Hasso–Plattner–Institut für Softwaresystemtechnik
an der Universität Potsdam

Quantitative Modeling and Analysis with FMC-QE

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
in der Wissenschaftsdisziplin "Software Engineering"

eingereicht an der
Mathematisch Naturwissenschaftlichen Fakultät
der Universität Potsdam

von
Stephan Kluth

Wolfsburg, den 18.07.2011

Published online at the
Institutional Repository of the University of Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2011/5298/
URN urn:nbn:de:kobv:517-opus-52987
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-52987

Gutachter:

Prof. Dr.-Ing. Werner Zorn
Prof. Dr. Dr. h.c. Otto Spaniol
Prof. Dr. Paul Müller

Abstract

The modeling and evaluation calculus FMC-QE, the Fundamental Modeling Concepts for Quantitative Evaluation [143], extends the Fundamental Modeling Concepts (FMC) for performance modeling and prediction. In this new methodology, the hierarchical service requests are in the main focus, because they are the origin of every service provisioning process. Similar to physics, these service requests are a tuple of value and unit, which enables hierarchical service request transformations at the hierarchical borders and therefore the hierarchical modeling. Through reducing the model complexity of the models by decomposing the system in different hierarchical views, the distinction between operational and control states and the calculation of the performance values on the assumption of the steady state, FMC-QE has a scalable applicability on complex systems.

According to FMC, the system is modeled in a 3-dimensional hierarchical representation space, where system performance parameters are described in three arbitrarily fine-grained hierarchical bipartite diagrams. The hierarchical service request structures are modeled in Entity Relationship Diagrams. The static server structures, divided into logical and real servers, are described as Block Diagrams. The dynamic behavior and the control structures are specified as Petri Nets, more precisely Colored Time Augmented Petri Nets. From the structures and parameters of the performance model, a hierarchical set of equations is derived. The calculation of the performance values is done on the assumption of stationary processes and is based on fundamental laws of the performance analysis: Little's Law and the Forced Traffic Flow Law. Little's Law is used within the different hierarchical levels (horizontal) and the Forced Traffic Flow Law is the key to the dependencies among the hierarchical levels (vertical). This calculation is suitable for complex models and allows a fast (re-)calculation of different performance scenarios in order to support development and configuration decisions.

Within the Research Group Zorn at the Hasso Plattner Institute, the work is embedded in a broader research in the development of FMC-QE. While this work is concentrated on the theoretical background, description and definition of the methodology as well as the extension and validation of the applicability, other topics are in the development of an FMC-QE modeling and evaluation tool and the usage of FMC-QE in the design of an adaptive transport layer in order to fulfill Quality of Service and Service Level Agreements in volatile service based environments. Especially the close collaboration the development of the FMC-QE Tool promotes the validation of FMC-QE.

This thesis contains a state-of-the-art, the description of FMC-QE as well as extensions of FMC-QE in representative general models and case studies. In the state-of-the-art part of the thesis in chapter 2, an overview on existing Queueing Theory and Time Augmented Petri Net models and other quantitative modeling and evaluation languages and methodologies is given. Also other hierarchical quantitative modeling frameworks will be considered. The description of FMC-QE in chapter 3 consists of a summary of the foundations of FMC-QE, basic definitions, the graphical notations, the FMC-QE Calculus and the modeling of open queueing networks

as an introductory example. The extensions of FMC-QE in chapter 4 consist of the integration of the summation method in order to support the handling of closed networks and the modeling of multiclass and semaphore scenarios. Furthermore, FMC-QE is compared to other performance modeling and evaluation approaches. In the case study part in chapter 5, proof-of-concept examples, like the modeling of a service based search portal, a service based SAP NetWeaver application and the Axis2 Web service framework will be provided. Finally, conclusions are given by a summary of contributions and an outlook on future work in chapter 6.

