Event Simulation and Continuous Simulation. Clearly, models of such systems must also be capable of changing their state in a similar manner.

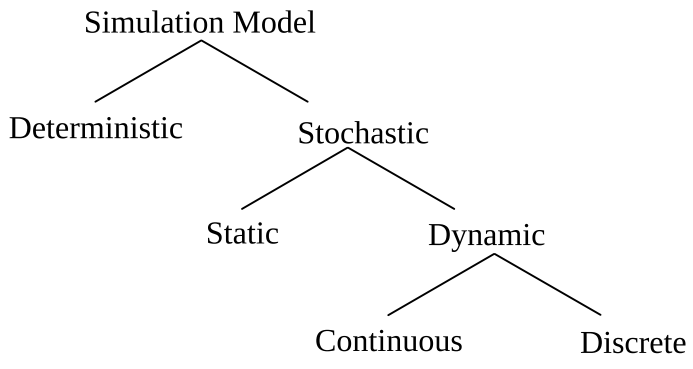


Figure 12: A classification of simulation methodology

4.3 Discrete Event Simulation

Discrete simulation examines problems in which the ordering and timing of events is the focus of interest. In such systems, the interest is on the time at which some activity commences or ceases. For example, in simulating a computer network to estimate the effective system capacity or queue sizes, we may be interested in the start time and duration of job processing rather than details of the signal transmission on the network. In such problems, it is not efficient to advance time in small fixed steps but to advance to the time of the next event. Since, in general, events can occur at any time, the time advance is non-uniform and can be alternately large or small [39].

4.3.1 Types of Discrete Event Simulation

Discrete simulation can be further subdivided in terms of the methodology followed as follows:

- **Event scheduling**: Event scheduling is the first way simulations were developed. An *event* is anything that changes the system statistics (also known as the *state* of the system) other than the mere passage of time. The essential idea of event scheduling is to move along the time scale until an event occurs, then, depending on the event, modify the system state, and possibly schedule new events. We will see in the next section how to generate the random times needed in order to be able to generate such things as the service times and the arrival times. Based on this, it is a trivial exercise to run through a simulation of the system. The events are stored in an *event queue*, which lists all events in order. The first event in the queue is taken off, and other events may then be added (assuming that an event only triggers other events in the future).
- **Process-oriented**: In the process-oriented approach, the simulation programmer composes a set of process descriptions. Each process description serves as a model of one kind of active entity in the simulated system. An active instance of a process description is called a process. In a simulation system, there is a process management facility, which allows processes to become active, to operate in the simulated environment and to eventually terminate.

4.3.2 Discrete Event Simulation Structure

Although there are various flavours and paradigms in discrete event simulation, there has evolved a basic structure that is used by most simulation packages. Regardless of how complex a discrete-event simulation package may be, it is likely to contain the basic components that we will describe in this section.

The structural components of a discrete-event simulation include entities, activities and events, resources, a random number generator, a calendar, system state variables and statistics collectors.

4.3.2.1 Entities

The best way to understand the function of an entity is to understand that *entities cause changes in the state of the simulation*. Without entities, nothing would happen in a simulation. In fact, one stopping condition for a simulation model is the condition where there are no active entities in the system. Entities have *attributes*. Attributes are characteristics of a given entity that are unique to that entity. Attributes are critical to the understanding of the performance and function of entities in the simulation.

4.3.2.2 Activities and Events

Activities are processes and logic in the simulation. *Events* are associated with conditions that occur at a point in time which cause a change in the state of the system. An entity *interacts* with activities. Entities *interacting with* activities *create* events.

There are three major types of activities in a simulation: *delays*, *queues and logic*. The *delay* activity is when the entity is delayed for a definite period. At the point that the entity starts the delay, an event occurs. This event schedules the entity on the *calendar* (which we will get to later). If the delay is for *d* time units, then the entity is scheduled to complete the delay *d* time units after the current time of the simulation. At that time, the delay expires and another event is generated. *Queues* are places in the simulation were entities wait for an unspecified period. Entities can be waiting on *resources* (which we will get to later) to be available or for a given system condition to occur. *Queues* are most commonly used for waiting in line for a resource or storing material that will be taken out of the queue when the right conditions exist. *Logic* activities simply allow the entity to effect the state of the system through the manipulation of state variables (which we will get to later) or decision logic.

4.3.2.3 Resources

In a simulation, resources represent anything that has a restricted (or constrained) capacity. Common examples of resources include workers, machines, nodes in a communication network, traffic intersections, etc. It should also be noted that very complex resources could be utilized in a simulation.

4.3.2.4 Random Number Generator

Every simulation package has a random number generator. The random number generator (technically called a pseudo-random number generator) is a software routine that generates a random number between zero and one that is used in sampling random distributions. Everything that is random in the simulation uses the random number generator as an input to determine values.

4.3.2.5 The Calendar

The *calendar* for the simulation is a list of events that are scheduled to occur in the future. In every simulation, there is only one calendar of future events and it is ordered by the earliest scheduled time first. In a later example, it will become more clear how the calendar works and why it is important in the simulation. At this point, just remember that, at any given point in time, every event that has already been scheduled to occur in the future is held on the calendar.

4.3.2.6 System State Variables

Depending on the simulation package, there can be several system state variables, but the one system state variable that every simulation package has is the current time of the simulation. The current time variable is updated every time an entity is taken from the calendar.

4.3.2.7 Statistics Collectors

Statistics collectors are a part of the simulation that collects statistics on certain states (such as the state of a resources), or the value of global variables, or certain performance statistics based on attributes of the entity. There are three different types of statistics that are collected, *counts, time-persistent,* and *tallies.* Counts, are very straightforward, they *count values of variables*. *Time-persistent* statistical collectors give the time-weighted values of different variables in the simulation. A common variable to track is the utilization of a resource. *Tally* statistical collectors take a sequence of real-valued observations and calculate the mean and standard derivation of the collected observations without regard to the amount of time between observations.

4.4 Simulation Model

4.4.1 MSC-Based System Description

As discussed before, communication between system components can be described by MSCs or HMSCs. The MSC basic components used to describe the system behaviour are instances and messages. Each message sequence chart describes message exchange between instances. To calculate the performance measures of such systems, these MSCs are extended by time consumptions and afterwards mapped to a queueing network model, which can be solved either by queueing network analysis or by simulation.

In the queueing network model, instances are considered as queueing stations, which have one or more servers. Messages are considered as customers that visit different queueing stations to get service. When a message visits an instance, it is served if the server is empty or it waits for service in the queue of this instance and served later when the server is free. The service time of each message is calculated according to the speed of the server and the service amount assigned to this message. The service time is defined as the service amount divided by the server speed.

4.4.2 Modelling of System Components

To build a simulation model of such systems, we should define how to model the following system components:

- Instances
- Messages

As we discussed in the previous section, each instance of the system is considered as a queueing station in the queueing network model. The queueing station may have one or more servers and one queue. When a message (a customer) arrives at a certain instance (a queueing station), it is served if the server is idle and the queue is empty, otherwise it waits in the queue of this queueing station and when the server is idle, it is scheduled according to a certain scheduling discipline.

According to the structure and behaviour of queueing stations, we can model them as resources in the simulation. We used a discrete event simulation package called JavaDEMOS to develop simulation (for details, see Appendix A). In JavaDEMOS, according to resource synchronizations, resources are classified into two categories, Res (for the mutual exclusion synchronization) and Bin (for the producer/consumer synchronization).

In our simulated systems, instances (resources) are shared, but they must only be accessed by one message at a time, one has a mutual exclusion situation. In these situations, resources are requested and released by the same message. In addition, when a message requests an unavailable resource, it must wait (it is blocked). Messages are modelled in simulation as entities. Each message does the following actions:

- Request the resource.
- Delay, for certain period.
- Release the resource.

After that, the message schedules the next message and then it is destroyed. Each message spends some delay at the resource. This delay is for service process and is presented by a negative exponential distribution (any other kind of stochastic distributions is also possible). The mean service time is calculated according to the resource speed and the service amount assigned to this message. The service time is defined as the service amount divided by the server speed. Note that, other types of service distribution like Erlang or hyperexponential distributions can also be used.

4.4.3 Scheduling Instances of the MSC

To impose a relevant load onto the system, the execution rate or the interarrival, i.e. the number of instantiations per time unit to each MSC should be specified. The interarrival times for MSCs of the system may have the same values or different values, i.e. if we describe a system by a set of MSCs, which are composed by a certain composition operator to form an HMSC, each MSC may have or may have not a different mean interarrival time.

Each MSC in such system is initiated by a certain traffic source, which generates only that type of MSC according to a certain predefined mean interarrival time. This will be discussed in details in the following sections.

4.4.4 How Are Messages Scheduled?

Let us consider that the system under study is described by more than one MSC. In this case, these MSCs are composed either vertically or in parallel or using any other composition operator. It is important to know the way the MSCs are composed because this will affect the average end-to-end delay of the whole system. To schedule messages of such systems, we should distinguish between:

- Initiating a new MSC
- Starting a new MSC
- Generating the next message of the MSC

Each MSC has a traffic source that initiates instances of this MSC. For example if we have four MSCs, then we have also four traffic sources which are responsible of generating instances of the corresponding MSC according to a certain interarrival time. To initiate a new MSC, the traffic source corresponding of this MSC sends a message to the instance that will send the first message of the MSC. The message sent by the traffic source, is referred to as message zero. It is not a real message of the MSC so it does not need any resource time for service. It is just a green light to the instance to start the MSC and send the first message. Receiving the message zero is considered as an order to the receiving instance to start the MSC and then the first message of the MSC is sent.

After scheduling the first message in the MSC, it starts its actions. First, it requests the resource, then spends some time delay for service and after that it releases the resource. After that, it generates the next message (the second message) and after that, it is destroyed. The rest of messages of the MSC are scheduled in the same way until reaching the last message of the MSC.

4.4.5 Simulation Input

As we discussed before MSC-based description of the system under study depends on defining a set of MSCs that describes the communication between system components. These MSCs are extended by some parameters, like the complexity class annotations and resources speeds to define the mean service times at different resources for different messages of different complexity classes.

The set of MSCs together with the complexity class annotations is called the **Load Model.** The load model shows the sender and receiver of each message, the name of the message and the associated complexity class. Resources and their processing properties like resources speeds and service amounts for different complexity classes and hence the calculated mean service times are called the **Resource Model**.

To impose a relevant load onto the system a **Traffic Model** is specified. The Traffic Model contains the execution rates, i.e. the number of instantiations per time unit for each MSC. To provide these data as an input to the simulator we used a suitable representation as text using Excel sheets. We can summarize the input for the simulator as three different excel sheets, as follows:

- *Traffic Model*: The traffic model contains the number of instantiations per time unit for each MSC.
- *Load model*: The load model defines name, sender, receiver and complexity of each message.
- *Resource Model*: In this sheet the following is specified:
  - ✓ CPU speeds
  - ✓ Service amounts
  - ✓ Service times

In the following section we will show the simulator package classes and how they work together to calculate the performance measure of the system under study.

4.4.6 Building Blocks of the System

In this section, we will discuss the simulation main classes and their functions in the simulation model. The graph in Figure 13 describes the main classes of the simulation model and the relation between them.

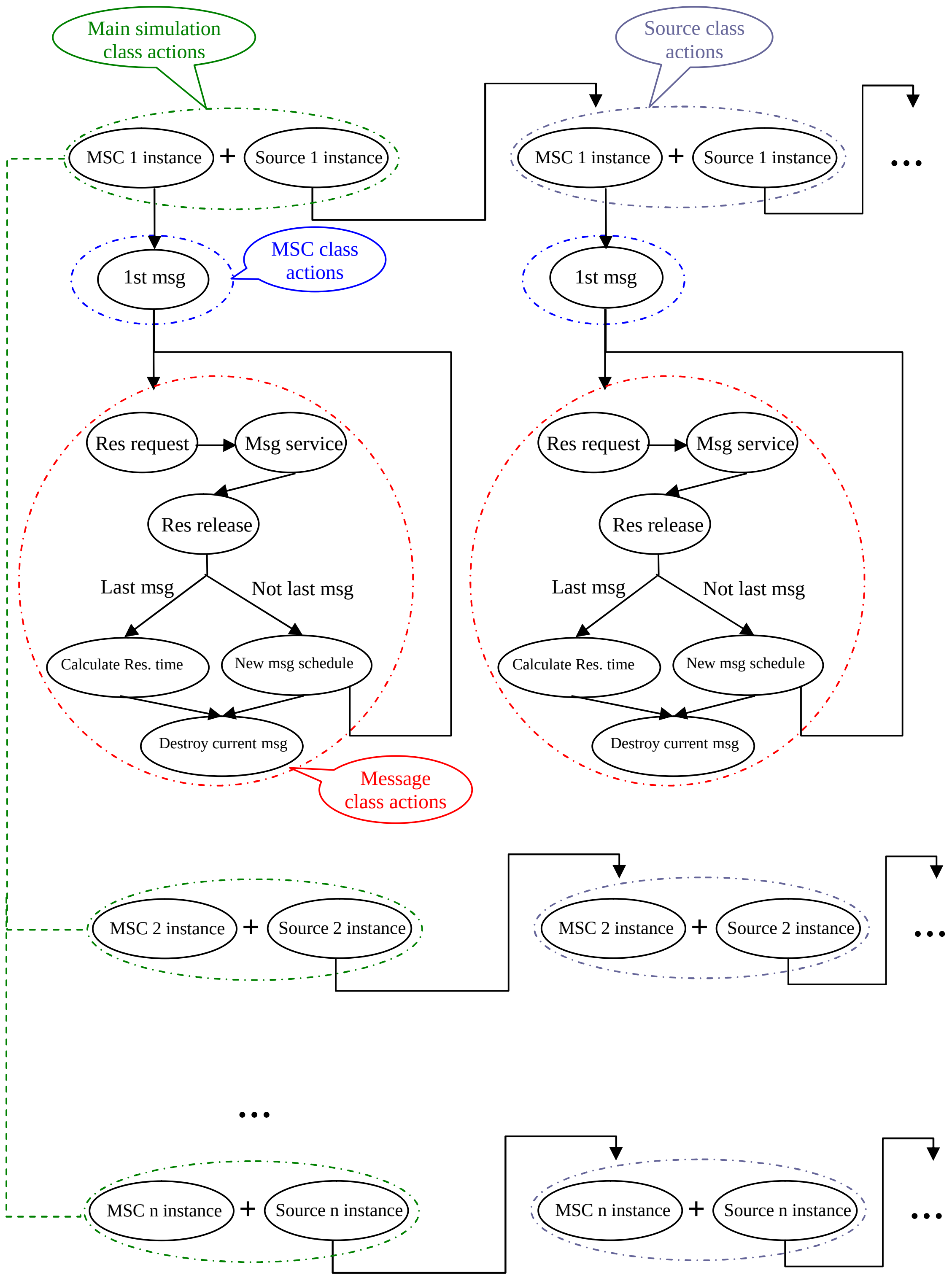


Figure 13: Simulation model structure

In the following, we will classify these classes according to their functions into four groups:

- Input reading.
- Resources.
- Generating messages.
- The simulation manager.

4.4.6.1 Input Reading

Class MSCdata is responsible of reading resources related data like resources speeds and service amounts for different complexity classes. It also reads the set of MSCs, which describes the system under study. It reads the name, sender, receiver and complexity of each message. Before starting the actual simulation tasks, the class "MSCdata" is invoked from within the main simulation class to save these data into arrays, to be available during the simulation to schedule messages of different MSCs.

4.4.6.2 Resources

Class CPU defines the behaviour of the resources of systems under study. This class extends the JavaDEMOS class Res. Class Res defines a certain kind of resources in which resources are shared and cannot be accessed by more than one process at a time. Objects of type CPU have the same characteristics like Res objects, they are requested and released by the message objects. If a message, requesting an unavailable resource then it must wait (is blocked) otherwise it is served by the CPU for a certain service time depending on the CPU speed and the service amount assigned to this message which depends on the complexity class that it belongs to.

4.4.6.3 Generating Messages

Messages are grouped into MSCs. Each MSC implements a certain task and all MSCs cooperate to form the system behaviour. As we discussed before each MSC is initiated by an external message sent from the traffic source of this MSC. Then the MSC schedules the first message. After that, this first message implements its actions and schedules the next message until the last message of this MSC is reached.

To model this behaviour we implemented three classes that schedule messages. These classes are:

- Class Source
- Class MSC
- Class Message

Class Source is responsible of initiating MSCs. It models the behaviour of the traffic source. For example, if the system consists of five MSCs then we have five corresponding “Source” objects. Each one sends a message to the corresponding MSC to start scheduling its first message. Class Source also schedules the next source instance according to the predefined interarrival time distribution (see Figure 13).

After receiving the message of the traffic source, a new instance of the MSC is scheduled. This new instance schedules the first message of the MSC. The class MSC is responsible of modelling the behaviour of the MSCs (see Figure 13). To distinguish different MSCs, class MSC has an argument called index to keep the MSC number value. The same is done with the class Source.

The scheduled message has some arguments, e.g. the index of the MSC that contains this message and the message scheduling time. These arguments help calculating the time needed to finish each message of this MSC. This message time is inherited to the next message of the same MSC and is added to the time of this new message, so when the last message of the MSC is ended this time will be the response time of this MSC. The scheduled message has information about the node to which it will be sent. Class Message models this behaviour (see Figure 13). It requests the CPU of the node that receives this message. Then some delay is spent for serving this message. This time is calculated according to a certain service time distribution. After being served, the message time is calculated and then added to the inherited time of previous messages of the same MSC. Then the current message schedules the next message, according to the load model, which is saved into arrays by the “MSCdata” class, and inherits the time to the new message. Finally, the current message is destroyed.

The same scenario is implemented by each message until it reaches the last message of the MSC. At this point, the response time of this MSC is modified and when the simulation ends, we obtain the average response times of different MSCs.

4.4.6.4 Main Simulation Class

The main class that manages the simulation is a java class, which extends the JavaDEMOS class "Entity". This class implements the following:

- It declares and initializes the simulation time attribute (parameter).
- It declares the type of arrival rates distribution and initializes the mean arrival rate attribute for each MSC.
- It also declares the response time attribute for each MSC, which is calculated using the batch means method.
- It also defines the batch size and confidence level that will be used.
- After the declaration of the attributes, the scheduling process of the traffic sources start.
- After that, the corresponding MSCs start.
- When the simulation time is over, we obtain the simulation results.

4.5 Simulation Output

In this section, we will discuss the simulation results for steady state measures. The JavaDEMOS package provides the user with a complete report about the resources used in the simulation, data collectors like Accumulate and Tally class objects, distributions, batch means objects.

After the end of the simulation, we get a report about the resources (CPUs). This report contains information for each resource. The information includes name of the resource, start time, maximum queue length of the queue of the resource, average queue length, the limit of the resource, which determines how many messages can be served at the same time, and the utilization of the resource.

Here, we measure the average response times for different MSCs. To do this we generate confidence intervals using the batch means method. The batch means method and the confidence interval calculations are implemented automatically by JavaDEMOS package by the two classes “BatchMeans” and “ConfidenceInterval” respectively. We declare objects of type BatchMeans for each MSC in the system. We use a confidence level of value 0.9. After the end of the last message of each MSC instance, the batch mean object is updated and finally at the end of the simulation a report is obtained. This report contains the average response time, minimum and maximum response time, the estimated standard deviation and the confidence interval for the end-to-end delay of each MSC. In addition, another report that contains data about number of batches and batch size used to determine the average response time of each MSC is introduced.

Another report on the distributions used in the simulation is also introduced. This report contains names of distributions, type of distribution, start time, the mean value for distributions of type negative exponential, the seed value and the next sample value. We used two distributions one for the MSC interarrival time and the other is for service time process.

We can also make use of the so-called Traces. Trace can help us to see gradually how the simulation proceeds. It shows the model times, the entities and their actions.

Chapter 5

Case Studies

5.1 Case Study One: Client-Server Systems

Client/server (C/S) systems are composed of client processes that submit requests to one or more server processes. Servers passively await for client requests and may enlist other servers in order to reply to a request originating from a client. These processes, clients and servers, are usually organized in multi-tiered software architecture. Usually, clients and servers execute on different machines connected by networks. A brief introduction to client-server systems is given in Appendix B.

5.1.1 Communication-Processing Delay Diagrams

The communication-processing delay diagrams describe a request live cycle, which is generated by a client in a C/S system. Communication-processing delay diagrams are graphic notation to illustrate how requests spend their time at each resource including clients, servers, LAN segments and WANs [45].

A communication-processing delay diagram (see Figure 14) is a sequence of parallel time axes drawn vertically with time increasing from top to bottom. There are two types of time axes:

- Communication time axes (dashed lines), corresponding to time spent in LAN segments and WANs.

- Processing time axes (solid lines), corresponding to time spent processing elements such as client and server processors, client and server storage devices, and routers.

Figure 14 shows time axes for a client, a server, and a LAN segment in a two-tier C/S configuration. Diagonal arrows in a delay diagram indicate requests going from clients to servers and vice versa. These arrows cross-dashed lines associated with the networks traversed by the request. Requests and replies are labelled by a pair of the form *[id, m]* where *id* identifies the request and its reply and *m* indicates the average size in bytes of the message carrying the request or reply. For example, request *r* in Figure 14 is m_1 bytes long and its reply is m_2 bytes long. The network transmission time in seconds is equal to the message size in bits divided by the network bandwidth *B* in bits per second (bps) [45].

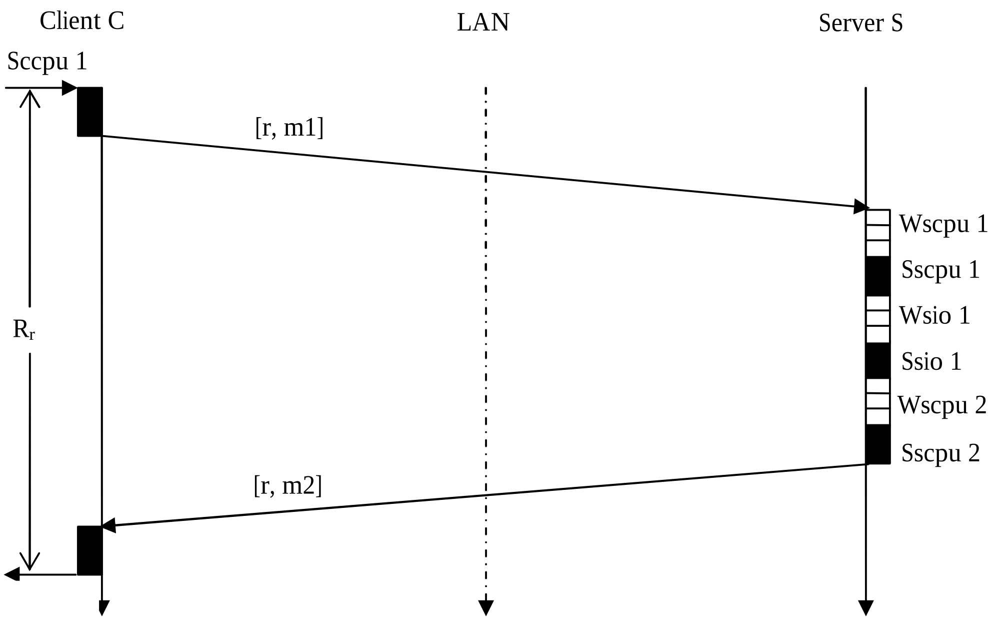


Figure 14: Example for communication in a two-tier architecture (See [45])

5.1.1.1 A Two-tier Architecture

In a two-tier architecture, a client talks directly to a server, with no intervening server. It is typically used in small environments (less than 50 users). A common error in client/server development is to prototype an application in a small two-tier environment and then scale up by simply adding more users to the server. This approach will usually result in an ineffective system, as the server becomes overwhelmed. To properly scale to hundreds or thousands of users, it is usually necessary to move to a three-tier architecture [45].