Zusammenfassung

FMC-QE (Fundamental Modeling Concepts for Quantitative Evaluation [143]) ist eine auf FMC, den Fundamental Modeling Concepts, basierende Methodik zur Modellierung des Leistungsverhaltens von Systemen mit einem dazugehörenden Kalkül zur Erstellung von Leistungsvorhersagen wie Antwortzeiten und Durchsatz. In dieser neuen Methodik steht die Modellierung der hierarchischen Bedienanforderungen im Mittelpunkt, da sie der Ursprung aller dienstbasierenden Systeme sind. Wie in der Physik sind in FMC-QE die Bedienanforderungen Tupel aus Wert und Einheit, um Auftragstransformationen an Hierarchiegrenzen zu ermöglichen. Da die Komplexität durch eine Dekomposition in mehreren Sichten und in verschiedene hierarchische Schichten, die Unterscheidung von Operations- und Kontrollzuständen, sowie dazugehörige Berechungen unter Annahme der Stationarität reduziert wird, skaliert die Anwendbarkeit von FMC-QE auf komplexe Systeme.

Gemäß FMC wird das zu modellierende System in einem 3-dimensionalen hierarchischen Beschreibungsraum dargestellt. Die quantitativen Kenngrößen der Systeme werden in drei beliebig frei-granularen hierarchischen bi-partiten Graphen beschrieben. Die hierarchische Struktur der Bedienanforderungen wird in Entity Relationship Diagrammen beschrieben. Die statischen Bedienerstrukturen, unterteilt in logische und reale Bediener, sind in Aufbaudiagrammen erläutert. Außerdem werden Petri Netze, genauer Farbige Zeit-behaftete Petri Netze, dazu verwendet, die dynamischen Abläufe, sowie die Kontrollflüsse im System zu beschreiben. Anschließend wird eine Menge von hierarchischen Gleichungen von der Struktur und den Parametern des Modells abgeleitet. Diese Gleichungen, die auf dem stationären Zustand des Systems beruhen, basieren auf den beiden Fundamental Gesetzen der Leistungsanalyse, dem Gesetz von Little und dem Verkehrsflussgesetz. Das Gesetz von Little definiert hierbei Beziehungen innerhalb einer hierarchischen Schicht (horizontal) und das Verkehrsflussgesetz wiederum Beziehungen zwischen hierarchischen Schichten (vertikal). Die Berechungen erlauben Leistungsvorhersagen für komplexe Systeme durch eine effiziente (Neu-)Berechnung von Leistungsgrößen für eine große Auswahl von System- und Lastkonfigurationen.

Innerhalb der Forschungsgruppe von Prof. Dr.-Ing Werner Zorn am Hasso Plattner Institut an der Universität Potsdam ist die vorliegende Arbeit in einen größeren Forschungskontext im Bereich FMC-QE eingebettet. Während hier ein Fokus auf dem theoretischen Hintergrund, der Beschreibung und der Definition der Methodik als auch der Anwendbarkeit und Erweiterung gelegt wurde, sind andere Arbeiten auf dem Gebiet der Entwicklung einer Anwendung zur Modellierung und Evaluierung von Systemen mit FMC-QE bzw. der Verwendung von FMC-QE zur Entwicklung einer adaptiven Transportschicht zur Einhaltung von Dienstgüten (Quality of Service) und Dienstvereinbarungen (Service Level Agreements) in volatilen dienstbasierten Systemen beheimatet. Speziell die Kooperation im Bereich der Anwendungsentwicklung führte die Entwicklung von FMC-QE in dieser Arbeit im Bereich Validierung voran.

Diese Arbeit umfasst einen Einblick in den Stand der Technik, die Beschreibung von FMC-QE sowie die Weiterentwicklung von FMC-QE in repräsentativen allgemeinen Modellen und Fall-

studien. Das Kapitel 2: Stand der Technik gibt einen Überblick über die Warteschlangentheorie, Zeit-behaftete Petri Netze, weitere Leistungsbeschreibungs- und Leistungsvorhersagungstechniken sowie die Verwendung von Hierarchien in Leistungsbeschreibungstechniken. Die Beschreibung von FMC-QE in Kapitel 3 enthält die Erläuterung der Grundlagen von FMC-QE, die Beschreibung einiger Grundannahmen, der graphischen Notation, dem mathematischen Modell und einem erläuternden Beispiel. In Kapitel 4: Erweiterungen von FMC-QE wird die Behandlung weiterer allgemeiner Modelle, wie die Modellklasse von geschlossenen Netzen, Synchronisierung und Mehrklassen-Modelle beschrieben. Außerdem wird FMC-QE mit dem Stand der Technik verglichen. In Kapitel 5 werden Machbarkeitsstudien, wie die Modellierung eines dienstbasierten Suchportals, einer dienst-basierten SAP-NetWeaver Anwendung und des Axis2 Web Service Frameworks, beschrieben. Schließlich werden in Kapitel 6 eine Zusammenfassung und ein Ausblick gegeben.

Acknowledgements

First of all, I would like to gratefully thank Prof. Dr.-Ing. Werner Zorn for the opportunity to work in his research group. Through his ideas of FMC-QE [137–146] and a lot of fruitful discussions and support, this thesis was possible.

I would also like to thank the other reviewers for the time and effort they spent to supervise this thesis.

Additionally, I would like to thank the HPI Research School for the support and the constant pressure especially through the biannually retreats with it's demanding reports and presentations. A special thank in this area goes to Prof. Dr. Andreas Polze, the head of the research school. I would also thank the other members of the Research School for the cooperations and discussions. Especially I would like to thank Michael Schöbel for discussions on mathematical questions.

Another thank goes to my colleagues Tomasz Porzucek, Flavius Copaciu, Rami-Habib Eid-Sabbagh, Dominic Wist, Dr.-Ing. Ralf Wollowski, Sebastian Kuhle, Nanjun Li and Raveendra Babu Darsi in the research group for their support as well as the FMC research group especially Dr.-Ing. Peter Tabeling and the Operating Systems and Middleware research group especially Dr. Martin von Löwis.

I would also like to thank the co-authors of my papers for their cooperation. Here I especially would like to thank Marcel Seelig and Mathias Fritzsche.

Another thank goes to the administration of the Hasso Plattner Institute and Hasso Plattner himself for providing the environment for the research. Here a special thank goes to Prof. Dr. Christoph Meinel, Annett Seidler, Sabine Wagner, Ilona Pamperin, Ralf Gruner and Jens Luef.

Last but not least I would like to thank my parents, family and friends for their support. Especially I would like to thank Britta Knüppel for the careful proofreading of my thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the area of mathematical-analytic performance modeling and analysis there are essentially two methodical approaches: Queueing Theory and Time Augmented Petri Nets, whereas both approaches have their assets and drawbacks. The advantages of the Queueing Theory are the matured analysis techniques and the good computability. A disadvantage is limited power in modeling of complex systems, as for example systems are only modeled from the perspective of the server structures and therefore control flows are often neglected. Time Augmented Petri Nets are more powerful in concerns of modeling concurrent processes and control flows. But here, with rising complexity, the problem of state space explosion arises, which could set a border for the practical applicability. As in Queueing Theory, in Time Augmented Petri Nets only one view of the system, here the dynamic view, including the control flows, is modeled. The server structures behind are often neglected, which could cause problems, when shared resources or special queueing or scheduling strategies should be considered.

The modeling and evaluation calculus FMC-QE, the Fundamental Modeling Concepts for Quantitative Evaluation, extends the Fundamental Modeling Concepts (FMC) by aspects of performance modeling and performance prediction. In this new methodology the hierarchical service requests are in the main focus, because they are the origin of every service provisioning process. Similar to physics, the service requests are a tuple of value and unit, in order to provide a service request transformation at the borders of the different hierarchical levels. Through reducing the complexity of the models by decomposing the system in different hierarchical views, the distinction between operational and control states and the calculation of the performance values on the assumption of the steady state, FMC-QE has a scalable applicability on complex systems.

According to FMC, the modeled system is represented in a 3-dimensional hierarchical representation space. The system performance parameters are described in three arbitrarily fine-grained hierarchical bipartite diagrams. The hierarchical service request structures are modeled in Entity Relationship Diagrams. The static server structures, divided into logical and multiplexer servers, are described in Block Diagrams. The dynamic behavior and the control structures are specified in Petri Nets, more precisely Colored Time Augmented Petri Nets. From the structures and parameters of the performance model, a hierarchical set of equations is derived. The calculation of the performance values in this hierarchical set of equations is done on the assumption of stationary processes and is based on fundamental laws of the performance analysis: Little's Law and the Forced Traffic Flow Law. Little's Law is used within the different hierarchical levels (horizontal) and the Forced Traffic Flow Law is the key to the connections among the hierarchical levels (vertical). This calculation is suitable for complex models and

allows a fast calculation of different performance scenarios in order to support system development and configuration decisions.