5.1.1.2 A Three-tier Architecture

A three-tier architecture introduces a server (or an "agent") between the client and the server. The role of the agent is manifold. It can provide translation services (as in adapting a legacy application on a mainframe to a client/server environment), metering services (as in acting as a transaction monitor to limit the number of simultaneous requests to a given server), or intelligent agent services (as in mapping a request to a number of different servers, collating the results, and returning a single response to the client) [45].

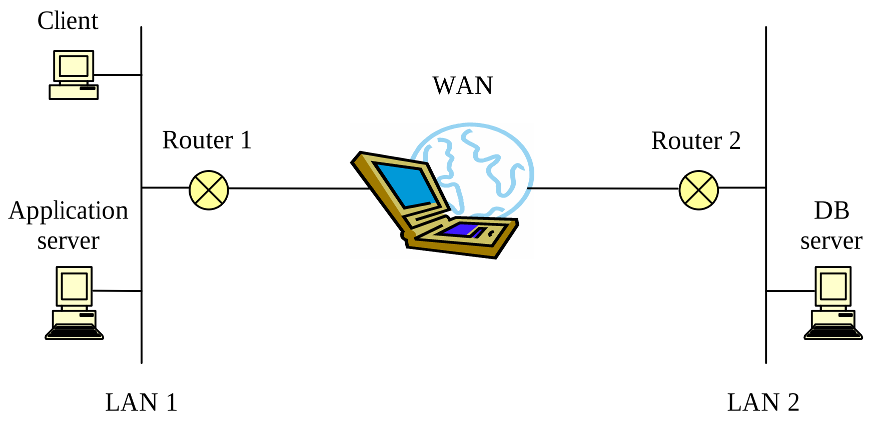


Figure 15: Three-tier architecture (See [45])

Consider now the three-tier C/S architecture depicted in Figure 15. The client sends a request to the application server located on the same LAN (LAN 1). The application logic is executed at the application server and may require several accesses to the database (DB). Each access to the DB server has to traverse LAN 1 to reach router 1, traverse the WAN and arrive, through router 2, at LAN 2-the LAN where the DB server is located.

Figure 16 shows a communications-processing delay diagram that illustrates the flow of a request in this C/S architecture. This diagram shows some instances of network waiting times, denoted as W_{net}, for LANs 1 and 2. The routers were not shown in the diagram to avoid cluttering [45].

The following section will show how we can convert the communications-processing delay diagrams describing the Client/Sever systems to Message Sequence Charts. In addition, service parameters for different resources are discussed. Then, we describe the process of extending the MSC description by these parameters and then map it into a queueing network model and after that, some performance measures like the utilization of the resources and the response time for the request are calculated using either analytical queueing network algorithms or simulation.

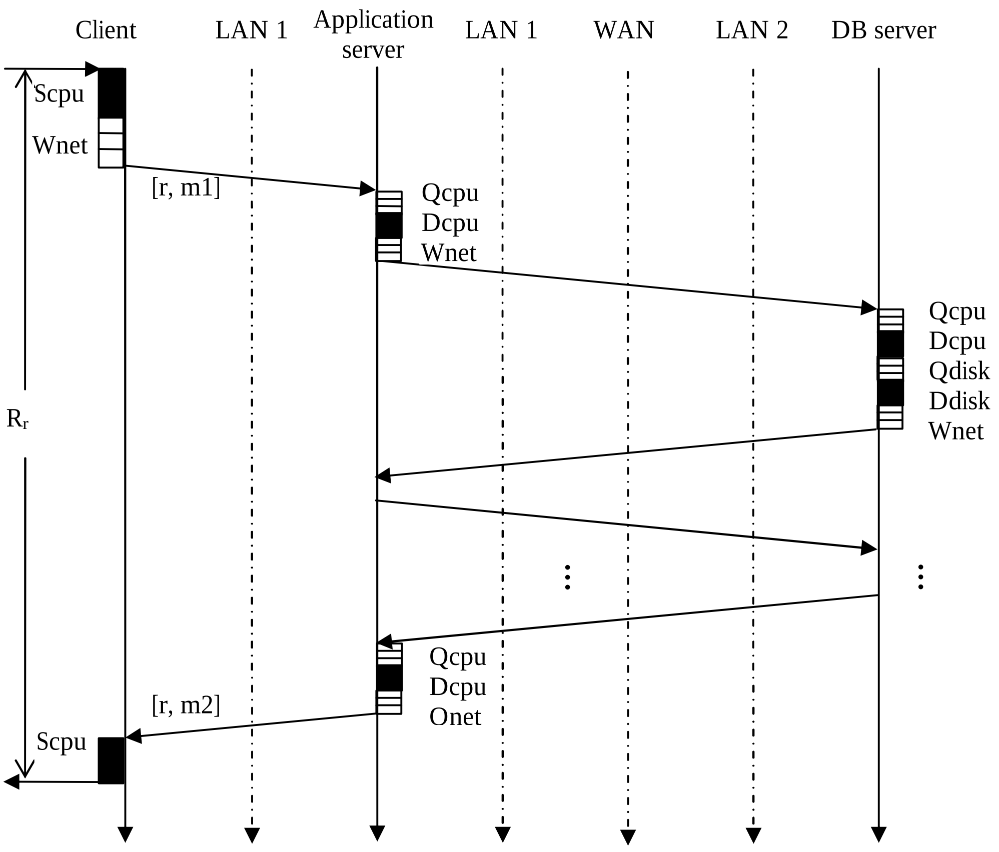


Figure 16: Three-tier C/S system (See [45])

5.1.2 The Queueing Network Model

5.1.2.1 Describing Client Requests by MSCs

In this section, we show how we can get an MSC-based description for client requests, which are described by communication-processing delay diagrams. A Message Sequence Chart, as described in section 2.2, has two major components, instances and messages.

Within the instance body the events order is specified. To completely describe the client request using the MSCs notions, we should decide accurately the following:

- Instances.
- Events at each instance.
- Message exchange order between instances.

We can consider each resource as an instance, which has its own events (sending and receiving message events). For example, consider the three-tier architecture described by the communication-processing delay diagram (Figure 16). We have, one resource CPU at the client and one resource at application sever, LAN1, WAN, LAN2. At the database server, we have two resources CPU and IO. This way we get seven instances in the MSC-based description of the system in Figure 16 (as shown in Figure 17).

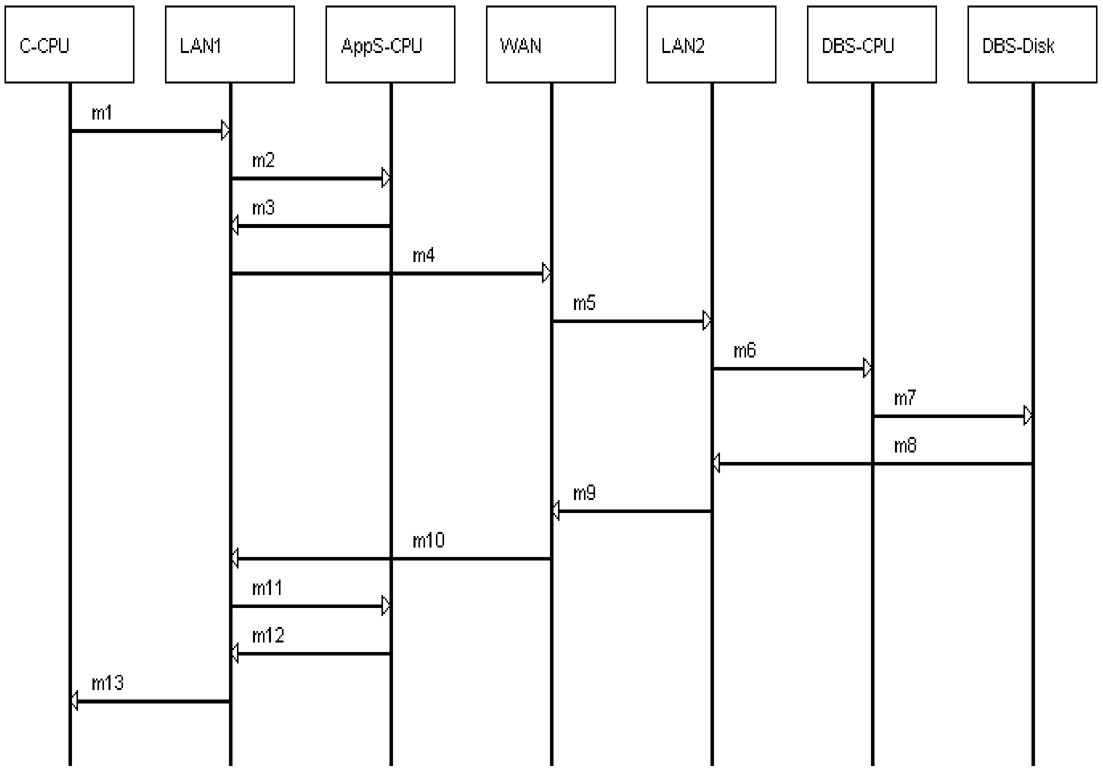


Figure 17: The MSC corresponding to the three-tier client-server system architecture

In communication-processing delay diagrams, there are two types of time axes, communication time axes (dashed lines), corresponding to time spent in LAN segments and WANs and processing time axes (solid lines), corresponding to time spent processing elements such as client and server processors, client and server storage devices, and routers. These time axes are equivalent to those of the instances in the MSC.

Diagonal arrows in a delay diagram indicate requests going from clients to servers and vice versa. As Figure 16 shows, these arrows cross the dashed lines associated with the networks traversed by the request. These diagonal arrows of the communications-processing delay diagrams are altered by messages in the MSC-based description of the system. One diagonal line is substituted by a number of messages equals the number of the dashed lines that it crosses plus one. For example, in Figure 16 the dashed line from application server to database server is substituted by four messages in the MSC. These messages are m3, m4, m5 and m6 (Figure 17).

In addition, internal messages between resources of the same server must be described. For example, in Figure 16 the database server uses the CPU and IO resources after receiving a signal from LAN2 and after that it sends another signal to LAN1. This situation is described by m6, m7 and m8 as shown in Figure 17.

5.1.2.2 Extending the MSC-based Description by Time Consumption

The next step is to extend the previous MSC-based description by time consumption. Two important parameter sets that affect the performance are:

- *Resource parameters*: Intrinsic features of a resource that affects performance. Examples include disk seek times, latency and transfer rates, network band width, router latency, and CPU speed ratings.
- *Service times:* Specify the sum of all service times for a request at a resource. Examples include the CPU time of transactions at the database server, the total transmission time of replies from the database server in LAN, and the total I/O time at the web.

Detailed descriptions about how to calculate service rates at different resources are discussed in [45].

We discussed how the C/S systems described by communication-processing delay diagram, are described by MSCs. This MSC-based description is extended with the performance parameters. The next step is to map the extended MSC-based description into a queueing network model and after that solve this queueing network model either by analytical queueing networks formulas discussed in Chapter 3 or alternatively by using simulation. This will be discussed in the following sections.

5.1.2.3 Mapping the Extended MSC-based Description into a Queueing Model

As already discussed, requests in a C/S system, are served by several types of resources (e.g. processors, disks, networks, and routers). Each time a request visits a resource, it may need to queue for the use of the resource. The various queues that represent a distributed C/S system are interconnected, giving rise to a network of queues, called a queuing network (QN). Figure 18 shows the queuing network corresponding to the three-tier C/S system shown in Figure 16.

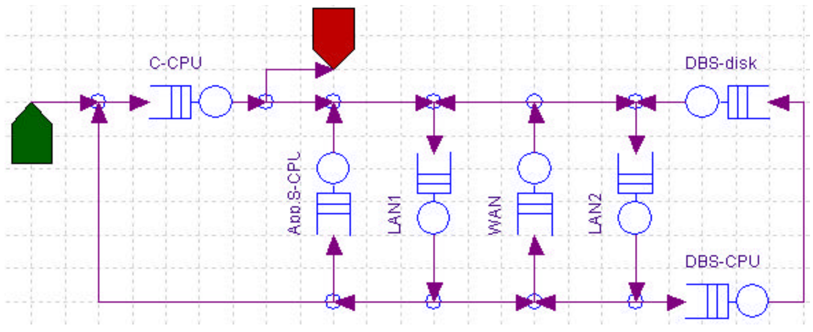


Figure 18: The three-tier C/S system as a QN (Drawn with WinPEPSY-QNS [8 and 78])

Performance prediction is the process of estimating performance measures of a computer system for a given set of parameters. Typical performance measures include response time, throughput and resource utilization. Performance prediction requires the use of models. Two types of models based a unique queueing model (see Figure 18), simulation models and analytical models may be used. Both types of models have to consider contention for resources and the queues that arise at each system resource.

5.1.3 Input Description

One important question is how the simulator as well as the analytical queueing networks formulas uses the extended MSC description. The answer is that instead of using the graphical presentation for the MSCs, textual tables are used. For each resource, we make a table. This table describes messages received by this resource. Therefore, we have a set of tables, each represent messages sent by each resource. Each line of the table keeps data about which resource sent the message and to which one it will send. The data saved in the table is enough to keep track of the next message of the MSC, in the same order as the original MSC.

The performance parameters are also saved in other tables. For example, the service times for each request at different resources are saved in another table. Using these tables together with the arrival rates parameters for different request kinds, one can get some performance measures like utilization, throughput and response time either by using analytical formulas or by running the simulator.

Resource Name:	**The corresponding message table**					
Client	**Source Node:**	**Source FE:**	**Source ID:**	**Next Node:**	**Next FE:**	**Next ID:**
	-1	-1	0	1	1	1
	1	1	13	-1	-1	-1
LAN1	**Source Node:**	**Source FE:**	**Source ID:**	**Next Node:**	**Next FE:**	**Next ID:**
	0	1	1	2	1	2
	2	1	3	3	1	4
	3	1	10	2	1	11
	2	1	12	0	1	13
Application Server	**Source Node:**	**Source FE:**	**Source ID:**	**Next Node:**	**Next FE:**	**Next ID:**
	1	1	2	1	1	3
	1	1	11	1	1	12
WAN	**Source Node:**	**Source FE:**	**Source ID:**	**Next Node:**	**Next FE:**	**Next ID:**
	1	1	4	4	1	5
	4	1	9	1	1	10
LAN2	**Source Node:**	**Source FE:**	**Source ID:**	**Next Node:**	**Next FE:**	**Next ID:**
	3	1	5	5	1	6
	6	1	8	3	1	9
DB Server CPU	**Source Node:**	**Source FE:**	**Source ID:**	**Next Node:**	**Next FE:**	**Next ID:**
	4	1	6	6	1	7
DB Server Disk	**Source Node:**	**Source FE:**	**Source ID:**	**Next Node:**	**Next FE:**	**Next ID:**
	5	1	7	4	1	8

Table 1: Messages table

Table 1 and Table 2 show the MSC description in a text form that can be read by both the simulator and by the analytical QN algorithms. In Table 1, we can recognize seven different tables corresponding to the seven resources used in the tree-tier C/S system. Table 2 shows service times at different resources.

Each table, as we mentioned before describes the messages received by the resource. Each line inside the table describes three related parts for a message, the first one, is the source part, which describes the sender of this message and its ID. The second describes which resource will receive the next message that will be sent by this resource. Instead of writing the full names of the resource, we write only an ID number. In this example, the resources' IDs are ranged from one to seven.

Resources	**Resource Service time (sec)**	**Resource Service rate (requests / sec)**
Client-CPU	0.25	4
LAN1	0.01	100
Application sever-CPU	0.14	7
WAN	0.1	10
LAN2	0.01	100
Database server-CPU	0.25	4
Database server-IO	0.25	4

Table 2: Service times at different resources

In the following the two performance models will be introduced, the analytical queueing networks model as well as the simulation model.

5.1.4 Analytical Queueing Networks Algorithms and Simulation Results

Using the formulas discussed in Chapter 3, we get the utilization of each resource and the response times at different resources. Alternatively, identical results are obtained using simulation. In our example, the simulation runs for 1000 seconds using the JavaDEMOS simulator (see Appendix A). The arrival rate used is 3 requests / second.

Table 3 summarizes the results for both simulation and analytical queueing networks analysis. The results of interest are the utilization of each resource and the mean response time of the client request at different resources. Table 3 shows a 90% confidence interval for the mean response time of the MSC presenting the client request in the three-tier C/S architecture at different resources as well as the total end-to-end delay. From Table 3 it can be seen that results from analytical and simulation models are approximately identical. As seen in Table 3 most of response time results of the analytical queueing network lie in the confidence interval.

Resources	**Simulation Results**			**Analytical QN Results**		
	% utilization	**Response Time (sec) ± (90% Con. Int.)**	**Request end-to-end delay(sec) ± (90% Con. Int.)**	**% utilization**	**Response Time (sec)**	**Request end-to-end delay(sec)**
Client-CPU	75.464 %	1.193±0.279	5.593±0.162	75.0 %	1.0	5.56
LAN1	12.158 %	0.046±0.01		12.0 %	0.045	
Application sever-CPU	85.434 %	1.733±0.209		85.71 %	1.999	
WAN	59.922 %	0.509±0.042		60.0 %	0.499	
LAN2	6.01 %	0.021±0.0		6.0 %	0.021	
Database server-CPU	77.212 %	1.174±0.19		75.0 %	1.0	
Database server-IO	75.646 %	0.931±0.124		75.0 %	1.0	

Table 3: Three-tier C/S example: Simulation vs. analytical QN results

One important thing is that the CPU time needed to get the results of the queueing networks algorithm is some seconds (about 5 seconds as shown in Table 4) whereas the CPU time needed for the simulation to get the results is approximately an hour.

CPU time (min : sec)	
Simulation	**QN Algorithm**
60:00	00:05

Table 4: Three-tier C/S example: CPU time for JavaDEMOS vs. QN algorithm

Evaluating these results using queueing networks algorithms in some seconds has a great advantage that is system developers can investigate, in early design stages, which amount of traffic can be carried by the planned configuration. Such analytical results show the scope of possible parameter settings and allow a better planning of simulation scenarios which include more details and are closer to reality. Of course this is difficult to be done using simulation which needs more CPU time than the queueing networks algorithms to give the same results.

5.2 Case Study Two: Open Multi-class Systems

5.2.1 Single Web Server

We now turn our attention to the problem of modelling systems that have different kinds of HTTP requests with different service times at different resources. For this purpose, we choose the problem of a single web server (this example is taken from [47]) to present it as an example, to show how these kinds of systems can be modelled and can be evaluated using MINA tool. Figure 19 shows a typical environment with a single web server at the site. The web server is connected to a LAN, which is connected to a router that connects the site to the ISP and then to the Internet. Different HTTP requests to the web server are corresponding to different documents size ranges. For example, consider that the HTTP LOG[5] of the web server shows the distribution of the document sizes and the percent of the requests in each category as well as the CPU time per HTTP request (see Table 5).

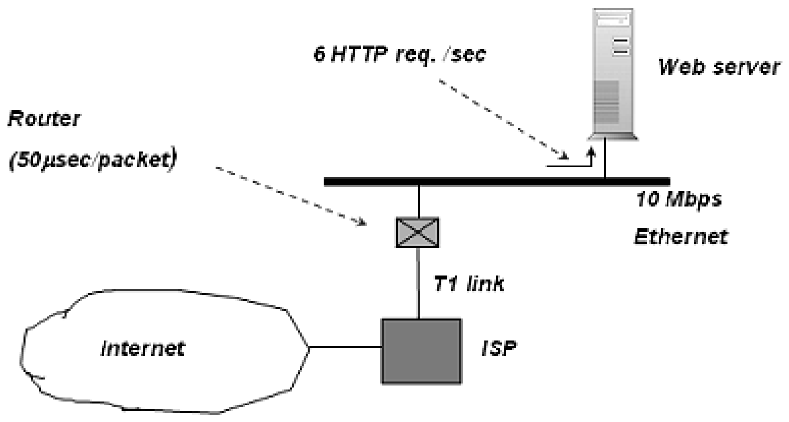


Figure 19: A single web server (see [47])

Class	Avg. File Size (KB)	% requests	CPU time per HTTP requests (sec)
1	5.0	35	0.00645
2	10.0	50	0.00816
3	38.5	14	0.01955
4	350.0	1	0.14262

Table 5: File size distributions

[5] HTTP LOG records information about every access to a web Server

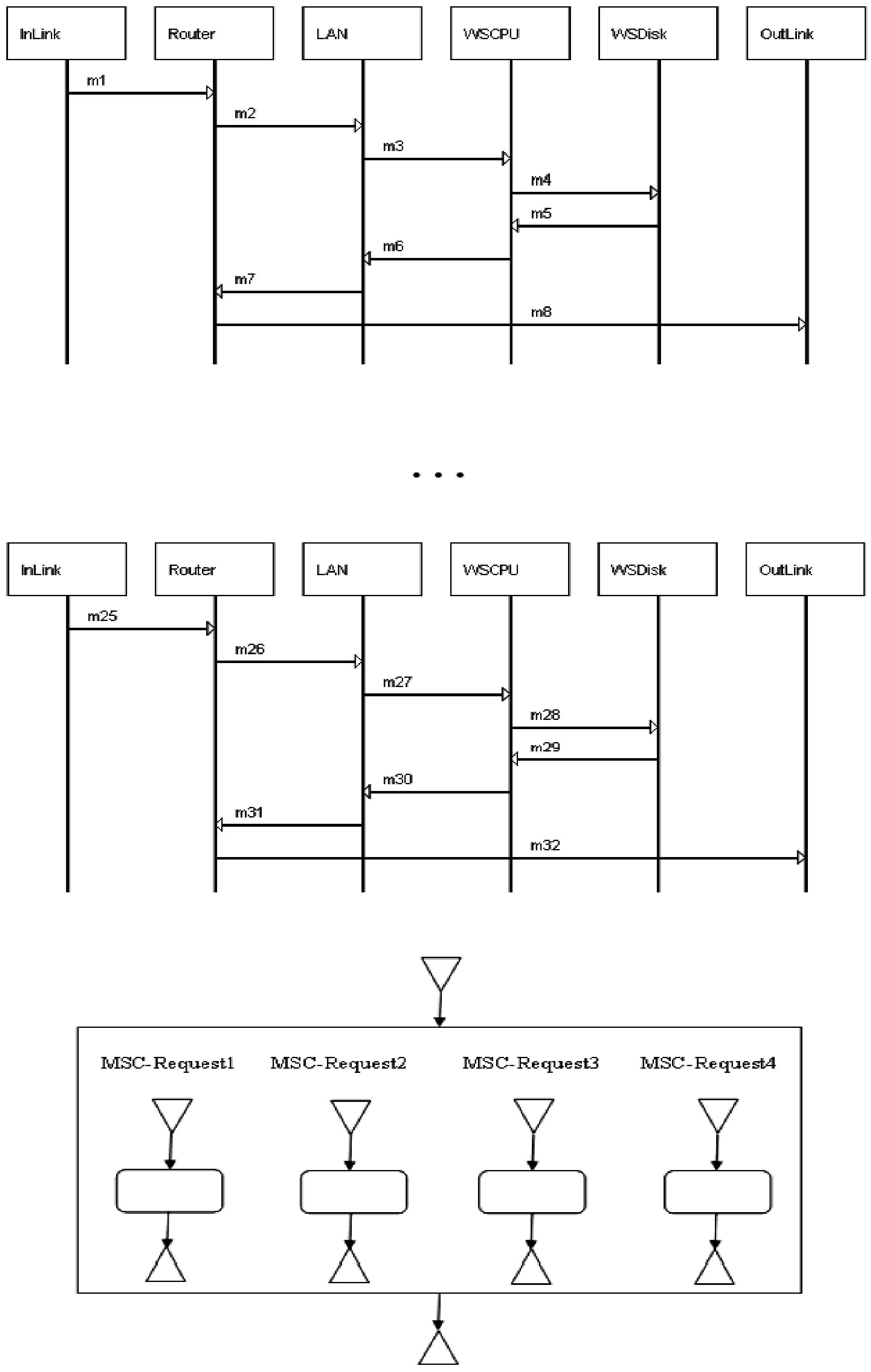


Figure 20: HMSC for the requests of the web server system

5.2.2 Describing the System by HMSC

According to the data in Table 5 we can say that the web server CPU receives four different kinds of HTTP requests, 35% of class 1, 50% of class 2, 14% of class 3 and 1% of class 4. To describe the four different HTTP requests in the single web server, each class of HTTP request is described by an MSC and these MSCs corresponding to the different HTTP requests are composed in a parallel composition manner to get an HMSC that describes the behaviour of the different HTTP requests (see Figure 20).