In this thesis FMC-QE is described, extended and applied to different representative examples as well as compared to other existing performance modeling and evaluation frameworks. The thesis is structured as follows:

In chapter 2 the State of the Art of performance modeling and evaluation from the viewpoint of FMC-QE is explained. This includes an overview on the Queueing Theory and an explanation of quantitative measurements, fundamental laws and formulas as well as algorithms for the prediction of performance values for queueing servers and queueing networks in the Queueing Theory. This chapter gives also a resume on Time Augmented Petri Nets, describes some classifications and outlines important types of Time Augmented Petri Nets. While hierarchies and the hierarchical modeling are the key to complexity, other hierarchical quantitative modeling and aggregation methods are also discussed. This includes decomposability, the adaption of Norton's Theorem to Queueing Theory, formal hierarchies, aggregation and hierarchies in Time Augmented Petri Nets and the Forced Traffic Flow Law. In addition to this, Layered Queueing Networks (LQN) are summarized, as a special quantitative hierarchical modeling methodology.

The main concepts of FMC-QE are described in chapter 3. This starts with a description of the basis of FMC-QE, the Fundamental Modeling Concepts (FMC) and the Fundamental Modeling Concepts extended for Communication Systems (FMC-eCS). After that, basic definitions of the service request, the hierarchical modeling as well as performance parameters and values are given. Then, the 3-dimensional graphical representations divided in service request structures, static (server) structures as well as dynamic behavior and control flow are described. The mathematical calculus and the computation of the performance predictions in the Tableau are also specified. An Open Queueing Network is modeled as an example for the modeling and transformations as well as the calculations of FMC-QE. The development of an FMC-QE Tool for the quantitative modeling and evaluation is in the focus of another PhD. student (Tomasz Porzucek) in the research group and is shortly referenced in this chapter.

In chapter 4 FMC-QE is used to model and evaluate selected types of problems, to provide extensions to the core methodology in order to to solve these problems. In the first part sample Closed Queueing Networks are modeled and the corresponding performance values are derived. This includes an integration of the summation method [17] into FMC-QE in order to extend the range of applications to closed models through the summation approximation method. The handling of multiclass scenarios is also described in this chapter. Then, the semaphore synchronization, a classical Time Augmented Petri Net problem, is addressed. This part describes and compares an approximation for the performance prediction of semaphore synchronization problems. Furthermore, FMC-QE is compared to other performance modeling and prediction approaches.

Chapter 5 provides case studies, modeled and evaluated. The HPI Search Portal illustrates the applicability to larger systems. The second example, the modeling of a service based system inside an SAP NetWeaver / SAP WebAS environment[1], focuses on the comparison of the performance predictions of a performance model, a simulation and the corresponding measured values in the real system. The third example, the modeling of interacting hierarchical Proto-

[1]SAP AG, SAP NetWeaver, Website: http://www.sap.com/germany/plattform/netweaver/index.epx, August 2009

col Stacks in Apache Axis2[2], is a Proof-of-Concept for the hierarchical modeling with the key aspects of multiplex and synchronization in the calculations.

Chapter 6 summarizes the main contributions and the author proposes possible future work.

[2]Apache Software Foundation, Apache Axis2 Architecture Guide, Website: http://ws.apache.org/axis2/1_3/Axis2ArchitectureGuide.html, August 2007

Chapter 2

State of the Art

The following chapter describes the State of the Art of performance modeling and performance predictions from the viewpoint of FMC-QE. This chapter is given to provide an overview of other quantitative modeling and evaluation approaches and to form the basis for further chapters.

The chapter is structured as follows: Section 2.1 provides an overview of the main mathematical background of FMC-QE, the Queueing Theory, especially the fundamental laws Little's Law and the Forced Traffic Flow Law, the calculation of the different servers and algorithms and theorems for the calculation of the performance values of Queueing Networks. In addition to the Petri Nets used in FMC to model the dynamic behavior of systems, the Time Augmented Petri Nets influenced the development of the FMC-QE Petri Nets and are therefore described in section 2.2. The hierarchical modeling is the key to complex systems and therefore of central concern in FMC-QE. Section 2.3 gives an overview of other quantitative hierarchical and aggregation modeling techniques. As a special hierarchical quantitative modeling technique, Layered Queueing Networks are also briefly explained in this section.