Each MSC presents one of the four different request classes. Each MSC describes the communication between the six instances corresponding to the incoming link, outgoing link, LAN, router, web server CPU and web server disk resources. The communication between the instances is described by eight messages.

The different instances exchange these eight messages in a certain order as shown in Figure 20. The incoming link sends the request to the router, which sends it to the LAN, and the LAN sends the request to the web server CPU, then the web server CPU sends it to the web server disk. After that, the web server disk sends the answer back to the web server CPU, the web server CPU sends it back to the LAN, the LAN sends it to the router, and the router sends it to the outgoing link.

5.2.3 Extending the System MSC-based Description by Performance Parameters

As we mentioned before the message flow between system components together with the performance parameters, like the service times at each resource and the arrival rates for different classes, are saved in tables. These tables are used as input to both the analytical queueing network formulas and the simulator to calculate some performance measures like utilization of different resources and response times for the four MSCs representing the four HTTP request classes. Table 6 describes the Service Demand[6] for different classes at each system component.

[6] The service demand at a queue is defined as the product of the average number of visits made by a request to the queue, multiplied by the average service time per visit.

Components	1	2	3	4
LAN	0.0044	0.0085	0.0325	0.2942
Router	0.0006	0.0007	0.0017	0.0124
Outgoing link	0.0269	0.0535	0.2055	1.8679
Incoming link	0.0016	0.0016	0.0016	0.0016
Web server CPU	0.0064	0.0082	0.0196	0.1426
Web server Disk	0.0300	0.0600	0.2310	2.1000

Table 6: Service demands (see [47])

To get the service time for different classes at each system component, the average number of visits made by a request at each system component for different classes must be defined.

Components	1	2	3	4
LAN	1	1	1	1
Router	2	2	2	2
Outgoing link	2	2	2	2
Incoming link	2	2	2	2
Web server CPU	1	1	1	1
Web server Disk	1	1	1	1

Table 7: The visit count at each resource component for the four classes

Table 7 shows the visits count at each system component for different classes. Table 8 shows the arrival rate for each class.

Class	1	2	3	4
Arrival rate (request/sec)	2.1	3.0	0.84	0.06

Table 8: The arrival rate for the four MSC classes

In the following, we will show the queueing network model corresponding to extended HMSC description of the web server HTTP requests.

5.2.4 Queueing Network Model for a Single Web Server

The extended MSC description is mapped into a queuing network model (see Figure 21). We are assuming here that we are dealing with a web server that is publicly available on the Internet. Thus, there is a very large population of unknown size of clients that will access the web server. Thus, we can only characterize the arrival rate of requests for various document sizes. Therefore, we will model the web server as an open multi-class QN model.

The incoming, outgoing links and the LAN are represented by load-independent queues. The router is represented by a delay queue. The web server is represented by two load-independent queues: one is for the CPU and the other one is for the disk.

5.2.5 The Analytical Queueing Network and the Simulation Results

In this section, we will introduce the results for the single web server HTTP requests example. These results ate typically the response time for each HTTP request and the utilization of different resources. To do this, we are using either analytical queueing networks techniques or alternatively simulation based on the queueing network model described in Figure 21.

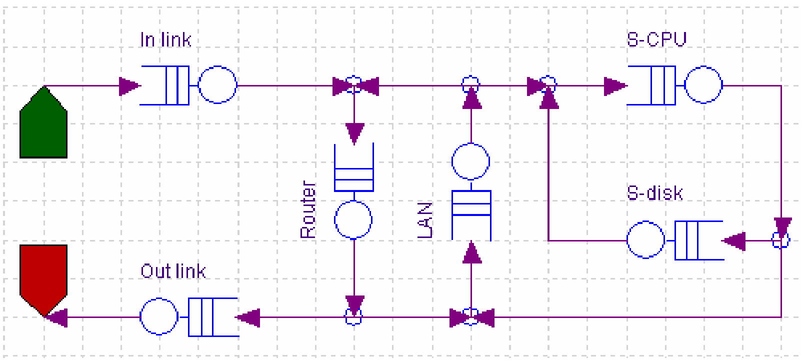


Figure 21: QN model for a single web server (Drawn with WinPEPSY-QNS [8 and 78])

The arrival rates used are described in Table 8 and are measured in requests/sec. The service demands are shown in Table 6 and are measured in seconds. The parameters used for arrival rates and service rates are used in both the queueing network model and the simulation model. In this example, the simulation runs for 28 hours (the model time) using the JavaDEMOS simulator. Running the simulator, we can get the utilization and the request response times for different system components. Using the formulas discussed in Chapter 3, we get the utilization of each resource and the response times of the four MSC classes.

Table 9 summarizes the results of the utilization of different resources. Table 9 shows the results for the simulation as well as the analytical queueing networks algorithms and compares it with the results of the example (in the textbook, see [47]). Table 10 summarizes results of the response times of the four HTTP requests. These results are for both simulation and analytical queueing networks algorithms and are compared with the results of the example (in the textbook, see [47]).

System Component	% utilization		
	Simulation	QN Analysis	Example
LAN	7.955 %	7.9 %	8.0 %
Router	0.446 %	0.5 %	0.5 %
Outgoing link	50.438 %	50.1 %	50.2 %
Incoming link	0.902 %	0.9 %	0.9 %
Web server CPU	6.33 %	6.3 %	6.3 %
Web server Disk	56.052 %	56.3 %	56.3 %

Table 9: Utilization for simulation, queueing network analysis and the example result

If we have a look on these results, we find that the utilization (see Table 9) for both the textbook results and the queueing networks algorithm results are identical for all resources because they use the same exact analytical algorithm. For the same reason the response, time results for the four HTTP requests obtained by the textbook and the queueing networks algorithm is identical (see Table 10). In addition, the JavaDEMOS simulation results are almost identical with the queueing networks algorithm results.

In Table 10, it can also be noticed that, there is an extreme difference in the response time between the JavaDEMOS simulation results and the queueing networks algorithm results at some resources, e.g. the outgoing link and the web server disk recourses whereas the results are identical at other resources, e.g. the incoming link resource. At the rest of the resources, the response time results are not identical but also the difference is not large.

In the next paragraphs, we will explain why we obtained such results. At the incoming link resource, the response time results obtained from the JavaDEMOS simulation and the queueing networks algorithm are identical because the service demands used for the four HTTP requests are identical and hence the service times are identical and in this case the queueing network algorithm used produces exact results for the response time. Hence, these results and the results obtained from the JavaDEMOS simulator are identical. At the outgoing link and the web server recourses, the response time results obtained from the JavaDEMOS simulation and the queueing networks algorithm are extremely different. The reason is that service demands used for the four HTTP requests are different and hence the service times are different and in this case, the queueing network algorithm used produces an approximate result for the response time. The difference between these results and the results obtained from the JavaDEMOS simulator are very large when the difference in the service demands for the four HTTP requests are very large. When the difference in the service demand is not very large the difference in the response time results are also not very large, e.g. at the resources web server CPU, LAN and router.

System Component	Response time											
	HTTP request 1			HTTP request 2			HTTP request 3			HTTP request 4		
	Sim.	QN	Ex.	Sim.	QN	Ex.	Sim.	QN	Ex.	Sim.	QN	Ex.
LAN	0.011	0.005	0.005	0.015	0.009	0.009	0.039	0.035	0.035	0.303	0.32	0.32
Router	0.0	0.001	0.001	0.001	0.001	0.001	0.002	0.002	0.002	0.012	0.012	0.012
Outgoing link	0.555	0.054	0.054	0.588	0.107	0.107	0.746	0.412	0.413	2.548	3.748	3.749
Incoming link	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.001	0.002	0.002
Web server CPU	0.015	0.007	0.007	0.016	0.009	0.009	0.024	0.021	0.021	0.146	0.152	0.152
Web server Disk	1.365	0.069	0.069	1.388	0.137	0.137	1.462	0.529	0.529	2.674	4.806	4.806

Table 10: Response times for simulation, queueing network analysis and the example result

CPU time (min : sec)	
Simulation	**QN Algorithm**
300:00	00:06

Table 11: Single web server example: CPU time for JavaDEMOS vs. QN algorithm

Table 11 compares the CPU time needed to get the results of the queueing networks algorithm and simulation. It shows that the CPU time in case of the queueing network analysis is very low (about 6 seconds) compared with the CPU time in case of the simulation (approximately 5 hours). So using this queueing network analysis to decide for example what kind of resources are needed to carry the traffic under specified service levels, is easier than making a simulation runs which needs more CPU time than the queueing networks algorithms.

Here, we show the results of an experiment in which we used identical service demands for different types of HTTP requests. Table 12 show the values of the service demand for the four HTTP requests at different resources. We used the same values for the visits count at each system component for different HTTP requests classes. In addition, the same arrival rates for each class are used as shown in Table 8.

Components	HTTP request 1	HTTP request 2	HTTP request 3	HTTP request 4
LAN	0.0325	0.0325	0.0325	0.0325
Router	0.0124	0.0124	0.0124	0.0124
Outgoing link	0.1200	0.1200	0.1200	0.1200
Incoming link	0.0016	0.0016	0.0016	0.0036
Web server	0.0064	0.0064	0.0064	0.0064
Web server	0.0300	0.0300	0.0300	0.0300

Table 12: Identical service demands

System Component	% utilization	
	Simulation	QN Analysis
LAN	19.415 %	19.4 %
Router	7.383 %	7.4 %
Outgoing link	71.59 %	71.9 %
Incoming link	0.901 %	0.9 %
Web server CPU	3.801 %	3.8 %
Web server Disk	18.081 %	18.0 %

Table 13: Utilization results for simulation and queueing network analysis

The utilization results for both the JavaDEMOS simulation and the queueing networks algorithm are identical (see Table 13). In addition, the results of the response times for the four HTTP requests at different resources are nearly identical although there are some resources that have a high utilization, e.g. the outgoing link has a utilization of 71.59 %.

System Component	Response time							
	HTTP request 1		HTTP request 2		HTTP request 3		HTTP request 4	
	Sim.	QN	Sim.	QN	Sim.	QN	Sim.	QN
LAN	0.04	0.04	0.04	0.04	0.04	0.04	0.042	0.04
Router	0.013	0.013	0.014	0.013	0.013	0.013	0.012	0.013
Outgoing link	0.408	0.429	0.405	0.429	0.398	0.429	0.388	0.429
Incoming link	0.002	0.002	0.002	0.002	0.002	0.002	0.001	0.002
Web server CPU	0.007	0.007	0.007	0.007	0.007	0.007	0.008	0.007
Web server Disk	0.037	0.037	0.038	0.037	0.037	0.037	0.034	0.037

Table 14: Response time results for simulation and queueing network analysis

As seen in Table 14, the response time results of the analytical queueing networks algorithm and simulation are identical for most resources. The only exception is at the "outgoing link" resource. Although, when we calculated a 90% confidence interval for the mean response time of the four HTTP requests at different resources, we found that the response time analytical results lie in the confidence intervals obtained.

System Component	**Response time**							
	HTTP request 1		**HTTP request 2**		**HTTP request 3**		**HTTP request 4**	
	90% CI	**QN**	**90% CI**	**QN**	**90% CI**	**QN**	**90% CI**	**QN**
Outgoing link	[0.377, 0.439]	0.429	[0.376, 0.434]	0.429	[0.363, 0.433]	0.429	[0.33, 0.446]	0.429

Table 15: QN results and CI of the response time at the "Outgoing link"

Table 15 shows the 90% confidence interval for the mean response time of the four HTTP requests at the "outgoing link" resource. We can easily note that all analytical response time values lie in the corresponding confidence intervals.

From the previous discussion we can conclude that using methods of open queueing networks to get the response time for MSCs of different classes is not efficient. Another technique that allows every open queueing network to be replaced by a suitably constructed closed network and then any method for closed queueing networks, e.g. the MVA (Mean Value Analysis) method, can be used [13].

The principle of the closing method is quite simple; the external world of the open network is replaced by a $-/G/1$ node with the following characteristics:

- The service rate of the new node is equal to the arrival rate of the open network and in the case of multiple class networks, the service rate of the new node is given by $R \cdot l_{0r}$, with $r = 1, ..., R$, where R is the number of classes.

- The coefficient of variation, of service time at the new node is equal to the coefficient of variation of the interarrival time of the open network.

- If the routing behaviour of the open network is specified by visit ratios, then the visit ratio of the new node is equal to 1. Otherwise the routing probabilities are assigned so that the external world is directly replaced by the new node.

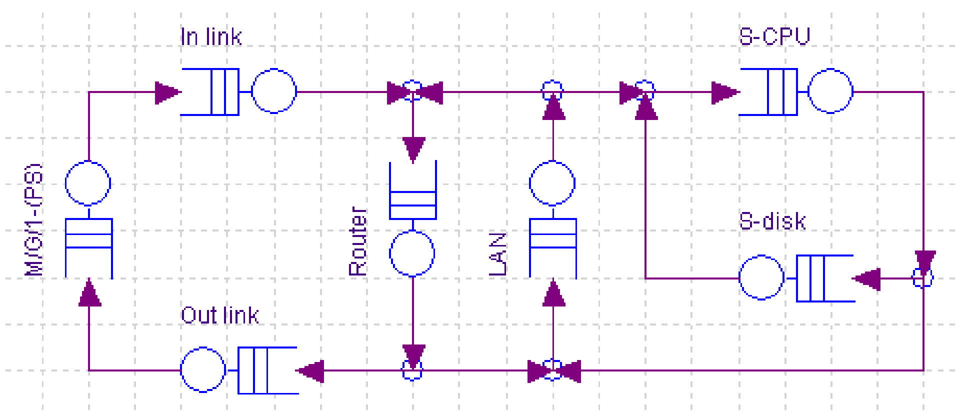


Figure 22: A closed QN for a single web server with the additional -/G/1 node for the closing method (Drawn with WinPEPSY-QNS [8 and 78])

The idea behind this technique is shown in Figure 21 and Figure 22. A very high utilization of the new node is necessary to reproduce the behaviour of the open network with adequate accuracy. This utilization is achieved when there is a large number of customers *K* in the closed network.

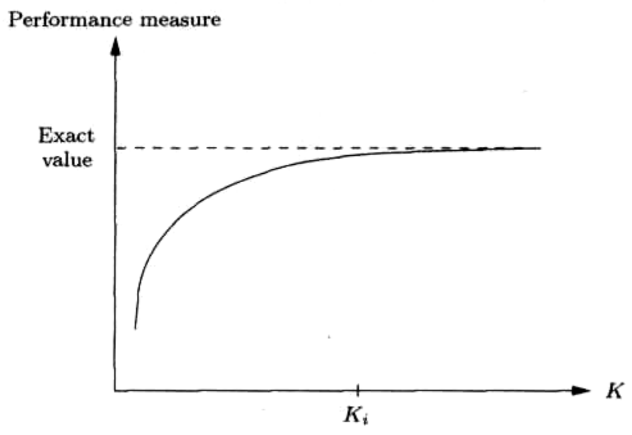


Figure 23: Threshold in the closing method

Note that, the performance measures, e.g. the response times, are sufficiently accurate after the number of customers in the network has passed a certain threshold value K_i (see Figure 23). In the case of using the MVA method the proposed value for the number of customers is 100 customers (see [13]). This leads to the following situation, when we applied this technique for this case study to calculate the response times for the four HTTP requests; we found that the CPU time needed to calculate the results is too long compared with the CPU time needed by the simulation. Hence, in the case of a large open queueing networks with more than one class of customers, it is better to use simulation than using the closing method, which has a big disadvantage that it needs a long CPU time to calculate the performance results.

Chapter 6

A large Application Example

6.1 Description of the Example

6.1.1 Introduction

The main goal of this chapter is to show how to evaluate the performance of complex communication systems using both simulation and queueing network models.

The presentation follows closely the work published in the context of the IPonAir project on architectures of future mobile communication systems (cf. [31]). The IPonAIR/MxRAN[7] project aims at a flexible radio access architecture that supports multi-band, multi-standard radio systems integration and the usage of existing and future IP-based protocols. A part of this project is the development of a discrete event simulation system, which is to study the performance behaviour of different system designs.

In [22, 55, 71 and 76] it is proposed to develop a simulation environment to analyze alternative network architectures and protocol stacks with respect to signalling performance. The authors describe a use case approach to construct a general event driven signalling protocol performance model. To this end Message, Sequence Charts (MSCs) are employed as an input of use cases to a performance simulation tool.

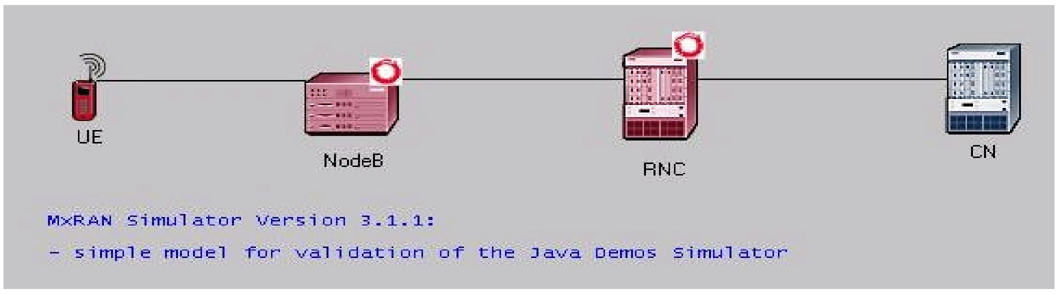
In the next sections, we will describe the model concept, which is the basis of both the

[7] MxRAN stands for Multi-band, Multi-standard Radio Access Network

MxRAN simulator as well as JavaDEMOS simulator. Then we present an implementation to derive the simulation model based on the information contained in the MSC description. After that, a comparison between the MxRAN simulator results and the MINA tool simulation results will be introduced using different models.

6.1.2 Modelling Concept

The physical network architecture of the system under study is shown in Figure 24. Figure 24 shows that this system consists of four components (nodes), the UE (User Equipment), Node B (Base Station), RNC (Radio Network controller) and CN (Core Network) which are considered with respect to signalling traffic.



MxRAN Simulator Version 3.1.1:
- simple model for validation of the Java Demos Simulator

Figure 24: Physical network architecture (see [22 and 55])

The UEs communicate with the corresponding Node B via a radio access link. The Node B has a high bandwidth (ATM) connection to the RNC, which in turn interfaces to the core network. All mobile terminals in one radio cell are aggregated in one UE cluster modelling their overall behaviour.

The signalling traffic is defined by a set of activities named "system functions" (SFs) which generate the dominant signalling load (e.g. MOBILE ORIGINATED CALL SETUP and RELEASE, etc.) in the system.

The relevant protocol entities involved in these activities are "Functional Entities" (FEs). The Functional Entities (FEs) lie inside the network nodes and they are responsible of exchanging SFs, which are represented by a specific sequence of signalling messages. Figure 25 shows the functional entities in the network nodes these sequences of messages can readily be described in the form of HMSCs.

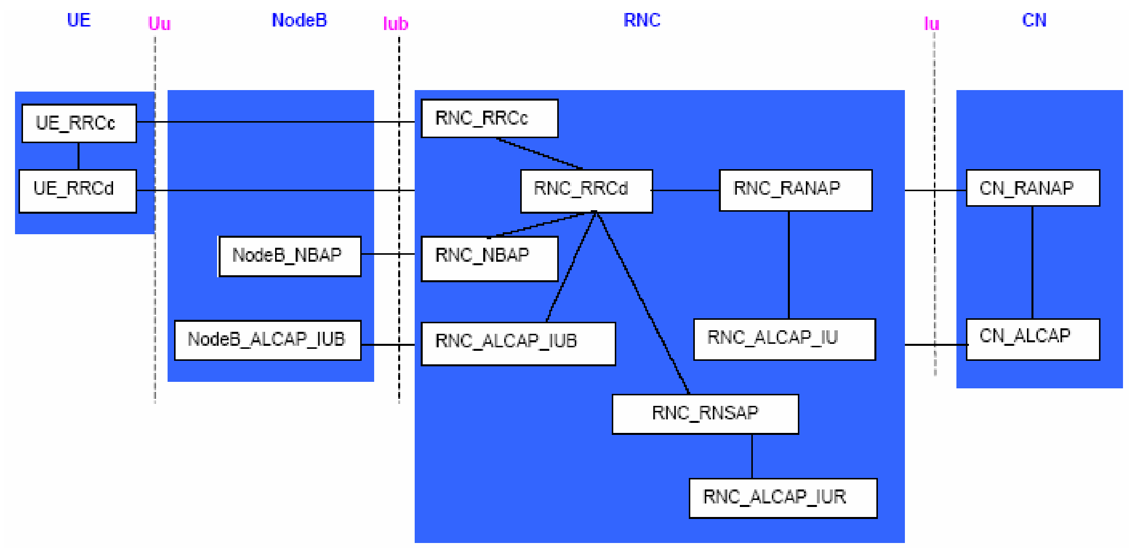
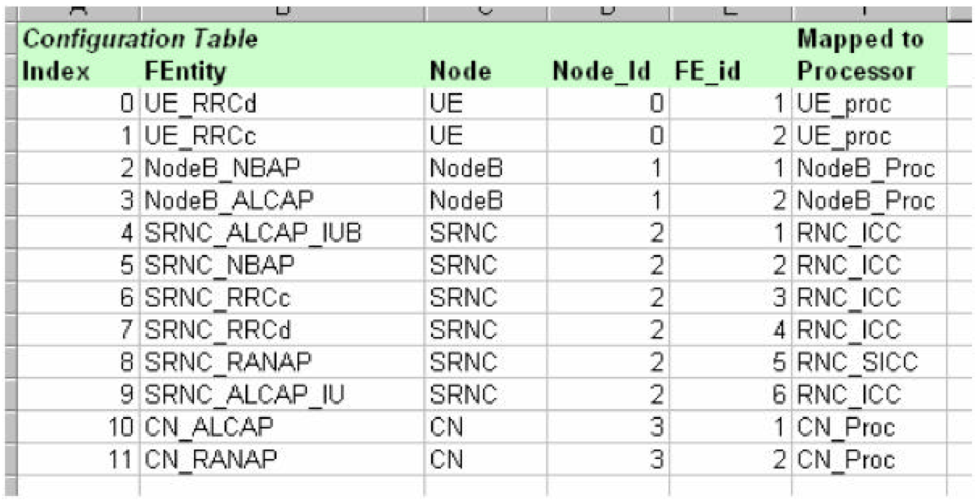


Figure 25: Functional entities in the network nodes (see [55 and 71])

Each FE is mapped to a certain resource, which serves messages sent by the corresponding FE. The mapping of the FEs onto the resources is called the **System Configuration Model**. Therefore, the System Configuration Model defines which FE runs on which processor. Here we map all FEs of the same node to one resource. Figure 26 shows the System Configuration Model.



Configuration Table					Mapped to
Index	FEntity	Node	Node_Id	FE_id	Processor
0	UE_RRCd	UE	0	1	UE_proc
1	UE_RRCc	UE	0	2	UE_proc
2	NodeB_NBAP	NodeB	1	1	NodeB_Proc
3	NodeB_ALCAP	NodeB	1	2	NodeB_Proc
4	SRNC_ALCAP_IUB	SRNC	2	1	RNC_ICC
5	SRNC_NBAP	SRNC	2	2	RNC_ICC
6	SRNC_RRCc	SRNC	2	3	RNC_ICC
7	SRNC_RRCd	SRNC	2	4	RNC_ICC
8	SRNC_RANAP	SRNC	2	5	RNC_SICC
9	SRNC_ALCAP_IU	SRNC	2	6	RNC_ICC
10	CN_ALCAP	CN	3	1	CN_Proc
11	CN_RANAP	CN	3	2	CN_Proc

Figure 26: System configuration model (see [71])

The main signalling flows of the system under study are described by a set of MSCs. Each message of each MSC is annotated with a certain complexity class. From this, we derive the processing time needed on a processor.

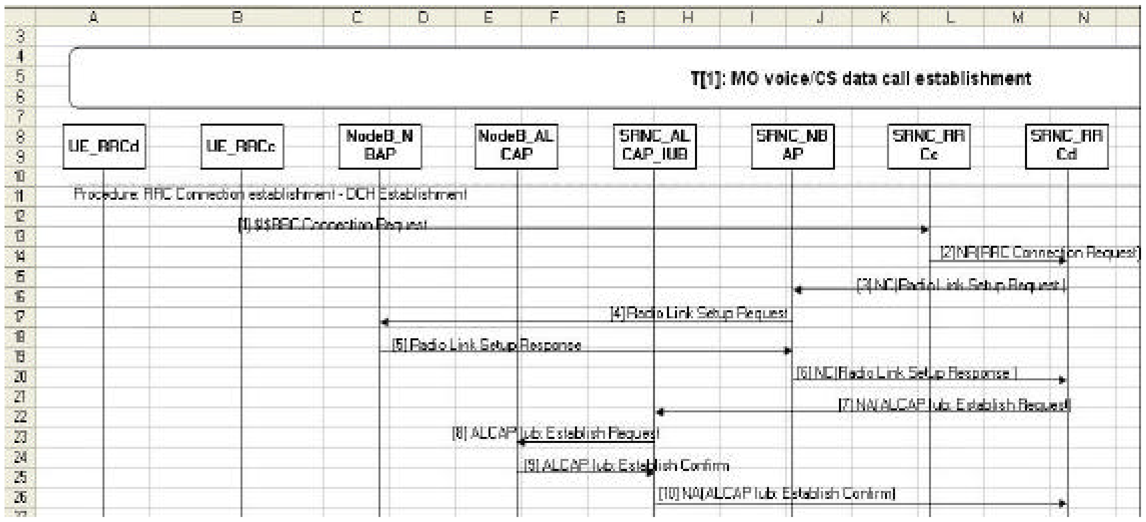
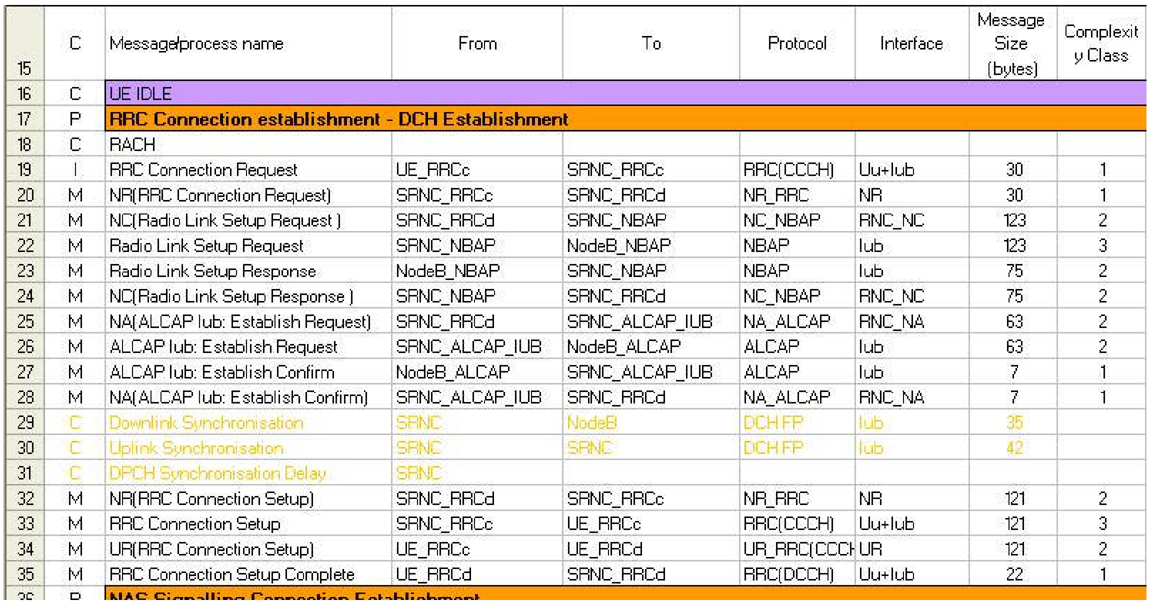

T[1]: MO voice/CS data call establishment
UE_RRCd
UE_RRCc
NodeB_NBAP
NodeB_ALCAP
SRNC_ALCAP_IUB
SRNC_NBAP
SRNC_RRCc
SRNC_RRCd
Procedure: RRC Connection establishment - DCH Establishment
[1] $$RRC Connection Request
[2] NR[RRC Connection Request]
[3] NC[Radio Link Setup Request]
[4] Radio Link Setup Request
[5] Radio Link Setup Response
[6] NC[Radio Link Setup Response]
[7] NA[ALCAP Iub: Establish Request]
[8] ALCAP Iub: Establish Request
[9] ALCAP Iub: Establish Confirm
[10] NA[ALCAP Iub: Establish Confirm]

Figure 27: Graphical representation of the load model (see [71])

The set of MSCs together with the complexity class annotations is called the **Load Model**. The graphical representation of the load model is shown in Figure 27 and the corresponding load model is shown in Figure 28 as an Excel sheet.



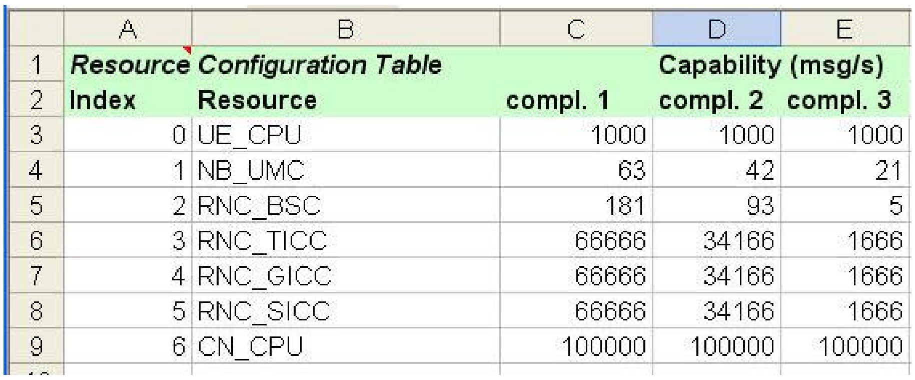
15	C	Message/process name	From	To	Protocol	Interface	Message Size (bytes)	Complexity Class
16	C	UE IDLE						
17	P	**RRC Connection establishment - DCH Establishment**						
18	C	RACH						
19	I	RRC Connection Request	UE_RRCc	SRNC_RRCc	RRC(CCCH)	Uu+Iub	30	1
20	M	NR(RRC Connection Request)	SRNC_RRCc	SRNC_RRCd	NR_RRC	NR	30	1
21	M	NC(Radio Link Setup Request)	SRNC_RRCd	SRNC_NBAP	NC_NBAP	RNC_NC	123	2
22	M	Radio Link Setup Request	SRNC_NBAP	NodeB_NBAP	NBAP	Iub	123	3
23	M	Radio Link Setup Response	NodeB_NBAP	SRNC_NBAP	NBAP	Iub	75	2
24	M	NC(Radio Link Setup Response)	SRNC_NBAP	SRNC_RRCd	NC_NBAP	RNC_NC	75	2
25	M	NA(ALCAP Iub: Establish Request)	SRNC_RRCd	SRNC_ALCAP_IUB	NA_ALCAP	RNC_NA	63	2
26	M	ALCAP Iub: Establish Request	SRNC_ALCAP_IUB	NodeB_ALCAP	ALCAP	Iub	63	2
27	M	ALCAP Iub: Establish Confirm	NodeB_ALCAP	SRNC_ALCAP_IUB	ALCAP	Iub	7	1
28	M	NA(ALCAP Iub: Establish Confirm)	SRNC_ALCAP_IUB	SRNC_RRCd	NA_ALCAP	RNC_NA	7	1
29	C	Downlink Synchronisation	SRNC	NodeB	DCH FP	Iub	35	
30	C	Uplink Synchronisation	SRNC	SRNC	DCH FP	Iub	42	
31	C	DPCH Synchronisation Delay	SRNC					
32	M	NR(RRC Connection Setup)	SRNC_RRCd	SRNC_RRCc	NR_RRC	NR	121	2
33	M	RRC Connection Setup	SRNC_RRCc	UE_RRCc	RRC(CCCH)	Uu+Iub	121	3
34	M	UR(RRC Connection Setup)	UE_RRCc	UE_RRCd	UR_RRC(CCCH	UR	121	2
35	M	RRC Connection Setup Complete	UE_RRCd	SRNC_RRCd	RRC(DCCH)	Uu+Iub	22	1
36	P	**NAS Signalling Connection Establishment**						

Figure 28: The load model (see [55])

The format of the Excel sheet is defined as illustrated in Figure 28, which displays the Excel representation of the "RRC Connection Setup" procedure. After completing the Excel sheet with the signalling sequences of interest a transformation algorithm implemented by the VBA (Visual Basic Application) is started to generate an OPNET suitable table representation of the MSCs kept in the Excel sheet. MINA tool uses this Excel sheet. When setting up the load model within Excel, the following must be specified:

- The names of the MSCs and single signalling messages
- The FEs which send and receive signalling messages
- The used protocols and interfaces
- The message lengths as well as complexity classes for particular signalling messages
- Some information relevant for the VBA transformation of the load model.

All messages within the SFs are ranged within three complexity classes. The resources and their processing properties are called the **Resource Model**. The resource speed factor together with the complexity class factor is used to calculate the resource capability for messages belonging to that complexity class (see Figure 29).

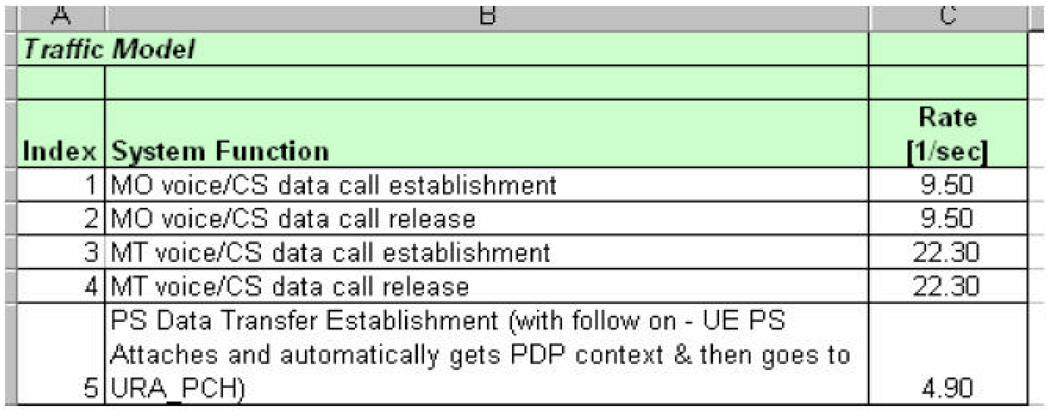


	A	B	C	D	E
1	*Resource*	*Configuration Table*		Capability (msg/s)	
2	Index	Resource	compl. 1	compl. 2	compl. 3
3	0	UE_CPU	1000	1000	1000
4	1	NB_UMC	63	42	21
5	2	RNC_BSC	181	93	5
6	3	RNC_TICC	66666	34166	1666
7	4	RNC_GICC	66666	34166	1666
8	5	RNC_SICC	66666	34166	1666
9	6	CN_CPU	100000	100000	100000

Figure 29: The resource model (see [71])

To impose a relevant load onto the system a Traffic Model is specified. The Traffic Model contains the execution rates, i.e. the number of instantiations per time unit for each MSC. Part of the Traffic Model is shown in Figure 30.

In order to clarify the modelling concept presented above, the “RRC Connection Setup” procedure is taken as an example. The RRC Connection Setup establishes the radio connection between the UE and the Node B and further connects the UE to the RNC. It is used in several voice and data call establishment-signalling sequences to set up a signalling channel between mobile terminals (UE) and the corresponding RNC via a base station (Node B).



A	B	C
Traffic Model		
Index	**System Function**	**Rate [1/sec]**
1	MO voice/CS data call establishment	9.50
2	MO voice/CS data call release	9.50
3	MT voice/CS data call establishment	22.30
4	MT voice/CS data call release	22.30
5	PS Data Transfer Establishment (with follow on - UE PS Attaches and automatically gets PDP context & then goes to URA_PCH)	4.90

Figure 30: The traffic model (see [71])

Figure 31 depicts the "RRC Connection Setup" procedure in the form of a HMSC as described in [59]. In Figure 31 the network elements UE, RNC and Node B communicate with each other by exchanging signalling messages.

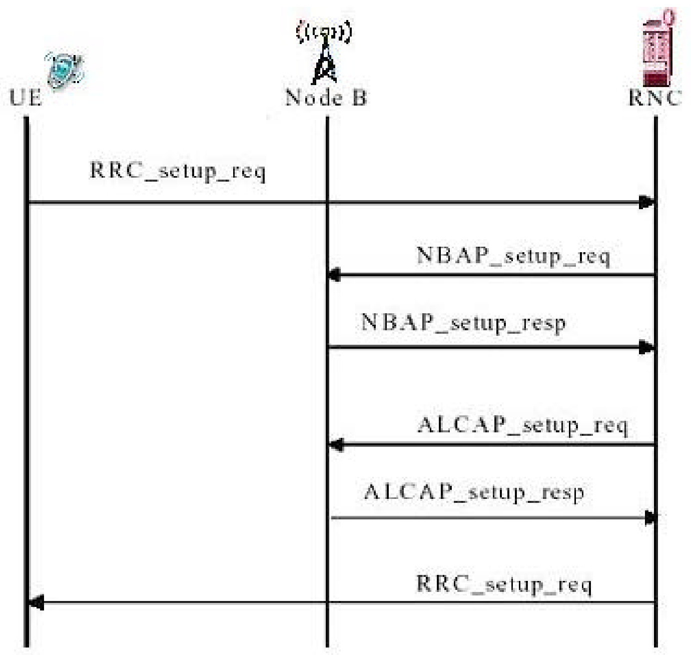


Figure 31: HMSC for "RRC Connection Setup" (see [22, 55, 71 and 76])

To model the node internal structure for each node under investigation, the relevant network elements and the FEs to be modelled must be identified. Those are as follows:

- RRC (Radio Resource Control) protocol entity within the UE.
- ALCAP (Access Link Control Application Part) and NBAP (Node B Application

Part) protocol entities in the Node B.

- ALCAP, NBAP and RRC protocol entities within the RNC.

Because of this identification step, the MSCs specified in standards documents have to be refined so that the relevant protocol entities depicted in Figure 25 explicitly communicate with each other.

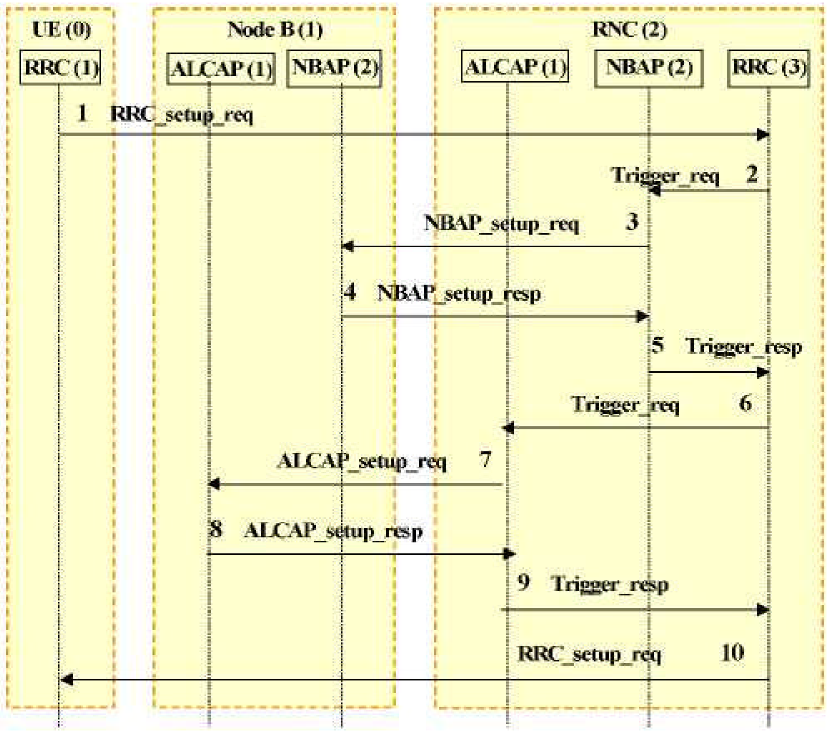


Figure 32: Refined MSC with FEs (see [22, 55, 71 and 76])

Figure 32 illustrates the refined MSC, which is extended by additional trigger messages in order to realize the exchange of signalling messages between FEs inside a node.

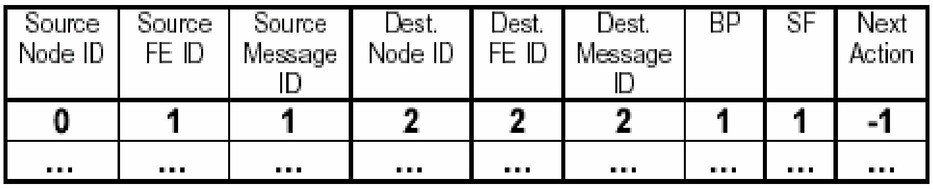
6.2 System Implementation

In this section, we describe the implemented model in some more detail. We already showed the network view of the OPNET model. All structural components can be instantiated several times to create more complex networks. The UE cluster aggregates the behaviour of all users. This aggregation is realized by a specific calculation of the packet interarrival time for each SF (e.g. MOBILE ORIGINATED CALL SETUP and RELEASE), depending on the number of active users.

Each FE keeps two tables with relevant processing and routing information for this particular FE in order to react as required by the MSC logic. These tables are called Action Table and Supplementary Action Table. These tables are derived from the refined MSCs, which are specified in Excel sheets. In order to derive these tables from the refined MSCs, the SFs are sequentially numbered and the Basic Procedures (BPs) within a SF (e.g. BP “RRC Connection Setup” within SF “MOBILE ORIGINATED CALL SETUP”) are identified, which are also numbered sequentially. Furthermore, the messages a BP consists of are sequentially numbered in order to identify each single message within an MSC. In addition, each network node and each FE get an unambiguous ID. In this context, the format of packets within the modelled system also has to be mentioned. It consists of the following:

- Source Node ID
- Source FE ID
- Destination Node ID
- Destination FE ID
- System Function
- Basic Procedure
- Message ID

The Source Node ID is the ID of the node, which contains the FE that sent the packet while the Source FE ID, is the ID of the FE that sent it. The Destination Node ID and Destination FE ID specify the node and FE, which is the next recipient of the packet. System Function and Basic Procedure specify the number of the respective SF and BP. The Message ID specifies the number of a single message within a BP. To see an example for the use of message numbers and node and FE IDs we refer to Figure 32.



Source Node ID	Source FE ID	Source Message ID	Dest. Node ID	Dest. FE ID	Dest. Message ID	BP	SF	Next Action
0	**1**	**1**	**2**	**2**	**2**	**1**	**1**	**-1**
...	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**

Figure 33: Extract from table of (FE RRC, Node RNC) (see [22])

Figure 33 shows an extract of the table for FE RRC in node RNC for the SF MOBILE ORIGINATED CALL SETUP. In Figure 32 FE 1 (= RRC) in node 0 (= UE cluster) sends message 1 (= RRC Setup Request) of BP 1 (= RRC Connection Setup) within SF 1 (= MOBILE ORIGINATED CALL SETUP) to FE 3 (= RRC) in Node 2 (= RNC). The content of the sent packet is as follows: Source Node ID = 0, Source FE ID = 1, Destination Node ID = 2, Destination FE ID = 3, SF = 1, Message ID = 1. If a FE gets a message, it has to extract the values of the parameters Source Node ID, Source FE ID, SF, BP and Message ID and compare them to its table. In this example case the table of FE RRC in Node RNC tells:

If the packet mentioned above is received, the packet's Message ID has to be changed from ID 1 to 2 and the Destination FE ID from 3 to 2. This means that FE 2 (= NBAP) in node 2 (= RNC) is the recipient of message 2 (= additional trigger message) of BP 1 (= RRC Connection Setup) within SF 1 (= MOBILE ORIGINATED CALL SETUP). Additionally the FE RRC in node RNC changes the Source Node ID of the packet from 0 (= UE cluster) to 2 (= RNC) and the Source FE ID from 1 (= RRC in UE cluster) to 3 (= RRC in RNC). We call this proceeding "message handling" within a FE. The column "Next Action" within the routing table points to the supplementary action table of a FE, which has to execute supplementary actions. A supplementary action means that an additional packet has to be generated. This can be necessary for example if forks occur within the MSCs (e.g. a FE has to acknowledge one message and has to send another message to a different FE) which lead to parallelisms within the MSCs.