2.1 Queueing Theory

The Queueing Theory forms the main mathematical basis for FMC-QE. Therefore this section provides an overview of the most important measures, laws, theorems and algorithms. Important quantitative measures are defined in subsection 2.1.1. After that, two fundamental laws of the Queueing Theory, Little's Law [98] and the Forced Traffic Flow Law [43, 95] are introduced. In order to define and calculate the quantitative measures of single stations or servers in a Queueing Network, the Kendall Notation for server stations [83] and equilibriums for general systems are described in 2.1.3. This is further enriched by formulas for different special types of queueing stations in the appendix A. In the sections 2.1.4 to 2.1.7, important theorems and algorithms for the estimation of performance values for whole Queueing Networks are explained. This includes Jackson's Theorem for Open Queueing Networks [77, 78], the Theorem of Gordon and Newell [64] for Closed Queueing Networks, the networks of Baskett, Chandy, Muntz and Palacios (BCMP-Networks) [8] and the Mean Value Analysis (MVA) [118].

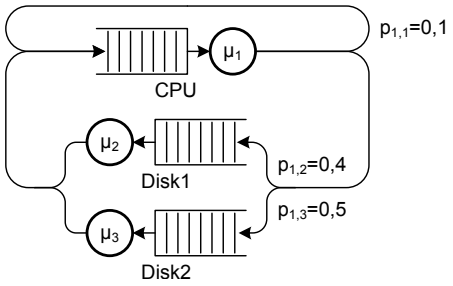


Figure 2.1: Sample Queueing Net [68]

Figure 2.1 provides a small Queueing network example in order to give an impression of the way of modeling in Queueing Theory as a graph of servers, queues and links with routing probabilities. This example shows a Closed Queueing Network with three servers (*CPU*, *Disk1* and *Disk2*) and queues in front of the servers. The service requests are processed at the *CPU*, then routed to *Disk1* with a probability of 40%, routed to *Disk2* with a probability of 50% or recalculated with a probability of 10%. After the service requests have passed one of the disks they are routed back to the CPU. This example will be later calculated using the Theorem of Gordon and Newell in section 2.1.5 and the Mean Value Analysis in section 2.1.7 as well as in FMC-QE in section 4.1.

2.1.1 Quantitative Measures

In the Queueing Theory a number of quantitative measures are defined. The most important ones are illustrated in figure 2.2 and described as:

Arrival Rate λ The arrival rate denotes the mean rate of service requests arriving at a service station.

Service Time X The service time is the time a server needs to process a service request.

Multiplicity m The parameter multiplicity defines the number of parallel servers in a server station. It is possible, that $m = \infty$. In this case every service request has a dedicated server. An example could be the user in a client server scenario, where every user-input request has a dedicated user.

Service Rate μ The service rate is a parameter for the mean maximal number of service requests a server could process in a given time period.

Departure Rate D The departure rate (throughput) denotes the rate, fulfilled service requests (service responses) leave the server.

Number of Service Requests n The mean number of service requests in a service station is denoted by n. It is the sum of the service requests in service n_s and the queued service requests n_q.

Waiting Time W The waiting time (queueing time) denotes the mean time a service request is queued in a service station.

Utilization ρ The utilization denotes the fraction of time the server is processing service requests (busy).

Response Time R The response time is defined as the time interval between the arrival of the service request and the corresponding departure of the service response at a service station.

Queue Size K The queue size defines the maximum number of service requests inside the service station (queued + in service).