If a FE has to execute a supplementary action there will be an integer value greater or equal to zero in the "Next Action" column which corresponds to the respective index of the row in the supplementary action table. If the value is set to –1 no supplementary action has to be executed. The supplementary action table provides the parameters for the additional packet that has to be generated. The different components of a node need different parameters from incoming packets in order to fulfil their routing functionality. We already mentioned that the FE needs the packet parameters Source Node ID, Source FE ID, SF, BP and Message ID.

It should be mentioned here that the OPNET (as well as JavaDEMOS simulator) node and network level models have to be modified only if new network elements are added or the allocation of FEs and resources is modified. Adding or modifying MSCs does not change the simulation model, because the impact of the SF logic is kept limited to the FE tables only which are loaded at simulation start. This allows for a simple data driven evaluation of various protocol scenarios. Results and Statistical evaluations

6.3 Results

6.3.1 Input Parameters

In this section, we will derive three examples, which gradually go from simple to complex to more complicated model. The three examples are for the system described by the load model shown in Figure 28, which has fourteen SFs. Messages of these SFs are exchanged through the fourteen FEs of the nodes system shown in Figure 24 and Figure 25. All FEs in the same node are mapped to a single resource.

In Table 16, the mapping between FEs of each node and the corresponding resource is defined. Table 16 also shows:

- The speed of each resource.
- The service amounts for complexity classes 1, 2 and 3 at each resource.
- The service amounts for complexity classes 1, 2 and 3 at each resource.
- The number of servers of each resource.

FEs Of node	Resource	message service amount			Speed (msg/s)	CPU Service Rate (msg/s)			No of Severs
		CC 1	CC 2	CC 3		CC 1	CC 2	CC 3	
UE	CPU1	1.00	1.00	1.00	10.00	10.00	10.00	10.00	10000
NodeB	CPU2	1.00	2.00	3.00	1.00	1.00	0.50	0.33	4.00
RNC	CPU3	1.00	2.00	36.00	60.00	60.00	30.00	1.67	2.00
CN	CPU4	1.00	1.00	1.00	100.00	100.00	100.00	100.00	20000

Table 16: System parameters

In the following, we show the results for these examples. The results will be for both simulation and queueing networks analysis. The model time in the three examples will be 24 hours. We also will compare these results with the OPNET MxRAN simulator results.

6.3.2 Model One

The first model is typically the model described in Figure 24, in which we have only one UE, one NodeB, one RNC and one CN. Table 17 and Table 18 summarize the results. Table 17 shows on one hand the mean response time results for both JavaDEMOS simulator and queueing networks analysis and on the other hand, OPNET simulator response time results. Table 17 also shows that the arrival rates for all SFs equal 0.03 per second. Table 18 shows the utilization for all resources.

System Function		**Arrival rates (1/s)**	**Mean response time [s]**		
			OPNET simulator	**JavaDEMOS simulator**	**QN Analysis**
1	MO voice/CS data call	0.03	13.2886	13.844	12.2670
2	MO voice/CS data call release	0.03	7.8075	8.076	6.7579
3	MT voice/CS data call	0.03	13.3261	14.023	12.5673
4	MT voice/CS data call release	0.03	3.4079	3.595	2.5744
5	PS Data Transfer Establishment (...)	0.03	10.8287	11.24	9.9703
6	PS Detach via power off (UE	0.03	8.5165	8.999	7.7568
7	Transition from URA_PCH to	0.03	5.7331	6.109	5.4200
8	Transition from CELL_DCH to	0.03	4.7842	4.937	4.4394
9	MO PDP Context Activation (...)	0.03	5.8673	6.086	5.3567
10	MO PDP Context Deactivation (...)	0.03	3.2866	3.533	2.5310
11	IMSI Detach Signalling Flow	0.03	7.8836	8.304	7.1495
12	Location Updating Signalling Flow	0.03	8.4477	8.864	7.5397
13	URA Update (URAU) Signalling	0.03	0.3725	0.398	0.2833
14	RA Update (RAU) Signalling Flow	0.03	0.3172	0.362	0.2100

Table 17: Response times for different SFs (model 1)

Resource	**Utilization (standardized) [%]**		
	OPNET simulator	**JavaDEMOS simulator**	**QN Analysis**
UE (pure delay)	0.00	0.00	0.00
Node B	42.84	42.86	42.75
RNC	43.71	43.62	43.45
CN.R (pure delay)	0.00	0.0	0.0
CN.R1 (pure delay)	0.00	0.0	0.0

Table 18: Resources utilization (model 1)

As we mentioned before, the CPU time needed to get the performance measures results by the queueing networks algorithms is some seconds. On the other hand, you need some hours and may be days to get these results using simulation. This is a very important point, that is one should use the results of queueing networks algorithms, which can be obtained very quickly, to investigate, in early design stages, which amount of traffic can be carried by the planned configuration. Such analytical results show the scope of possible parameter settings and allow a better planning of simulation scenarios which include more details and are closer to reality.

Table 19 compares the CPU time needed to get the results of the queueing networks algorithm and simulation. It shows that we can get the performance measures using the queueing network analysis very quickly (about 6 seconds) compared with the CPU time in case of the simulation (approximately 6 hours).

CPU time (min : sec)	
Simulation	**QN Algorithm**
240:00	00:06

Table 19: Model one: CPU time for JavaDEMOS vs. QN algorithm

6.3.3 Model Two

The second model is the same like the first one except that instead of one UE we have twenty UEs (see Figure 34).

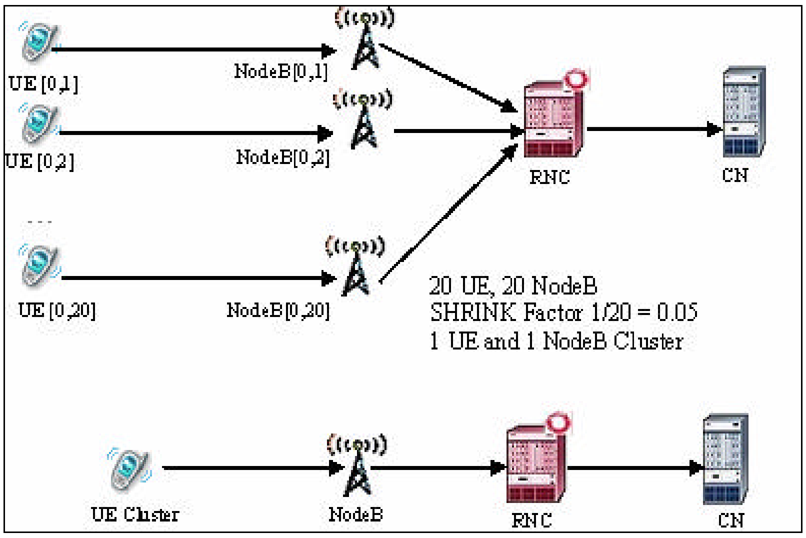


Figure 34: Physical network architecture (model 2)

Instead of modelling twenty UE nodes, we use an alternative approach called the SHRINK approach. As discussed before in section 3.7.2 the SHRINK-method is based on scaling down the complete model, cf. [2, 3, 63, 64 and 66]. To this end both arrival rates and service rates parameters is multiplied by a factor a, $0 < a < 1$. This factor is called the SHRINK factor. This new model is called the scaled down or shrinked model. This scaled down model works slower, i.e. arrivals are less frequent and service durations are longer, but the behaviour of both systems is approximately identical with respect to performance.

Here we assume a full-scale system with 20 UEs and 20 NodeBs. For a predefined rate of 0.03 the rate of the SFs starting in each UE should be 0.03/20=0.0015. Scaling down this system the rates of the UE SFs are still 0.0015. The scaled-down rates of the SFs starting in RNC or CN are 0.03/20=0.0015. Therefore, we have for all SFs a rate of 0.0015.

System Function		Arrival rates (1/s)	Mean response time [s]		
			OPNET simulator	JavaDEMOS simulator	QN Analysis
1	MO voice/CS data call establishment	0.03	12.8886	13.29	13.1426
2	MO voice/CS data call release	0.03	7.2632	7.609	6.7772
3	MT voice/CS data call establishment	0.03	12.9626	13.538	13.4120
4	MT voice/CS data call release	0.03	3.0799	3.234	3.4418
5	PS Data Transfer Establishment (...)	0.03	10.2919	10.848	10.1534
6	PS Detach via power off (UE initiated)	0.03	8.1946	8.458	7.9101
7	Transition from URA_PCH to	0.03	5.5593	5.833	5.4777
8	Transition from CELL_DCH to	0.03	4.6656	4.733	4.4726
9	MO PDP Context Activation (...)	0.03	5.7570	5.803	5.4304
10	MO PDP Context Deactivation (...)	0.03	3.1626	3.212	3.3411
11	IMSI Detach Signalling Flow	0.03	7.4859	7.859	7.1989
12	Location Updating Signalling Flow	0.03	8.1010	8.518	7.6689
13	URA Update (URAU) Signalling Flow	0.03	0.3536	0.399	0.2833
14	RA Update (RAU) Signalling Flow	0.03	0.3551	0.355	0.2100

Table 20: Response times for different SFs (model 2)

Resource	Utilization (standardized) [%]		
	OPNET simulator	JavaDEMOS simulator	QN Analysis
UE (pure delay)	0.00	0.0	0.0
Node B	2.15	2.15	2.14
RNC	44.39	43.79	43.45
CN.R (pure delay)	0.00	0.0	0.0
CN.R1 (pure delay)	0.00	0.0	0.0

Table 21: Resources utilization (model 2)

Table 20 and Table 21 summarize the results. Table 20 shows on one hand the response time results for both JavaDEMOS simulator and queueing networks analysis and on the other hand, OPNET simulator response time results. Table 20 also shows that the arrival rates for all SFs equal 0.03 per second. Table 21 shows the utilization for all resources.

Table 22 compares the CPU time needed to get the results of the queueing networks algorithm and simulation. It shows that we can get the performance measures using the queueing network analysis very quickly (about 6 seconds) compared with the CPU time in case of the simulation (approximately 40 minutes).

CPU time (min : sec)	
Simulation	**QN Algorithm**
40:00	00:06

Table 22: Model Two: CPU time for JavaDEMOS vs. QN algorithm

6.3.4 Model Three

The third model is the most complicated one and in which we consider nine UEs Clusters, each has twenty UEs and nine NodeBs Clusters, each has twenty NodeBs as in Figure 35, i.e. we assume a full-scale system with 180 UEs and 180 NodeBs.

For a predefined rate of 0.03 the rates of the SFs starting in each UE should be 0.03/180=0.00016667. Scaling down this system the rates of the UE SFs are still 0.00016667. With a SHRINK factor of 1/20 the scaled-down rates of the SFs starting in RNC or CN are 0.03/20=0.0015. The speed parameters in the parameter overview represent the full-scale system. They are modified according to the SHRINK factor during the simulation initial phase. The response time results also represent the full-scale system.

Table 23 and Table 24 summarize the results. Table 24 shows on one hand MINA tool response time results for both simulation and queueing networks analysis and on the other hand, MxRAN simulator response time results. Table 24 also shows that the arrival rates for all SFs equal 0.03 per second. Table 23 shows the utilization for all resources.

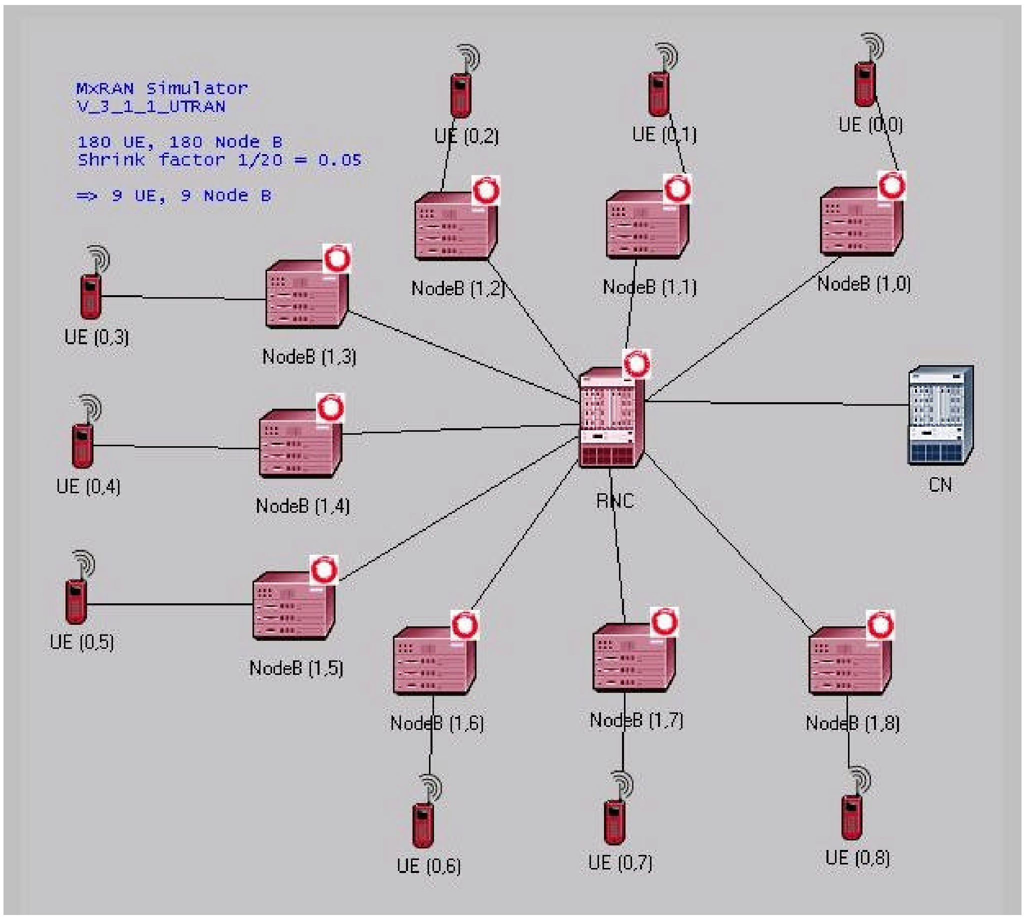


Figure 35: Physical network architecture (model 3)

System Function		Arrival rates (1/s)	Mean response time [s]		
			OPNET simulator	JavaDEMOS simulator	QN Analysis
1	MO voice/CS data call establishment	0.03	13.0316	13.338	12.2850
2	MO voice/CS data call release	0.03	7.6317	7.692	6.7735
3	MT voice/CS data call establishment	0.03	13.0532	13.586	12.5419
4	MT voice/CS data call release	0.03	3.3084	3.257	3.4785
5	PS Data Transfer Establishment (...)	0.03	10.3971	10.81	9.9822
6	PS Detach via power off (UE initiated)	0.03	8.2124	8.368	7.8400
7	Transition from URA_PCH to	0.03	5.5688	5.911	5.4039
8	Transition from CELL_DCH to	0.03	4.6857	4.825	4.4511
9	MO PDP Context Activation (...)	0.03	5.7009	5.751	5.3571
10	MO PDP Context Deactivation (...)	0.03	3.1636	3.194	3.3755
11	IMSI Detach Signalling Flow	0.03	7.5219	7.685	7.1777
12	Location Updating Signalling Flow	0.03	7.9470	8.546	7.6506
13	URA Update (URAU) Signalling Flow	0.03	0.3755	0.402	0.2833
14	RA Update (RAU) Signalling Flow	0.03	0.3349	0.357	0.2100

Table 23: Response times for different SFs (model 3)

Resource	**Utilization (standardized) [%]**		
	OPNET simulator	**JavaDEMOS simulator**	**QN Analysis**
UE (pure delay)	0.00	0.00	0.00
Node B	0.23	0.24	0.24
RNC	44.76	43.46	43.58
CN.R (pure delay)	0.00	0.00	0.00
CN.R1 (pure delay)	0.00	0.00	0.00

Table 24: Resource utilization (model 3)

Table 25 compares the CPU time needed to get the results of the queueing networks algorithm and simulation. It shows that we can get the performance measures using the queueing network analysis very quickly (about 10 seconds) compared with the CPU time in case of the simulation (approximately one hour).

CPU time (min : sec)	
Simulation	**QN Algorithm**
60:00	00:10

Table 25: Model Three: CPU time for JavaDEMOS vs. QN algorithm

6.3.5 Results Discussion

In this section, we introduce some remarks on the results of the three models of sections 6.3.2, 6.3.3 and 6.3.4.

The first remark is that if we look at the utilization results in Table 18, Table 21 and Table 24 we find that the results of the utilization obtained by the OPNET simulator, JavaDEMOS simulator and the queueing networks analysis are identical for all resources.

The situation is different in the case of the response time results. If we look at Table 17, Table 20 and Table 23, we find that there is a considerable difference in the results between OPNET simulator, JavaDEMOS simulator and the queueing network analysis. This difference between the results of the JavaDEMOS simulator and the OPNET simulator is due to using different types of service time distribution. The OPNET simulator uses a deterministic service time distribution whereas the JavaDEMOS simulator uses a service time distribution of type negative exponential.

System Function		Mean response time [s]		
		OPNET simulator	JavaDEMOS simulator	
			Neg. Exp.	Deterministic
1	MO voice/CS data call	13.2886	13.844	13.376
2	MO voice/CS data call release	7.8075	8.076	7.792
3	MT voice/CS data call	13.3261	14.023	13.684
4	MT voice/CS data call release	3.4079	3.595	3.453
5	PS Data Transfer Establishment (...)	10.8287	11.24	10.944
6	PS Detach via power off (UE	8.5165	8.999	8.662
7	Transition from URA_PCH to	5.7331	6.109	5.817
8	Transition from CELL_DCH to	4.7842	4.937	4.792
9	MO PDP Context Activation (...)	5.8673	6.086	5.904
10	MO PDP Context Deactivation (...)	3.2866	3.533	3.342
11	IMSI Detach Signalling Flow	7.8836	8.304	8.014
12	Location Updating Signalling Flow	8.4477	8.864	8.526
13	URA Update (URAU) Signalling	0.3725	0.398	0.366
14	RA Update (RAU) Signalling Flow	0.3172	0.362	0.322

Table 26: The response time in the case using deterministic service time distribution (model 1)

Table 26 shows the response time results for the JavaDEMOS simulator after using deterministic service time distribution. We can notice that these are closer to the results obtained by the OPNET simulator and of course are different from the results obtained by the JavaDEMOS in the case of using negative exponential service time distribution.

System Function		Mean response time [s]		
		OPNET simulator	JavaDEMOS simulator	
			Neg. Exp.	Deterministic
1	MO voice/CS data call	12.8886	13.29	13.21
2	MO voice/CS data call release	7.2632	7.609	7.477
3	MT voice/CS data call	12.9626	13.538	13.545
4	MT voice/CS data call release	3.0799	3.234	3.181
5	PS Data Transfer Establishment (...)	10.2919	10.848	10.747
6	PS Detach via power off (UE	8.1946	8.458	8.369
7	Transition from URA_PCH to	5.5593	5.833	5.698
8	Transition from CELL_DCH to	4.6656	4.733	4.687
9	MO PDP Context Activation (...)	5.7570	5.803	5.803
10	MO PDP Context Deactivation (...)	3.1626	3.212	3.152
11	IMSI Detach Signalling Flow	7.4859	7.859	7.737
12	Location Updating Signalling Flow	8.1010	8.518	8.279
13	URA Update (URAU) Signalling	0.3536	0.399	0.38
14	RA Update (RAU) Signalling Flow	0.3551	0.355	0.323

Table 27: The response time in the case using deterministic service time distribution (model 2)

System Function		Mean response time [s]		
		OPNET simulator	JavaDEMOS simulator	
			Neg. Exp.	Deterministic
1	MO voice/CS data call	13.0316	13.338	13.372
2	MO voice/CS data call release	7.6317	7.692	7.515
3	MT voice/CS data call	13.0532	13.586	13.653
4	MT voice/CS data call release	3.3084	3.257	3.263
5	PS Data Transfer Establishment (...)	10.3971	10.81	10.856
6	PS Detach via power off (UE	8.2124	8.368	8.458
7	Transition from URA_PCH to	5.5688	5.911	5.742
8	Transition from CELL_DCH to	4.6857	4.825	4.722
9	MO PDP Context Activation (...)	5.7009	5.751	5.868
10	MO PDP Context Deactivation (...)	3.1636	3.194	3.206
11	IMSI Detach Signalling Flow	7.5219	7.685	7.765
12	Location Updating Signalling Flow	7.9470	8.546	8.35
13	URA Update (URAU) Signalling	0.3755	0.402	0.387
14	RA Update (RAU) Signalling Flow	0.3349	0.357	0.327

Table 28: The response time in the case using deterministic service time distribution (model 3)

Table 27 and Table 28 show also the response time results for the JavaDEMOS simulator after using deterministic service time distribution for models 2 and 3. They also compare those results with the results obtained by the OPNET simulator and the results of the JavaDEMOS simulator in the case of using negative exponential service time distribution.

Now we will discuss another point that is the difference between the results of the JavaDEMOS simulator and the analytical queueing network formulas. The reason behind this difference is due to that, we use an approximate method (the method of decomposition) to calculate the response time because none of the product form methods is suitable for our problem. The decomposition method enhances the response time by calculating a correction factors that depends on the coefficients of variation of the arrival time distribution and the service time distribution. For our problem, we use arrival time distribution and service time distribution of type negative exponential. In this case, these coefficients of variation are of course equal to one and hence the calculated correction factor equals one. This situation makes the approximate method inefficient to produce better values for the response time in the case of using arrival time distribution and service time distribution of type negative exponential.

Arrival rates (1/s)	**Mean response time of SF1 [s]**		**Error**	**Resource %utilization**	
	QN analysis	**JavaDEMOS Simulation**		**NodeB**	**RNC**
0.002	12.03004	12.032	0.001956	2.137 %	2.175 %
0.01	12.03245	12.175	0.142552	14.079 %	14.25 %
0.02	12.0436	12.544	0.500398	28.313 %	28.808 %
0.03	12.07615	13.376	1.299848	42.736 %	43.501 %
0.04	12.17079	15.019	2.848206	57.233 %	58.179 %
0.05	12.50712	18.34	5.832877	71.423 %	72.558 %
0.06	14.6646	28.657	13.9924	85.532 %	86.886 %

Table 29: The mean response time for SF 1 for different arrival rate values (model 1)

We made several runs for the simulation as well as the analytical queueing network algorithm for model one as an example. We used a different value for the arrival rate at each run. The arrival rates used are the same for all SFs. The arrival rates used are shown in Table 29.