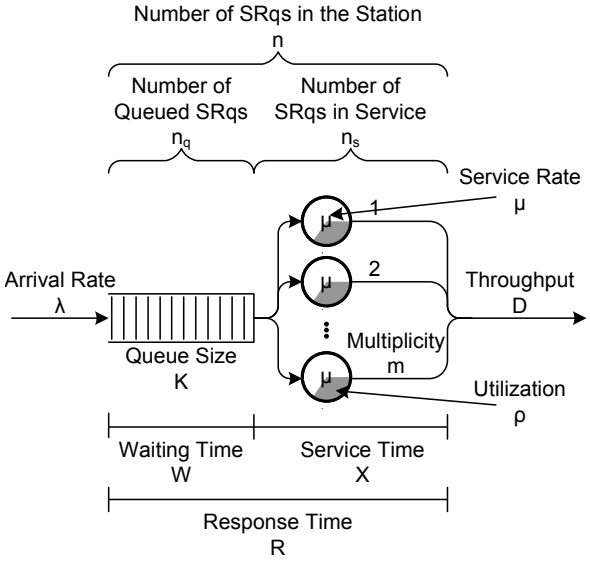


Figure 2.2: Performance Parameters and Values

In general usage, the term performance is also often used when quantitative measures are considered. In a more specific usage, performance would be a measure like throughput in terms of service requests per time unit or response time and not measures like the service time or the multiplicity. But as said, in general use service time could also be defined as a performance parameter which then is also the case in this thesis. Furthermore, there is a distinction between performance parameters and performance values, while performance parameters are an input in a the calculation and performance values are the results. In the further parts the relations between the different measures are described.

2.1.2 Fundamental Laws

Little's Law and the Forced Traffic Flow Law are two fundamental Queueing Theory laws.

Little's Law

Little's Law [98] is one of the fundamental laws in the Queueing Theory. As described in [125], the law defines the dependence between the mean number of jobs in a system n, the mean arrival rate λ and the mean response time R.

$$n = \lambda * R \tag{2.1}$$

Little's Law is illustrated in figure 2.3. Little's Law is a black box law which includes measures of the servers and the queues inside the black box.

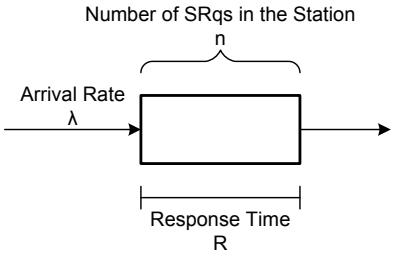


Figure 2.3: Little's Law

Forced Traffic Flow Law

The Forced Traffic Flow Law [43, 68, 79, 95] defines the transformation of a global arrival rate λ into a specific arrival rate λ_i using a traffic flow coefficient v_i as a law, defining relations for the network.

$$\lambda_i = v_i * \lambda \tag{2.2}$$

A more detailed discussion on the Forced Traffic Flow Law can be found in section 2.3.5 and 3.4.1 and is therefore omitted here.

2.1.3 Server Performance Values

The Queueing Theory distinguishes between a broad range of different server types with different queueing strategies, service time distributions and other parameters. The Kendall-Notation was established in order to define and classify these different types. This notation is described in this subsection. Also some general formulas on the server level are described. There are a lot of equilibriums for different Kendall-Servers in literature. In the different sources, the different variable identifiers are slightly different (sometimes conflicting) and information for one type of server is sometimes scattered over several sources. Therefore, the tables A.1 to A.9 in the appendix A consolidate and recapitulate these equilibriums using one consistent nomenclature, without repeating the derivations and proofs.

Kendall-Notation

The Kendall-Notation is the de facto standard for the description of elementary queueing systems. The basics of this notation are defined by David George Kendall in 1953 [83]. The notation defines a Queueing Station and the corresponding arrival process by a series of symbols ($A^x/B/m/K/C$), which are defined as [67]:

A Interarrival-time distribution, where x defines a group arrival process [125]

B Service-time distribution

m Number of parallel servers

K Capacity restrictions

C Queueing discipline

A and B are further specified by the following distribution notations [18, 67, 125]:

M Markov - exponential distribution (Poisson Process)

D deterministic distribution (constant interarrival or service time)

GI General Independent, General distribution with independent interarrival or service times

$\mathbf{E}_k$ Erlang-k-distribution

$\mathbf{H}_k$ Hyperexponentional distribution with k phases

The number of servers m denotes the number of parallel servers at the station and is defined in the range from 1 to ∞.

The parameter K defines the queue size including the places for the service requests in service (m). Often this parameter is omitted, if $K = 1$ or $K = \infty$

The standard queueing discipline C is FIFO (first in, first out resp. FCFS first come, first served) and is therefore often omitted. But other possible disciplines, then defined, are amongst others [67]:

LIFO Last In, First Out resp. LCFS Last Come, First Served

RSS Random Selection for Service

PR Priority Based

GD General Discipline

In special cases an additional $/M$ is defined, which defines the overall fixed population of service requests (or customers) each with an "arriving" parameter λ [88]. The corresponding formulas for this case are provided in table A.8.