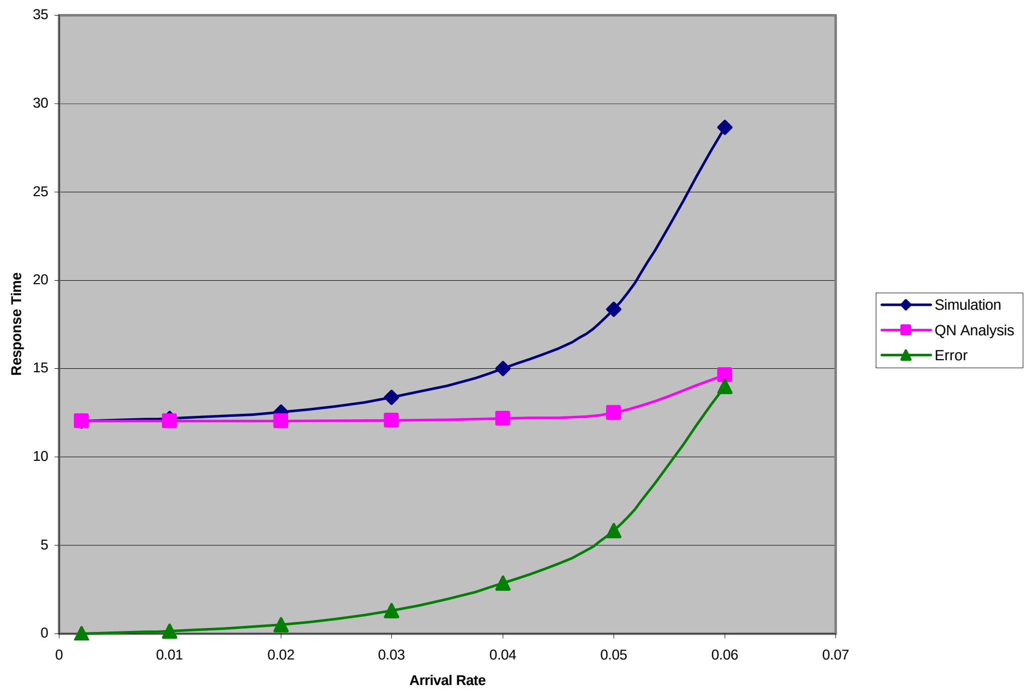


Figure 36: The mean response time of SF 1 for different arrival rate values (model 1)

We noticed that the response time error or the difference between the response times obtained by simulation and analytical queueing networks algorithm grows up when the value of the arrival rate increases for all system functions. The reason is that when the arrival rates increase the utilization of the resources increase and hence the response time increases. To demonstrate this fact we introduced the response time results for system function one as an example. Table 29 shows the obtained results from the JavaDEMOS simulator and the queueing networks algorithm as well as the difference between the two obtained results at different values of the arrival rate. Similar table can be obtained for the rest of the system functions.

Figure 36 show a graph of the results in Table 29. The graph shows that the error starts to grow up rapidly at arrival rate 0.02 and more. At arrival rate 0.2 the utilization of the NodeB and node RNC are 28.313 % and 28.808 % respectively as shown in Table 29. When the arrival rates are 0.06, the system is under a heavy load and the utilization of NodeB and node RNC increase to 85.532 % and 86.886 % respectively and the error becomes very large.

Chapter 7

Conclusion

7.1 Introduction

In this work, we considered methods and techniques for the performance evaluation of distributed systems. We developed a tool to evaluate the performance of such systems by analytic as well as by simulative techniques.

The tool chain is as follows:

- The system workload is described by MSCs.
- Then, notions for time consumption and resources are added in order to extend MSCs.
- The "Performance extended MSC" is included in a system performance model (Queueing Network Model).
- After that the performance evaluation by analytical techniques or by discrete event simulation can be done.
- Analytical techniques are used to obtain steady state performance measures like resource utilizations, throughput, and end-to-end delays.
- Additionally, simulation allows for the investigation of dynamic performance behaviour.

In the following we will review the different chapter of the thesis.

7.2 Summary of the Thesis

The system under study is initially described by means of Message Sequence Charts (MSCs). A Message Sequence Chart describes the message interaction between system components and their environment. More complex scenarios can be described by HMSCs which result from composing MSCs by a certain composition operator like sequential or parallel or loop or conditional composition parameters. A detailed discussion about MSCs, HMSCs and the compositional operators is given in Chapter 1.

In Chapter 2 we show how the MSC or HMSC descriptions can be extended by notions for time consumption and resources and afterwards included in a system performance model. Each message is associated with a service amount a_i to be executed at the receiving instance i. Furthermore, we consider the instances to behave like queueing stations, i.e. messages arriving at a busy instance are stored in a queue and will have to wait for service. Therefore, each message has to spend some wait time at arrival at an instance (including the case of zero wait time) followed by a service time which depends on speed of the instance and the required service amount. Moreover we consider MSCs to be “open“, i.e. the start of an MSC is triggered from the environment according to some interarrival distribution. Since we will employ analytical mean value formulas based on queueing network theory the interarrival distribution is assumed to be negative exponential. The same assumption is made for the service amounts. Since instances are, queueing stations and the messages can be considered to be customers or customers we obtain a queueing network. Each queueing station consists of a wait queue and a server. Here we assume that each station is of type -/M/1-FCFS and MSC arrivals occur according to a Poisson stream and the service times of the messages are also negative exponentially distributed.

The tool executes either a discrete event simulation using a simulation program implemented by JavaDEMOS package or alternatively by a suitable queueing network method to get the performance results. In this way steady state performance measures like resource utilizations and end-to-end delays can be calculated with low effort. The simulation uses the same input like the analytical formulas and allows for the investigation of dynamic performance behaviour or for the study of models including features, which cannot be handled by analytical formulas.

Chapter 3 describes how to calculate the response times for each MSC and the utilization at each queueing station in the queueing network model we considered. To do these calculations different algorithms are used. Jackson method [32 and 33] is used in the case that the queueing stations have a FCFS queueing disciplines, have a single class of customers and have a single server. An extension to the method of Jackson, the BCMP method [7], is used for networks that have queueing stations of queueing disciplines rather than FCFS like for example IS (Infinite Server) and PS (Processor Sharing). The open queueing networks with M/M/m queueing stations, FCFS queueing discipline, multiclass of customers with different service rates for different classes of customers do not satisfy the conditions of Jackson's method and the BCMP method and we can not use these methods to analyze such queueing networks. For this reason, a non-product form approximate method called *decomposition* method [12, 15, 24, 41, 67, 74 and 75] is used. Some remarks on how to apply the queueing networks formulas in some special cases are discussed in section 3.7. A remark on how to deal with the queueing networks formulas in the case that we have a set of MSCs describes the system. In this case, messages can be distinguished not only by the complexity class of the message itself but also by the MSC it belongs to. A discussion about the use of the SHRINK approach [2, 3, 63, 64 and 66] with the analytical formulas in case of the so called slow down models is described. Another remark about branching in an MSC and about how to calculate the response time for a certain predefined branch is also discussed. An overview of how to calculate the end-to-end delay in the case of systems described by HMSCs is given in section 3.8.

Chapter 4 described the simulation model which can be used alternatively. We presented how the system components are modelled in the context of discrete event simulation concepts. The scheduling process of instances of each MSC according to the predefined traffic is also described. We explained the way of scheduling of messages of each MSC. Simulation input parameters and how it is used by the simulator is also shown. We discussed some of the developed building blocks that are responsible for the reading of the input parameters, the resources behaviour, the scheduling of messages and the main simulation class. Finally, we discussed the output of the simulation.

In Chapter 5, we presented two case studies to make following the tool chain more clear. The case studies are taken from the Client Server systems examples (textbook in [45]). These systems are originally described by communication process delay diagrams. We defined rules to transfer such diagrams into MSCs. After that, we used the same resources speeds as in the original example and mapped the extended MSC into a queueing network model. We solved it using simulation and analytical queueing networks analysis and compared the obtained results with results in the textbook.

Chapter 6 shows how the tool can be used to evaluate the performance of complex communication systems using both simulation and queueing network models. The presentation follows closely the work published in the context of the IPonAir project on architectures of future mobile communication systems [31], see also [22, 55, 71 and 76]. A part of this project is the development of a discrete event simulation system, which is to study the performance behaviour of different system designs. The system under study is described by the 14 system functions (MSCs) with 482 messages exchanged between stations UE, NodeB, RNC and CN. We presented 3 Models, applied the tool for each one and obtained results identical to results obtained from the MxRAN simulator.

7.3 Contributions

In this thesis we suggest a tool chain, named MINA, to evaluate performance measures of distributed systems, e.g. communication systems and computer systems. The MINA tool chain describes the communication between system components by means of Message Sequence Charts. These system components exchange messages between each other and behave like queueing stations (resources) with one or more servers that serve customers (messages). To calculate performance measures for these systems we extend the MSC description by some performance parameters. These parameters are related to resources and their processing properties. Other parameters, e.g. message arrival rates, impose a relevant load onto the system.

After that, we map this extended MSC description into a queueing model. Based on this unique model the performance measures of the system under study can be obtained either by queueing network analysis with low effort or alternatively by simulation which allows for the investigation of dynamic performance behaviour or for the study of features which can not be handled by analytical formulas.

The queueing network model we use is an open model and the MINA tool offers some methods, e.g. Jackson's method, the BCMP method or an approximate algorithm, for calculating the system performance measures, e.g. the response time for different MSCs and the utilization of the resources.. The queueing network algorithms are useful to investigate in early design stages which amount of traffic can be carried by the planned configuration, or the other way round, what kind of resources are needed to carry the traffic under specified service levels. Such analytical results may be extremely helpful for a system developer. It can show the scope of possible parameter settings and allow a better planning of simulation scenarios, which include more details and are closer to reality. Another important point is that the analytical results require just some seconds of CPU time, whereas the simulation needs some hours or even some days.

Additional to the analytical formulas a simulation system has been developed, which uses the same input as the analytical model and delivers the same results, i.e. approximate mean values and additionally confidence intervals. Of course the simulator can be used to evaluate models which do not satisfy the conditions necessary for analytical evaluations; important examples for model features which violate these conditions are non-exponential distributed service times (e.g. low service time variations or even deterministic service), non-Poisson arrivals (e.g. bursty sources) and priority scheduling. Moreover, the state of all objects, e.g. resources, can be inspected at any time. Observing the dynamic performance behaviour of the values of the state variables of the resources, in particular the maximum queue length, the current queue length, the average queue length and the average wait time parameters can be inspected at any time.

Summarizing, the main progress of this thesis is given by the following results:

- Extension of a (semi) formal description technique (these are the MSCs) by performance parameters,
- its mapping to an executable performance model,
- the development of analytical formulas yielding exact or approximate mean values

for performance metrics, and alternatively,

- the simulative evaluation of the model by discrete event simulation.

7.4 Future Work

The work presented in this thesis can be extended in several directions. We propose here some considerations on possible future extensions of the approach previously described.

The tool has been applied to Client/Server systems and also to mobile communication systems. The possibility of applying the tool to other distributed systems like for example mobile Ad-hoc networks, peer to peer networks, sensor networks[8], sensitive networks[9], or any other distributed system to obtain system performance is one direction that needs more work in the future.

The analytical queueing network analysis is very helpful to be used in calculating the performance measures like resource utilizations and end-to-end delays with low effort. We used some methods for this purpose, but more methods are needed to cover a variety of situations that arises when we try to solve systems that are more complex.

Another point is to develop a graphical user interface for the tool. This graphical user interface could be used to enable users to execute simulation as well as analytical queueing networks methods. The graphical user interface will help the user of the tool to edit input and obtain results in an easy way.

[8] Sensor Networks are distributed networks made up of small sensing devices equipped with processors, memory, and short-range wireless communication.
[9] Sensitive networks are networks in which the introduction or the removal of a node/vertex dramatically changes the dynamic structure of the system.

Appendix A: JavaDEMOS Simulator

A.1 Introduction

The object oriented language SIMULA and its classes SIMULATION and DEMOS have been used for purposes of teaching for nearly three decades. In particular, the class DEMOS, which implements a scenario approach providing building blocks to allows for the flexible and effective construction of simulation programs.

JavaDEMOS is a Java library for discrete event simulation, which was inspired by the DEMOS system written by G. M. Birtwistle. JavaDEMOS is based on an implementation of the DEMOS features in Java. The syntax of the procedures is as close as possible to DEMOS, in order to simplify the translation of DEMOS programs to JavaDEMOS.

In addition, JavaDEMOS consists of a graphical front-end which permits the visualization of a simulation run and which allows basic interactions with the simulation system. The user can observe the scheduled entities in the event list, the state of model components, statistical data, and a simulation trace. Simulations can be executed completely, in single step mode or until reaching of a certain time or entity.

A.2 JavaDEMOS Concepts

Here a very brief summary of the JavaDEMOS concepts are given. The basic concept is the entity. Entities implement behaviour patterns, may acquire and release resources, may wait until certain conditions are fulfilled, are able to interact with each other in a master/slave mode and can of course be scheduled in the event list. For a thorough description, we refer to the original DEMOS documentation, in particular to the book and manual which are both due to Graham Birtwistle [10 and 11].

A.2.1 Entities and their Scheduling

The class **Entity** has its local scheduling methods. JavaDEMOS implements its own event list. The global scheduling methods are **hold ()** and **passivate (); time ()** returns the current model time. Only JavaDEMOS entities may be queued; if you wish to queue other items, you will have to write the methods yourself. The types of queue implemented are as follows:

- **Queue** (usually for holding several coopted entities until they are required by their masters),
- **WaitQ** (master/slave synchronization), and
- **CondQ** (waits until).

A.2.2 Reporting Aids

JavaDEMOS contains reporting aids like class **Report,** in particular all JavaDEMOS classes extend **Tab** allows **Report** membership. On generation, each facility object is entered into a special **Report** reserved for its type. In **Report,** all **Tab** objects are registered. **Report** offers the methods report and reset to invoke automatically the corresponding methods of all registered **Tab** objects. It is now very easy to write routines to report or reset each facility object created during program execution. There are data collections devices like:

- **Count** (incidences),
- **Tally** (time independent data),
- **Accumulate** (time dependent data), and
- **Regression** (for linear regressions).

Each of these classes extends class **Tab.** Another one is class **Histogram** (**Tally** plus a bar chart) which as well extends class **Tally.**

An additional feature of JavaDEMOS is the observation of time dependent behaviour of some performance measures. Furthermore, features for an extended output analysis have been developed. There are the new classes **BatchMeans** and **Confidencelnterval** for the analysis of interval estimates.

A.2.3 Random Numbers

JavaDEMOS contains random number generators as well as its method of generating well spread seeds. All distribution objects are extensions of class **Dist.** Distributions producing double results (**Constant, Empirical, Erlang, NegExp, Normal** and **Uniform)**. Distributions producing integer results **(Poisson, Randint)** distributions producing Boolean results **(Draw).** JavaDEMOS contains corresponding classes for those of DEMOS.

A.2.4 Resources "Classes Res and Bin"

There exists a class **Resource** and its subclasses **Res** (for the mutual exclusion synchronization) and **Bin** (for the producer/consumer synchronization).

When resources are shared, but they must only be accessed by one process at a time, one has a *mutual exclusion* situation. Examples are road intersections, tools, or file sharing by reading and writing processes. In these situations, resources are requested and released by the same process. A process requesting an unavailable resource must wait (is blocked).

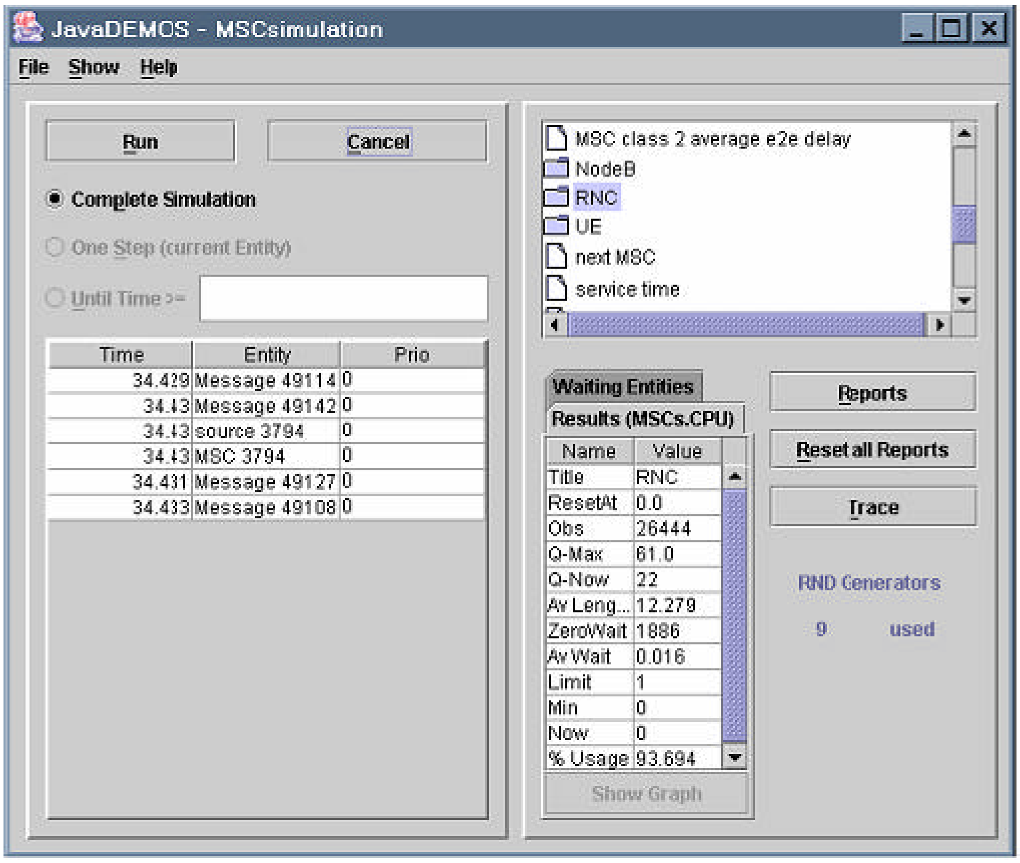
In producer/consumer synchronizations, producer processes make items available to consumer processes. Examples are a message sender and a message receiver, or two machines working on items in sequence. The synchronization here must ensure that the consumer process does not consume more items than have been produced. If necessary, the consumer process is blocked (must wait) if no item is available to be consumed. Producer and consumer processes are coupled by a buffer to allow asynchronous production and consumption. The buffer can be bounded (have a capacity limit) or unbounded (be able to store an unlimited number of items).

A.3 JavaDEMOS Package

JavaDEMOS consists of four packages:

- package default
- package demos

- ➢ package demosGui
- ➢ package result



● Complete Simulation

Time	Entity	Prio
34.429	Message 49114	0
34.43	Message 49142	0
34.43	source 3794	0
34.43	MSC 3794	0
34.431	Message 49127	0
34.433	Message 49108	0

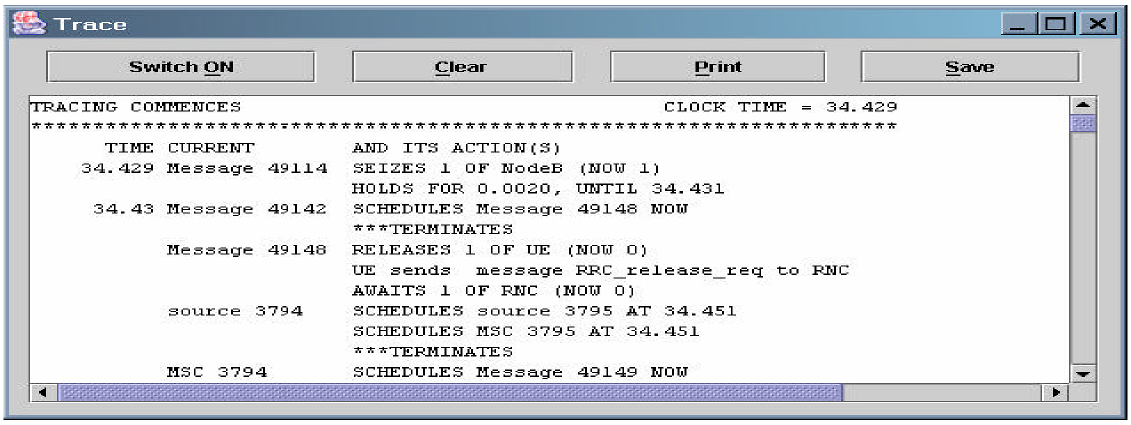
Name	Value
Title	RNC
ResetAt	0.0
Obs	26444
Q-Max	61.0
Q-Now	22
Av Leng...	12.279
ZeroWait	1886
Av Wait	0.016
Limit	1
Min	0
Now	0
% Usage	93.694

Figure 37: The graphical user interface of JavaDEMOS

ExecuteDEMOS is a part of the package demosGui and it is the simulation environment of JavaDEMOS. Firstly, we edit the program using any Java editor. Then we start the simulation by clicking the icon named run in the subdirectory run provided with JavaDEMOS package. When the simulation environment starts, the user is asked to select an Entity. The user selects a java class then presses the button open. After that, a new window appears with four options to control simulation. These four options are:

- ➢ **Complete Simulation:** The simulation will be carried out completely.
- ➢ **One Step (Current Entity):** The first Entity in the event list will be carried out.
- ➢ **Until time** >= x: The simulation will be carried out until time x will be reached.
- ➢ **Until Entity** = x: The simulation stops when the desired Entity becomes active.

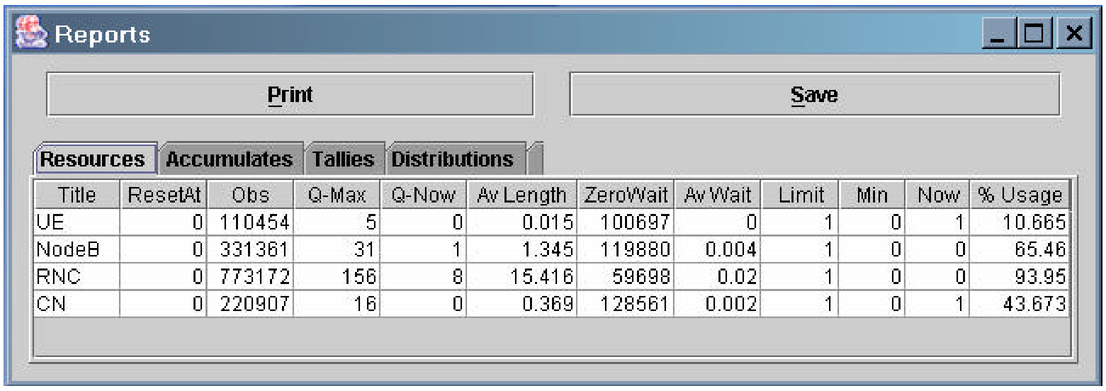
Afterwards the **Run** button is pressed. Figure 37 (on the left) shows a number of entities which are scheduled in the event list and their associated event times. It shows that messages, MSCs (SFs) and the sources (Traffic sources) are entities and also shows that the visited stations (UE, NodeB, …) are modelled as resources. Moreover, the state of all objects can be inspected at any time during the simulation. Figure 37 (on the right) shows the current values of the state variables of the station RNC, in particular the maximum queue length Q-MAX, the current queue length Q-NOW, the average queue length and the average wait time.



```
TRACING COMMENCES                                          CLOCK TIME = 34.429
*********************************************************************
      TIME CURRENT        AND ITS ACTION(S)
    34.429 Message 49114  SEIZES 1 OF NodeB (NOW 1)
                          HOLDS FOR 0.0020, UNTIL 34.431
     34.43 Message 49142  SCHEDULES Message 49148 NOW
                          ***TERMINATES
           Message 49148  RELEASES 1 OF UE (NOW 0)
                          UE sends  message RRC_release_req to RNC
                          AWAITS 1 OF RNC (NOW 0)
           source 3794    SCHEDULES source 3795 AT 34.451
                          SCHEDULES MSC 3795 AT 34.451
                          ***TERMINATES
           MSC 3794       SCHEDULES Message 49149 NOW
```

Figure 38: Trace-window

In Figure 38 we can see the Trace-window in which we can see gradually how the simulation proceeds. It shows the model times, the entities and their actions. Finally, a complete report can be obtained as shown in Figure 39. In addition, we can print the report by clicking on the **Print** button and save it on disk by clicking on Save.