General Systems

While in the tables A.1 to A.9 in appendix A different formulas for the calculation of server performance values are defined for the different server types, some formulas are defined for general systems and are therefore true for all of the systems.

The utilization ρ is defined as the fraction of time, a server is busy and could be calculated as the quotient of the arrival rate λ and the service rate μ (where the service rate μ is multiplicative inverse of the service time X: $\mu = \frac{1}{X}$) [88]:

$$\rho = \frac{\lambda}{\mu} \qquad (G/G/1) \tag{2.3}$$

For queueing stations with multiple servers m, this definition is the same, extended by the work capacity of the system [88]:

$$\rho = \frac{\lambda}{m\mu} \qquad (G/G/m) \tag{2.4}$$

For systems, where the the maximum service rate is not independent of the system state, ρ is defined as traffic intensity [88].

The response time R is defined as the sum of waiting time W and the service time X [88]:

$$R = W + X \tag{2.5}$$

Through Little's Law [98] the mean number of service requests in the system are defined as [88]:

$$n = \lambda R, \tag{2.6}$$

where in systems, where the the maximum service rate is not independent of the system state, the arrival rate λ is substituted by the effective arrival rate λ_{eff}.

The mean number of queued service requests is also defined through Little's law [98] as [88]:

$$n_q = \lambda W \tag{2.7}$$

or as the difference from the mean number of service requests in the system n and the mean number of service requests in service n_s (where $n_s = m\rho$, if the maximum service rate is independent of the system state) [88]:

$$n_q = n - n_s. \tag{2.8}$$

Therefore, Little's Law is valid for the whole station or system ($n = \lambda R$) as well as the server ($n_s = m\rho = \lambda X$) and the queue ($n_q = \lambda W$).

2.1.4 Open Queueing Networks - Jackson's Theorem

The results of Jackson [77, 78] have been considered as a break through in the analysis of Queueing Networks [18], because he developed Product Form Solutions for open networks. Through Product Form Solutions, the complexity of the computation of the overall steady-state probabilities and the global performance values could be reduced, as the steady-state probability of the network can be computed as the product of the steady-state probabilities of every node.

Jackson's Theorem [77]: If in an open network the ergodicity ($\lambda_i < \mu_i m_i$) holds for every node ($i = 1, ..N$), then the steady-state probability of the network can be computed as the product of the steady-state probabilities of every node:

$$p\left(k_1, .., k_N\right) = p_1\left(k_1\right) * .. * p_N\left(k_N\right) \tag{2.9}$$

The Jackson Networks have some constraints, which are listed in the following [18]:

- There exists only one single class of service requests in the network.
- The overall number of service requests is not limited.
- Each of the N nodes in the network can be connected to an external Poisson source and a service response can leave the network at each node.
- All service times are exponentially distributed.
- The queueing discipline at each node is FIFO (FCFS).
- A node i consist of $m_i \geq 1$ identical servers with the service rate μ_i with $i = 1, .., N$. The arrival rates λ_{0i} and the service rates μ_i could depend on the number of k_i service requests in the node i. In this case the service and arrival rates would be load dependent. [1]

An algorithm for the calculation of open networks, based on this theorem, is then mainly composed of three steps [18, 132]:

- *Step 1*: Calculation of the arrival rates λ_i for every node $i = 1, .., N$, based on the setup and solving of the equation system of the traffic equations.
- *Step 2*: Ergodicity check and computation of the state probabilities $p_i\left(k_i\right)$ for every station i. While every node could be considered as a M/M/1 resp. M/M/m station, the performance values of the nodes could be computed using the formulas in table A.3 and A.4.
- *Step 3*: Finally the overall steady-state probabilities and the global performance values can be computed.

The following network from [18] in figure 2.4 will exemplify this algorithm. This example will also be used in section 3.5 for the explanation of the FMC-QE transformations and the comparison of classic open network computation and the FMC-QE computations.