Title	ResetAt	Obs	Q-Max	Q-Now	Av Length	ZeroWait	Av Wait	Limit	Min	Now	% Usage
UE	0	110454	5	0	0.015	100697	0	1	0	1	10.665
NodeB	0	331361	31	1	1.345	119880	0.004	1	0	0	65.46
RNC	0	773172	156	8	15.416	59698	0.02	1	0	0	93.95
CN	0	220907	16	0	0.369	128561	0.002	1	0	1	43.673

Figure 39: Report-window for simulation environment

Appendix B: Client Server Systems

B.1 Introduction to Client-Server Systems

Client/server (C/S) systems are comprised of many different hardware resources including client workstations, servers with their processors and disks, LANs, WANs, and routers. Various types of software processes including applications, middleware, database management systems, protocol handlers, and operating systems share the use of the hardware resources. The shared use of these resources gives rise to contention that generates waiting queues. A C/S transaction spends a portion of its time receiving service at various resources as well as queuing for these resources. The delays encountered by a C/S transaction may be decomposed into:

- Service times: time spent using various resources such as processors, disks, and networks.
- Waiting times: time spent waiting to use resources that are being held by other transactions.

Client/server computing is the logical extension of modular programming. Modular programming has as its fundamental assumption that separation of a large piece of software into its constituent parts ("modules") creates the possibility for easier development and better maintainability. Client/server computing takes this a step farther by recognizing that those modules need not all be executed within the same memory space. With this architecture, the calling module becomes the "client" (that which requests a service), and the called module becomes the "server" (that which provides the service).

The logical extension of this is to have clients and servers running on the appropriate hardware and software platforms for their functions. For example, database management system servers running on platforms specially designed and configured to perform queries, or file servers running on platforms with special elements for managing files. It is the latter perspective that has created the widely believed myth that client/server has something to do with PCs or UNIX machines.

B.1.1 A Client process

The client is a process (program) that sends a message to a server process (program), requesting that the server perform a task (service).

Client programs usually manage the user-interface portion of the application, validate data entered by the user, dispatch requests to server programs, and sometimes execute business logic. The client-based process is the front-end of the application that the user sees and interacts with. The client process contains solution-specific logic and provides the interface between the user and the rest of the application system. The client process also manages the local resources that the user interacts with such as the monitor, keyboard, workstation CPU and peripherals. One of the key elements of a client workstation is the graphical user interface (GUI). Normally a part of operating system i.e. the window manager detects user actions, manages the windows on the display and displays the data in the windows.

B.1.2 A Server Process

A server process (program) fulfils the client request by performing the task requested.

Server programs generally receive requests from client programs, execute database retrieval and updates, manage data integrity and dispatch responses to client requests. Sometimes server programs execute common or complex business logic. The server-based process "may" run on another machine on the network. This server could be the host operating system or network file server; the server is then provided both file system services and application services. Alternatively, in some cases, another desktop machine provides the application services. The server process acts as a software engine that manages shared resources such as databases, printers, communication links, or high powered-processors. The server process performs the back-end tasks that are common to similar applications.

B.2 Characteristics of Client / Server Architecture

The basic characteristics of client / server architectures are:

- Combination of a client or front-end portion that interacts with the user, and a server or back-end portion that interacts with the shared resource. The client process contains solution-specific logic and provides the interface between the user and the rest of the application system. The server process acts as a software engine that manages shared resources such as databases, printers, modems, or high powered processors.
- The front-end task and back-end task have fundamentally different requirements for computing resources such as processor speeds, memory, disk speeds and capacities, and input / output devices.
- The environment is typically heterogeneous and multi-vendor. The hardware platform and operating system of client and server are not usually the same. Client and server processes communicate through a well-defined set of standard application program interfaces (API's) and RPC's.
- An important characteristic of client-server systems is scalability. They can be scaled horizontally or vertically. Horizontal scaling means adding or removing client workstations with only a slight performance impact. Vertical scaling means migrating to a larger and faster server machine or multi-servers.

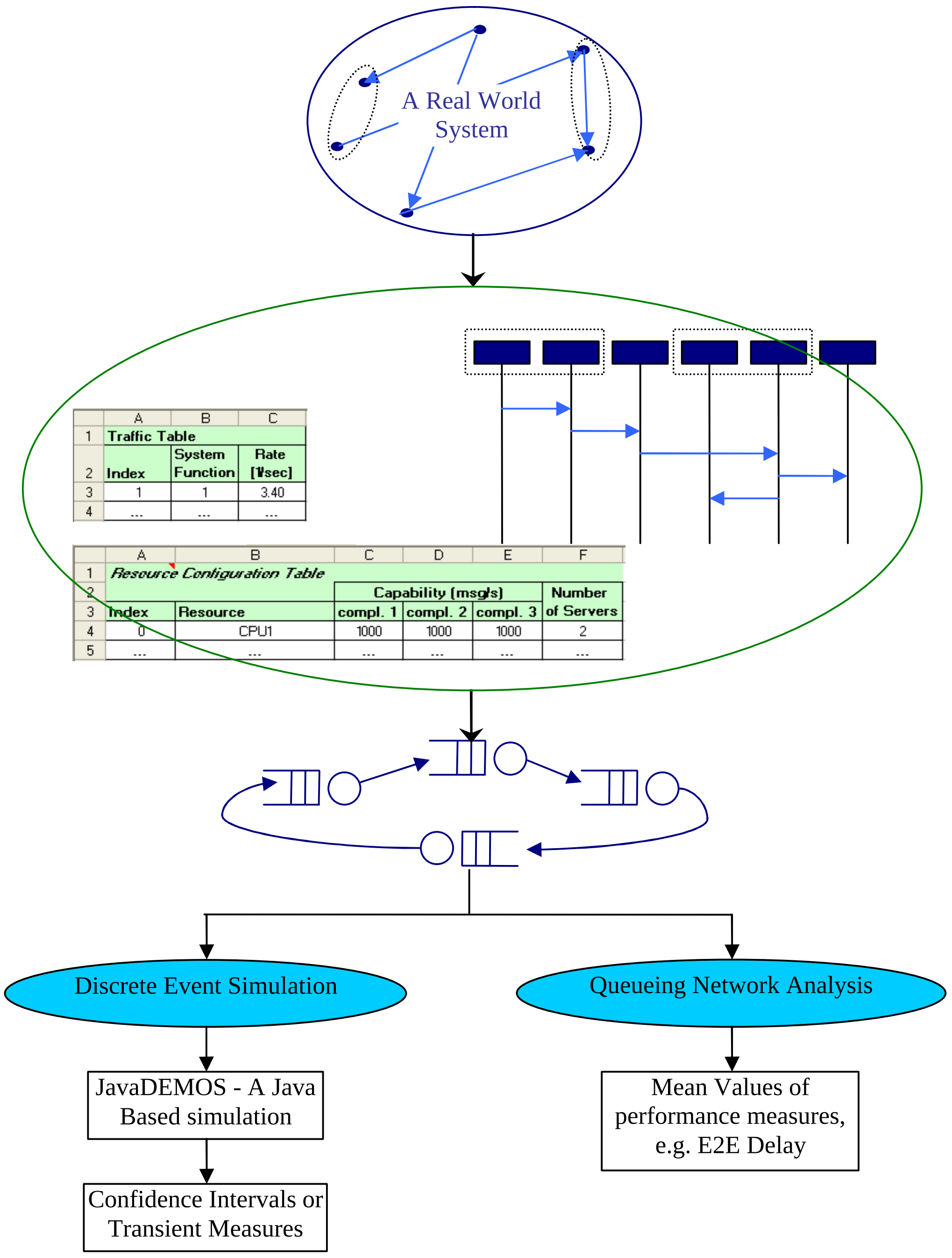
Appendix C: Description and Usage of the MINA Tool

C.1 Description of the Architecture of the MINA Tool

As we discussed before MINA tool calculates the performance measures of distributed systems, e.g. communication systems and computer systems by constructing a queueing network model, which can be solved either by simulation or by queueing network analysis. Figure 40 describes the MINA tool chain. It shows the steps needed to achieve the goal that is to calculate the system performance measures.

The first step is to describe the communication between system components by Message Sequence Charts. System components, e.g. nodes, communicate by sending and receiving messages. In real world systems, e.g. the mobile communication system of Chapter 6, this MSC-based description depends on defining a set of MSCs. Each MSC implements a certain function and communicates with other MSCs to define HMSCs, which describe the whole system behaviour. Inside the node one or more functional entities (FEs) may lie and they are responsible of exchanging messages of an MSC (see Figure 40, the dotted rectangle around the instances of the MSC means that these instances present FEs of the same node). In complex systems also messages of different MSCs may be divided into classes (complexity classes).

The next step is to consider each node as a queueing station (a resource with a server and a queue) and after that associate this MSC or HMSC description with parameters that allow calculating the performance measures of the system. These parameters (as shown in Figure 40) are saved in a text form as an Excel sheets. The first kind of parameters is related to resources and their processing properties like resources speeds and service amounts for different complexity classes. These parameters are used to calculate the mean service rates of different complexity classes at different resources as shown in the "resource table" (see Figure 40). The second kind of parameters imposes a relevant load onto the system. This is specified in the traffic table, which contains the execution rates, i.e. the number of instantiations per time unit for each MSC.


A Real World System
Discrete Event Simulation
Queueing Network Analysis
JavaDEMOS - A Java Based simulation
Confidence Intervals or Transient Measures
Mean Values of performance measures, e.g. E2E Delay

	A	B	C
1	Traffic Table		
2	Index	System Function	Rate [1/sec]
3	1	1	3.40
4	...	...	...

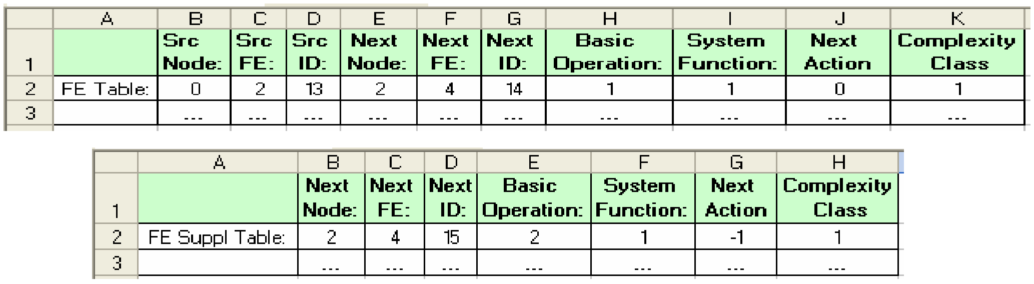
	A	B	C	D	E	F
1	*Resource Configuration Table*					
2			Capability (msg/s)			Number
3	Index	Resource	compl. 1	compl. 2	compl. 3	of Servers
4	0	CPU1	1000	1000	1000	2
5	...	...	...	...	...	...

Figure 40: MINA tool chain

At this step, we have a queueing network model, which can be solved either by discrete event simulation or by queueing network analysis. The queueing network model we use is open and the MINA tool offers some methods for calculating the system performance measures, e.g. the response time for different MSCs and the utilization of the resources, for open queueing networks according to the characteristics of the queueing network. The user of the tool can choose between using Jackson's method or BCMP method or an approximate algorithm, for more details about these methods see 3.6.3, by setting an argument to a certain value. The user of the MINA tool can obtain the performance measures results using the queueing network analysis very fast.

On the other hand, the user of the MINA tool can observe the dynamic behaviour of the performance measures values as well as obtaining confidence intervals of the response time of the different MSCs by using simulation. The simulator is built using JavaDEMOS package and hence all facilities of JavaDEMOS, e.g. obtaining traces and histograms and other features of JavaDEMOS, are available for the user of the MINA tool.

One important point is that the order of messages is important in the case of using the simulation whereas in the case of using the queueing network analysis, the most important is that the number of messages which visits each resource. Another point is that the load, which is described by the MSCs or HMSCs, is converted into an equivalent text form and saved in tables using Microsoft Excel sheets to be easily used by the simulator. To do this two tables are associated to each FE to keep track of all messages sent by this FE as shown in Figure 41.



	A	B	C	D	E	F	G	H	I	J	K
1		Src Node:	Src FE:	Src ID:	Next Node:	Next FE:	Next ID:	Basic Operation:	System Function:	Next Action	Complexity Class
2	FE Table:	0	2	13	2	4	14	1	1	0	1
3		...	...	...	...	...	...	...	...	...	...

	A	B	C	D	E	F	G	H
1		Next Node:	Next FE:	Next ID:	Basic Operation:	System Function:	Next Action	Complexity Class
2	FE Suppl Table:	2	4	15	2	1	-1	1
3		...	...	...	...	...	...	...

Figure 41: The load tables used by the simulator

In the case that the FE sends more than one message, The "FE Suppl Table" is used for keeping track of supplementary messages and the “FE Table” is used for keeping track of the other messages.

C.2 MINA Tool Functionality

MINA uses MSCs to describe the communication between components of distributed systems, e.g. communication systems and computer systems. To evaluate some performance measures, e.g. the response time of the MSCs describing these systems and the utilization of the system components (resources), MINA assigns values to the speed parameters of the resources, the arrival rates of the MSCs and also the service amounts for different classes of messages. Based on this combination of MSC description of the system and the input parameters, MINA builds a queueing network model. MINA enables the user to calculate the performance measures of the system in two ways either by simulation or by queueing networks analysis techniques.

C.3 Necessary Knowledge of the User and System Requirements

To be able to use the MINA tool your system should contain Microsoft Excel, JDK 1.4 (Java Development Kit) or higher and JavaDEMOS package. The user of the MINA tool should know the basics of the message sequence charts language, discrete event simulation, JAVA language and queueing networks algorithms. The user of the MINA should be also familiar with using both JavaDEMOS and Microsoft Excel.

C.4 Preparing the Input

The input as we described before is organized in some tables and saved as a Microsoft Excel sheets. We have five Excel sheets with five tables that should be filled by the user before starting to run the simulator or the analytical queueing network algorithm.

The first sheet (called “CPUSpeed”) describes the speed of each resource and it contains a table of two columns, one is for the resource name and the second is for the resource speed (see Table 30). The second sheet (called “CPUServiceAmount”) describes the service amount assigned to each complexity class at different resources and it contains a table of two or more than two columns, one is for the resource name and the rest of columns are for the service amount of each complexity class (see Table 31). The third sheet (called "CPUServiceRate") describes the service rate assigned to each complexity class at different resources. The value of the service rates in this sheet are calculated by dividing the speed of the resource (in the "CPUSpeed" sheet) by the service amounts (in the "CPUServiceAmount" sheet). The "CPUServiceRate" sheet has another column that describes the number of servers for each resource (see Table 32).

Resource Name	**Speed**
CPU1	10.00
CPU2	1.00
CPU3	60.00
CPU4	100.00

Table 30: An example of the "CPUSpeed sheet"

CPU Name	**Service Amount**		
	CC1	**CC2**	**CC3**
CPU1	1.00	1.00	1.00
CPU2	1.00	2.00	3.00
CPU3	1.00	2.00	36.00
CPU4	1.00	1.00	1.00

Table 31: An example of the "CPUServiceAmount sheet"

CPU Name	**Service Rate**			**Number of Servers**
	CC1	**CC2**	**CC3**	
CPU1	10.00	10.00	10.00	10000
CPU2	1.00	0.50	0.33	4.00
CPU3	60.00	30.00	1.67	2.00
CPU4	100.00	100.00	100.00	20000

Table 32: An example of the "CPUServiceRate sheet"

The fourth sheet (called "msg") describes the message flow from one node to the other (as shown in Figure 41). As we shown before, each node may have some FEs and it may also happen that the FE sends two messages at the same time. According to this structure the "msg" sheet has two tables for each FE. The first table contains data about all messages that are sent by this FE. This table contains columns for the sender of the previous message and the receiver of the current message. Also it contains a column about the complexity class of the current message and the next action. If the value of next action field is -1 then the next message is a normal forwarding message, otherwise the next message is a supplementary message. The second table describes the supplementary messages sent by this FE. The user of the MINA tool should be familiar with this kind of tables to be able to describe his own systems in the same way. The fields of both tables are shown in Figure 41.

The fifth sheet (called “FEtoCPU”) describes the mapping between the FEs and the corresponding resources. It contains a two columns table. The first column has an identification number consists of two digits to refer to the FE. The first digit denotes the node number and the second digit denoted the FE number. The second column contains the resource name.

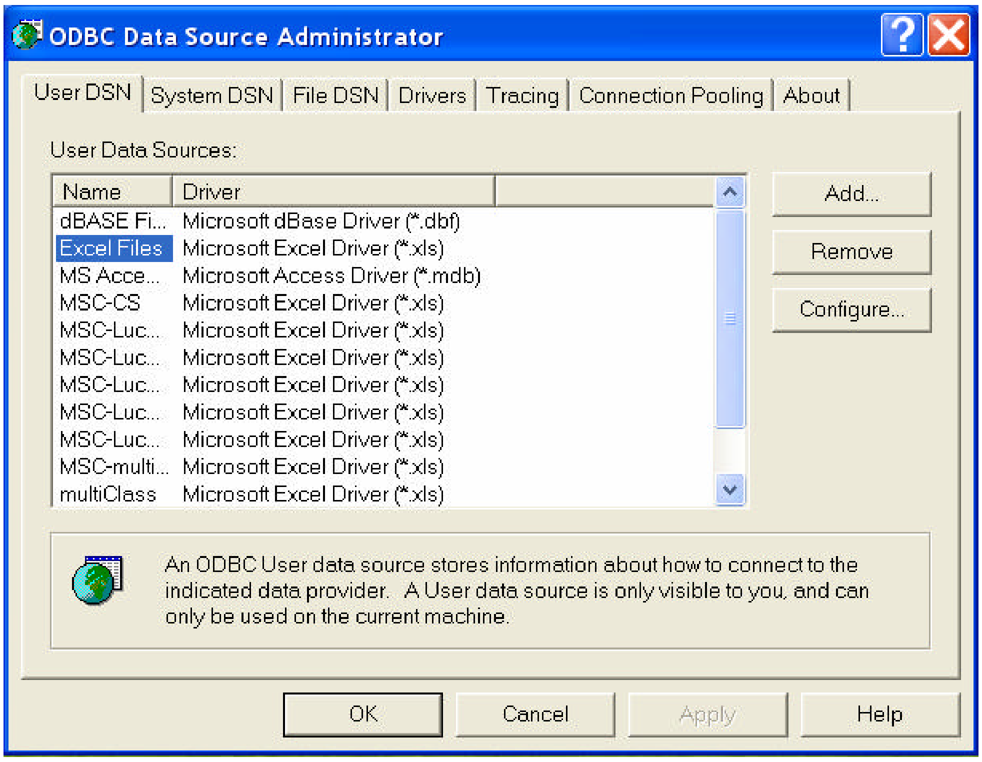
FE Name (Node FE)	CPU Name
0 1	CPU1
0 2	CPU1
1 1	CPU2
...	...

Table 33: An example of the “FEtoCPU sheet”

Note that the SHRINK factor parameters can be changed from the main simulation class “MSCsimulation.class”. The simulation time, type of arrival distribution, type of the service distribution can also be changed from the main simulation class “MSCsimulation.class”.

C.5 Before Starting

The next step before using the MINA tool is to set up your Excel spreadsheet as an ODBC (Open Data Base Connectivity) source. Using JDBC (Java Data Base Connectivity) in conjunction with ODBC enables the user of the MINA tool to deal with the Excel spreadsheets as if they were databases. After creating the Excel spreadsheet with which the user will interact, the user needs to register the spreadsheet as an ODBC Data Source. To do this, the user should open the “Windows Control Panel”. Next, open up “Administrative Tools”. Then, the user should double click on the “Data Sources (ODBC)” icon. In the “User DSN” tab, the user should choose the Excel files option and click Add (see Figure 42).

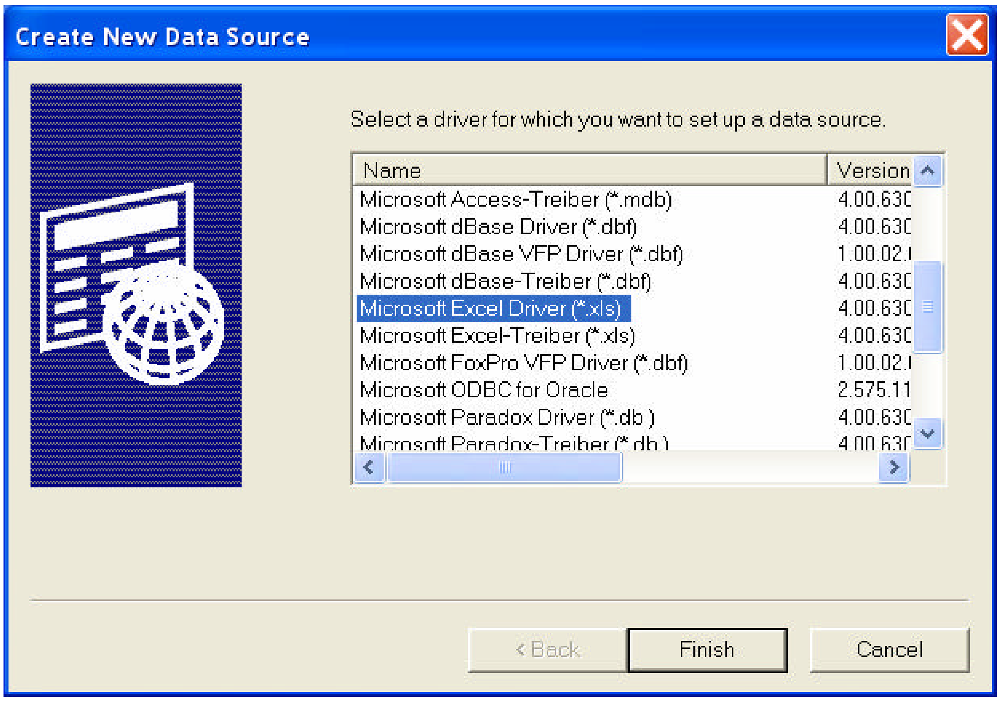


User Data Sources:

Name	Driver
dBASE Fi...	Microsoft dBase Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
MS Acce...	Microsoft Access Driver (*.mdb)
MSC-CS	Microsoft Excel Driver (*.xls)
MSC-Luc...	Microsoft Excel Driver (*.xls)
MSC-Luc...	Microsoft Excel Driver (*.xls)
MSC-Luc...	Microsoft Excel Driver (*.xls)
MSC-Luc...	Microsoft Excel Driver (*.xls)
MSC-Luc...	Microsoft Excel Driver (*.xls)
MSC-multi...	Microsoft Excel Driver (*.xls)
multiClass	Microsoft Excel Driver (*.xls)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

Figure 42: Add an Excel file data source



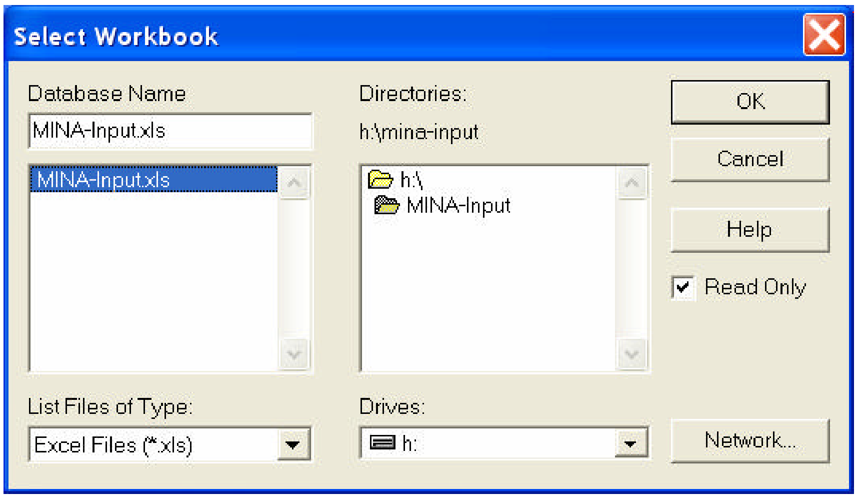
Select a driver for which you want to set up a data source.

Name	Version
Microsoft Access-Treiber (*.mdb)	4.00.63C
Microsoft dBase Driver (*.dbf)	4.00.63C
Microsoft dBase VFP Driver (*.dbf)	1.00.02.(
Microsoft dBase-Treiber (*.dbf)	4.00.63C
Microsoft Excel Driver (*.xls)	4.00.63C
Microsoft Excel-Treiber (*.xls)	4.00.63C
Microsoft FoxPro VFP Driver (*.dbf)	1.00.02.(
Microsoft ODBC for Oracle	2.575.11
Microsoft Paradox Driver (*.db)	4.00.63C
Microsoft Paradox-Treiber (*.db)	4.00.63C

Figure 43: Choose the Microsoft Excel driver

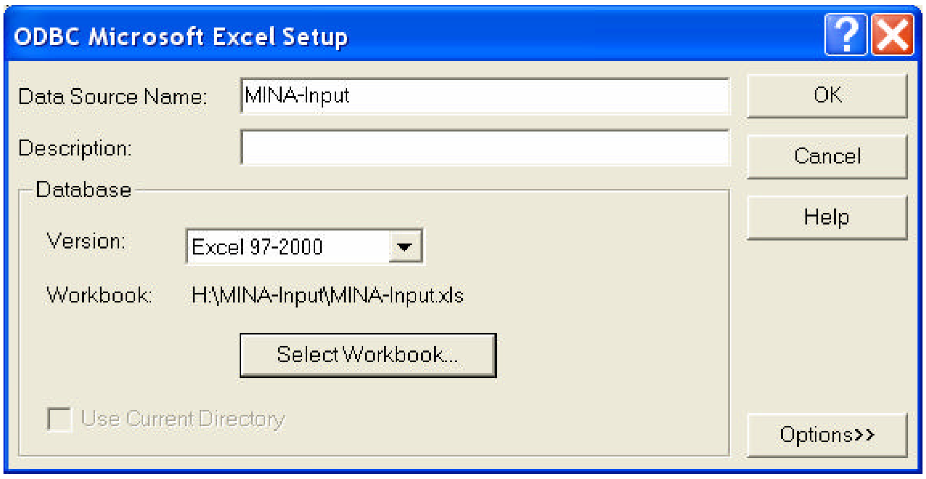
In the subsequent driver selection page, the user should choose the "Microsoft Excel

Driver" option and click Finish (see Figure 43). Next, the user will be presented with a window in which he will select the Excel file that he wants to setup as an ODBC source. The user should choose the “Select Workbook” button (see Figure 44) and choose the spreadsheet he created (MINA-Input.xls). The user should be returned to the “ODBC Microsoft Excel Setup” window. The user should go ahead and name his “Data Source Name” as MINA-Input (see Figure 45). In the “ODBC Data Source Administrator screen”, the user should see the “ODBC Data Source” he just created.



Database Name: MINA-Input.xls
MINA-Input.xls
Directories: h:\mina-input
h:\
MINA-Input
Read Only [✓]
List Files of Type: Excel Files (*.xls)
Drives: h:

Figure 44: Select the workbook you want to setup as a data source



Data Source Name: MINA-Input
Description:
Database
Version: Excel 97-2000
Workbook: H:\MINA-Input\MINA-Input.xls
Use Current Directory

Figure 45: Specifying a name for the data source

The following two statements are used in the code of the class “MSCdata” (the class which is responsible of reading the Excel spreadsheets):

```
static final String DRIVER_NAME = "sun.jdbc.odbc.JdbcOdbcDriver";

static final String DATABASE_URL = "jdbc:odbc:MINA-Input";
```

Note that the driver that is used to set up your Excel spreadsheet as an ODBC source is sun.jdbc.odbc.JdbcOdbcDriver, the JDBC-ODBC bridge driver. The user of the MINA tool does not have to download anything to use it. The driver is built into the JDK. Another important point is that the MINA-Input portion of the string of the database URL you specify, jdbc:odbc:MINA-Input, is the name of the data source you specified earlier. So, if the user of the tool likes to change this name, the name should be changed in both the source code as well as the "Data Source Name". The user of the MINA tool could then use the JDBC-ODBC bridge driver to interact with the spreadsheet using SQL (Structured Query Language). Now, the user of the MINA tool is ready to get the results using either simulation or queueing networks analysis.

C.6 Getting the results

The user of MINA tool is now ready to run the simulator main class "MSCsimulation.class" using the JavaDEMOS package (as described in Appendix A). In this case the user can make use of the GUI of the JavaDEMOS to easily run the simulator and also to follow dynamically the actions done by each entity and also to follow the dynamic behaviour of the resources. In this case all facilities of JavaDEMOS are available for the user, e.g. showing traces, drawing histograms and also showing a complete report of the results. The user should run the simulation for a long enough period to get the steady state results. Figure 39 shows an example of a complete report that can be obtained by the JavaDEMOS.

Figure 39 shows a table of results the used resources the first column header is "Title" which indicates the resource object name. The second is "ResetAt" which indicates the start time of the output evaluation. The third is "Obs" which indicates the number of observations. The next one is "Q-Max" which indicates the maximum queue length. Then, "Q-Now" which indicates the current queue length. The next column head is "Av Length" which indicates the average queue length. The next one is "ZeroWait" which refers to the number of entities which did not have to wait for a resource unit. The "Av Wait" column explains the mean waiting time for resource units. The “Limit” column displays the capacity of the resource. Then the “Min” and “Now” columns display the minimum and the current number of available resources respectively. Finally, the “% Usage” column displays the mean resource usage.

A similar reports are introduced for objects of type Bins, Tallies, Accumulate, Queues, Counts, Histograms. Also another report is introduced for distributions used during the simulation.

The user also can invoke the class “performanceEval.class” in the main class “MSCsimulation.class” to get the analytical results. These results could be obtained in seconds. The user can use one of these methods: Jackson’s method, the BCMP method or the method of decomposition by choosing the appropriate value of the string arguments and call the class with this argument. The user should choose the suitable method that gives approximately identical results compared with the simulation results.

The user of the MINA tool should expect the following results:

- ➢ The response time of each MSC describing the system.
- ➢ The utilization at each resource.
- ➢ In the case of simulation, the confidence interval for the response time of each MSC is also obtained.

Bibliography

1. Allen, O.: Probability, Statistics and Queueing Theory with Computer Science Applications, Academic Press, New York, 2nd. Edition, 1990.
2. Attermeyer, C.; Müller-Clostermann, B.: Modelling of Background Traffic–Concepts and Experimental Evaluation Using OPNET Modeler, Part I: Basic Models, Internal Report, February 2003.
3. Attermeyer, C.; Müller-Clostermann, B.: Modelling of Background Traffic–Concepts and Experimental Evaluation Using OPNET Modeler, Part II: MxRAN-Simulator Experiments using SRINK, Internal Report, June 2003.
4. Ball, P. D.: Introduction to discrete event simulation, in: The 2nd DYCOMANS workshop, Management and Control: Tools in Action, Algarve, Portugal, 367-376, May 15th-17th, 1996.
5. Banks, J.; Carson, J. S.; Nelson, B. N.; Nicol, D.: Discrete-event System Simulation, 3rd. Edition, Prentice-Hall.
6. Balsamo, S.: Product Form Queueing Networks in: Performance Evaluation, Origins and Directions, in: C. Lindemann, G. Haring and M. Reiser, (Eds.), Lectures Notes in Computer Science 1769, Springer, 377-401, 2000.
7. Baskett, F.; Chandy, K.; Muntz, R.; Palacios, F.: Open, Closed, and Mixed Networks of Queues with Different Classes of Customers, Journal of the ACM, 22(2), 248-260, April 1975.
8. Bazan, P.; Bolch, G.; German, R.: WinPEPSY-QNS Performance Evaluation and Prediction System for Queueing Networks, Proceedings of the 11th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'04), Magdeburg, 13.-16.06.2004.

9. Belachev, M.; Shyamasundar, R. K.: MSC+: From Requirement to Prototyped Systems, 13th Euromicro Conference on Real-Time Systems, Technical University of Delft, Delft, The Netherlands, 117-124, June 13th - 15th, 2001.

10. Birtwistle G. M.: DEMOS-a System for Discrete Event Modelling on Simula, Macmillan, 1985, see also http://www.cosc.canterbury.ac.nz/teaching/classes/cosc327/.

11. Birtwistle G. M.: DEMOS Reference Manual, http://www.informatik.uni-essen.de/Lehre/Material/DiskreteSim/DemosRefMan.txt.

12. Bolch, G.: Leistungsbewertung von Rechensystemen. Leitfaden und Monographien der Informatik, B.G.Teubner Verlagsgesellschaft, Stuttgart, 1989.

13. Bolch, G.; Greiner, S.; de Meer, H.; Trivedi, K. S.: Queueing Networks and Markov Chains, Wiley, 1998.

14. Braga, L.; Manione, R.; Renditore, P.: A Formal Description Language for the Modelling and Simulation of Timed Interaction Diagrams. In: Gotzhein and Bredereke (eds.) [25], 245-260, 1996.

15. Chylla, P.: Zur Modellierung und approximativen Leistungsanalyse von Vielteilnehmer-Rechensystemen, Dissertation, Faculty for Mathematics and Computer Science, Technical University Munich, 1986.

16. Diefenbruch, M.; Heck, E.; Hintelmann, J.; Müller-Clostermann, B.: Performance evaluation of SDL systems adjunct by queueing models, In: Braek, R.; Sarma, A. (Eds.): Proc. of SDL-Forum '95, Elsevier, 231-242, 1995.

17. Diefenbruch, M.; Hintelmann, J.; and Müller-Clostermann, B.: The QUEST-Approach for the Performance Evaluation of SDL-Systems, In: Gotzhein and Bredereke (eds.) [25], 229-244, 1996.

18. Diefenbruch, M.; Müller-Clostermann, B.: Queueing SDL: A Language for the Functional and Quantitative Specification of Distributed Systems, Proceedings Workshop on Performance and Time in SDL and MSC, Universität Erlangen-Nürnberg, 103-116, 1998.

19. Faltin, N.; Lambert, L.; Mitschele-Thiel, A.; Slomka F.: PMSC - Performance Message Sequence Chart, Technical Report 10/97, Universität Erlangen-Nürnberg, IMMD VII, Erlangen , 1997.

20. Faltin, N.; Lambert, L.; Mitschele-Thiel, A.; Slomka, F.: An Annotational Extension of Message Sequence Charts to Support Performance Engineering, In: SDL'97: Time for Testing - SDL, MSC and Trends, Evry, France, Eighth SDL Forum, North-Holland, 307-322, 1997.

21. Flüs, C.; Mohamed, H.; Müller-Clostermann, B.: JavaDEMOS: Java-based Discrete Event Simulation. In: MMB-Mitteilungen Nr. 41, 2002, cf. http://www.informatik.unibw-muenchen.de/mmb/mmb41/inhalt.htm .

22. Frangiadakis, N.; Nikolaidis, G.; Schefczik, P.; Wiedemann, A.: MxRAN Functional Architecture Performance Modelling, OPNETWORK 2002 conference, Washington D.C., USA, August 2002.

23. Friedrich-Alexander-Universität Erlangen-Nürnberg, Department of Computer Science 7, Home page: http://www7.informatik.uni-erlangen.de.

24. Gelenbe, E.; Pujolle, G.: Introduction to Queueing Networks, Wiley, Chichester, 1987.

25. Gotzhein, R.; Bredereke, J. (eds.): Formal Description Techniques IX: Theory, application and tools, IFIP TC6 WG6.1 International Conference on Formal Description Techniques IX / Protocol Specification, Testing and Verification XVI, Kaiserslautern, Germany, 1996.

26. Haverkort, B. R.: Performance of Computer Communication Systems: A Model-Based Approach, Wiley & Sons, New York, NY, USA, 1998.

27. Hennessy, J. L.; Patterson, D. A.: Computer Architecture-A Quantitative Approach, Morgan Kaufmann, Amsterdam, Boston, 3rd. Edition, 2003.

28. Herzog, U.: Formal Methods in Performance Evaluation. In: Brinksma, E.; Hermanns, H..; Katoen, J. P. (eds.), Lectures on Formal Methods and Performance Analysis, LNCS 2090, Springer, 2001.

29. Hintelmann, J.: TCP Performance Models based on Formal Specifications in SDL**,** IFIP WG 7.3, Performance 99, Workshop on TCP Modelling, Istanbul, Turkey, 1999.

30. Ingalls, R. G.: Introduction to Simulation, Proceedings of the 2001 Winter Simulation Conference, Peters, B. A.; Smith, J.S.; Medeiros, D.J.; Rohrer, M.W.(Eds.), 2001.

31. IPonAir Home page: http://www.iponair.de.

32. Jackson, J.: Networks of Waiting Lines, Operations Research, 5 (4):518-521, 1957.

33. Jackson, J.: Customershop-Like Queuing Systems, Management Science, 10 (1):131-142, October 1963.

34. Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modelling, John Wiley & Sons, Inc., New York, 1991.

35. JavaDEMOS software on the JavaDEMOS home page: http://www.informatik.uni-essen.de/SysMod/JavaDEMOS/

36. Jonsson, B.; Padilla, G.: An execution Semantics for MSC2000, 10th International SDL Forum Copenhagen, Denmark, Springer Verlag, LNCS 2078, 2001.

37. Kaaranen, H.; Ahtiainen, A.; Laitinen, L.; Naghian, S.; Niemi, V.: UMTS Networks: Architecture, Mobility and Services. John Wiley & Sons, Ltd., Chichester, England, 2001.

38. Kerber, L.: Scenario-based Performance Evaluation of SDL/MSC-Specified Systems. Performance Engineering, LNCS 2047, 185-201, Springer Verlag, 2001.

39. Kippar, J.: Introduction to Discrete Event Simulation, www.cs.tpu.ee/~jaagup/uk/ds/chp1/chapter1.html.

40. Kosiuczenko, P.: Time in Message Sequence Chart, In: Lengauer C., Griebl M., and Gorlatch S. (Eds.), Euro-Par'97 Parallel Processing, Springer LNCS 1300, 562-566, 1997.

41. Krämer, W.; Langenbach-Belz, M.: Approximate Formulae for General Single Systems with Single and Bulk Arrivals, In Proc. 8th Int. Teletrafic Congress (ITC), 235-243, Melbourne, 1976.

42. Kühn, P.: Approximate Analysis of General Queuing Networks by Decomposition, IEEE Transactions on Communication, 27(1), 113-126, January 1979.

43. Lambert, L.: PMSC for Performance Evaluation. in: Mitschele-Thiel, A.; Müller-Clostermann, B.; Reed, R. (eds.); Workshop on Performance and Time in SDL and MSC; February 17-19, University of Erlangen-Nürnberg, 1998.

44. Mauw, S.; Reniers, M. A.: High-Level Message Sequence Charts. In Cavalli and Sarma [20], 291–306, 1997.

45. Menascé, D. and Almeida, V.: Capacity Planning for Web Performance: Metrics, Methods & Benchmarks, Prentice-Hall, 1998.

46. Menascé, D., Almeida, V. and Dowdy, L.: Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems, Prentice-Hall, 1994.

47. Menascé, D.: Component-level Performance Models for the Web (a power point presentation), http://www.cse.msu.edu/~cse807/notes/slides/part7set1presentation.ppt.

48. Meyer, J. F.: Performability: a retrospective and some pointers to the future, Performance Evaluation 14, 139-156, 1992.

49. Mitschele-Thiel, A.: Integrating Performance and Time in SDL and MSC – Current State and Vision, In [54].

50. Mitschele-Thiel, A.: Methodology and tools for the development of high performance parallel systems with SDL/MSC, Proc. Software Engineering for Parallel and Distributed Systems, 1996.

51. Mitschele-Thiel, A.; Langendofer, P.; Henke, R.: Design and optimization of high-performance protocols with the DO-IT toolbox, Proc. of FORTE/PSTV '96.

52. Mitschele-Thiel, A.: Systems Engineering with SDL – Developing Performance Critical Applications, Wiley, 2000.

53. Mitschele-Thiel, A.; Müller-Clostermann, B.: Performance Engineering of SDL/MSC Systems, Computer Networks 31, 1801-1815, 1999.

54. Mitschele-Thiel, A.; Müller-Clostermann, B.; Reed, R. (eds.): Proceedings Workshop on Performance and Time in SDL and MSC. Report IMMD VII-1/98, University of Erlangen, 1998.

55. Mitschele-Thiel, A.; Schefczik, P.; Wiedemann, A.: Architecture Optimization of 3rd Generation Wireless Systems based on Use Cases, Applied Telecommunication Symposium (ATS) 2004, Arlington, Virginia, USA, April 2004.

56. Mohamed, H.: Discrete Event Simulation Using JavaDEMOS, Technical Report, University of Essen, 2001. home page: http://www.informatik.uni-essen.de/SysMod/publikationen/index.html.

57. Mohamed, H.; Müller-Clostermann, B.: Towards an Efficient Performance Evaluation of Communication Systems described by Message Sequence Charts, in: H. König, M. Heiner, and A. Wolisz (Eds.), Formal Techniques for Networked and Distributed Systems - FORTE 2003, Springer Lecture Notes in Computer Science (LNCS 2767), 413-429

58. MSC Standard, ITU-T. Recommendation Z.120, Message Sequence Charts. Geneva, 1999.

59. MSC-2000 MESSAGE SEQUENCE CHART (MSC), (revised in 2001), SDL Forum Version of Z.120 (11/99) rev. 1(14/11/01).

60. Müller-Clostermann, B.: Complexity Classes of Messages and Service Time Calculation, TD132-100, February 2003.

61. Müller-Clostermann, B.; Rathgeb, E.; Wiedemann, A.: MxRAN Simulator Final Report, 2004.

62. Padilla, G.: An execution Semantics for MSC2000. August 2000, see http://citeseer.nj.nec.com/padilla00execution.html.

63. Pan, R.; Prabhakar, B.; Psounis, K.; and Wischik, D.: SHRINK: A Method for Scaleable Performance Prediction and Efficient Network Simulation, In Proceedings of IEEE INFOCOM, San Francisco, USA, March 30 - April 3, 2003.

64. Pan, R.; Prabhakar, B.; Psounis, K.; Sharma, M.: A Study of the Applicability of a Scaling Hypothesis, Proceedings of the 4th Asia Control Conference, Singapore, 2002.

65. Trick, M. A.: An Introduction to Simulation, http://mat.gsia.cmu.edu/simul/.

66. Psounis, K.; Pan, R.; Prabhakar, B.; Wischik, D.: The Scaling Hypothesis: Simplifying the Prediction of Network Performance using Scaled-down Simulations, Proceedings of SIGCOMM HotNets-I, 2002.

67. Pujolle, G. and Ai., W.: A Solution for multiserver and multiclass open queueing networks. INFOR, 24(3):221-230, 1986.

68. QUEST (The Queuing SDL Tool) Concept home page, http://www.cs.uni-essen.de/SysMod/QUEST/doc/concept.pdf.

69. QUEST User Manual, www.cs.uni-essen.de/SysMod/QUEST/doc/manual-1.3.pdf.

70. Schaffer, C.: MSC/RT: A Real-Time Extension to Message Sequence Charts (MSCs), Technical Report TR140-96, Johannes Kepler Universität Linz, Institut für Systemwissenschaften, 1996.

71. Schefczik, P.; Mitschele-Thiel, A.; Soellner, M.; Speltacker, W.: On MSC-Based Performance Simulation, the Third International Workshop on Software and Performance (WOSP 2002), Rome, Italy, July 2002.

72. Schieferdecker, I.; Rennoch A.; Mertens, O.: Timed MSCs - an Extension to MSC96. In Wolisz et al. [79], 165-174, 1997.

73. Smith, C. U.; Williams, L. G.: Performance Engineering Evaluation of Object-Oriented Systems with SPEED, In: R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, (Eds.), Computer performance evaluation, Springer, LNCS 1245, 1997.

74. Whitt, W.: Performance of the Queueing Network Analyzer, Bell System Technical Journal, 62(9), 2817-2843, November, 1983.

75. Whitt, W.: The Queueing Network Analyzer, Bell System Technical Journal, 62(9), 2779-2815, November 1983.

76. Wiedemann, A.; Schefczik, P.; Nikolaidis G.: Simulation Methodology for Assessing MxRAN Architecture Performance, European Simulation Multiconference (ESM) 2003, Nottingham, UK, June 2003.

77. Willig, A.: Performance Evaluation Techniques, summer 2004, www.mcs.csuhayward.edu/~morgan/notes_CS4594/pdfs/performance_analysis.pdf.

78. WinPEPSY-QNS (the **P**erformance **E**valuation and **P**rediction **SY**stem for **Q**ueueing **N**etwork**S**), http://www7.informatik.uni-erlangen.de/~prbazan/pepsy/download.shtml.

79. Wolisz, A., Schieferdecker, I. and Rennoch, A. (eds.): Formale Beschreibungstechniken für verteilte Systeme, GMD-Studie Nr. 315, Berlin, Germany, GI/ITG, GMD-Forschungszentrum, 1997.