[1] A Queueing Station with more than one server and constant arrival rates is equivalent to a service station with one server and a load dependent service rate [18]:

$$\mu_i = \begin{cases} k_i \mu_i & k_i \leq m_i \\ m_i \mu_i & k_i \geq m_i \end{cases}$$

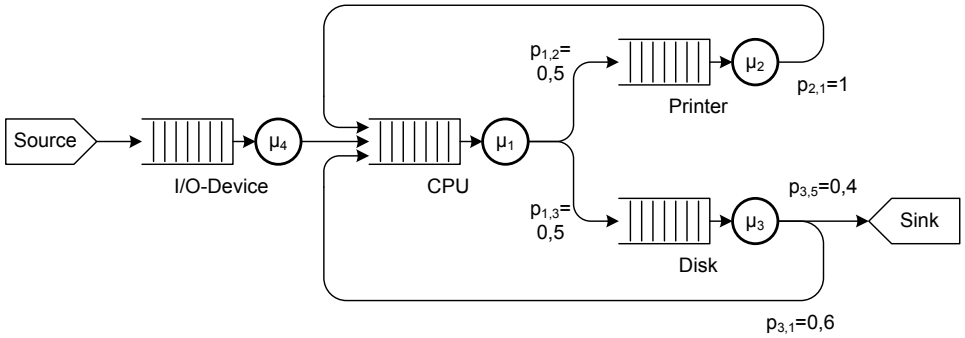


Figure 2.4: Sample Open Network - Jackson's Theorem [18]

In this example all nodes are single server FCFS stations. The service rates are exponentially distributed and defined as [18]:

$$\mu_1 = 25\,\frac{[Jobs]}{[s]};\ \mu_2 = 33\frac{Jobs}{3}\,\frac{[1]}{[s]};\ \mu_3 = 16\frac{2}{3}\,\frac{[Jobs]}{[s]};\ \mu_4 = 20\,\frac{[Jobs]}{[s]}$$

The exponential distributed arrival rate is defined as [18]:

$$\lambda = \lambda_{0,4} = 4\,\frac{[Jobs]}{[s]},$$

and the routing probabilities are defined as [18]:

$$p_{12} = p_{13} = 0,5;\ p_{41} = p_{21} = 1,0;\ p_{31} = 0,6;\ p_{30} = 0,4.$$

In this example the performance values: utilizations ρ_i, mean number of jobs at a noden_i, mean response timesR_i, mean waiting times W_i, mean queue lengths $n_{q,i}$ and mean overall response time R as well as the steady-state probability of state $(k_1, k_2, k_3, k_4) = (3, 2, 4, 1)$ are sought.

In **step 1** the traffic equations are set up [18]:

$$\lambda_1 = \lambda_2 p_{2,1} + \lambda_3 p_{3,1} + \lambda_4 p_{4,1} = 20\,\frac{[Jobs]}{[s]};\quad \lambda_2 = \lambda_1 p_{1,2} = 10\,\frac{[Jobs]}{[s]};$$
$$\lambda_3 = \lambda_1 p_{1,3} = 10\,\frac{[Jobs]}{[s]};\qquad \lambda_4 = \lambda_{0,4} = 4\,\frac{[Jobs]}{[s]}.$$

In **step 2** the performance values and state probabilities for the different nodes are calculated as [18]:

Utilizations ρ_i as $\rho_i = \frac{\lambda_i}{\mu_i}$ [18]:

$$\rho_1 = \frac{\lambda_1}{\mu_1} = 0,8;\ \rho_2 = \frac{\lambda_2}{\mu_2} = 0,3;\ \rho_3 = \frac{\lambda_3}{\mu_3} = 0,6;\ \rho_4 = \frac{\lambda_4}{\mu_4} = 0,2.$$

Mean number of jobs at a node (M/M/1) $n_i = \frac{\rho_i}{1-\rho_i}$ [18]:

$$n_1 = 4\,[Jobs];\ n_2 = 0,429\,[Jobs];\ n_3 = 1,5\,[Jobs];\ n_4 = 0,25\,[Jobs].$$

Mean response times of the servers $R_i = \frac{n_i}{\lambda_i}$ [18]:

$$R_1 = 0,2\,[s];\ R_2 = 0,043\,[s];\ R_3 = 0,15\,[s];\ R_4 = 0,0625\,[s].$$

Mean waiting times $W_i = \frac{\rho_i}{\mu_i - \lambda_i}$ [18]:

$$W_1 = 0,16\,[s];\ W_2 = 0,013\,[s];\ W_3 = 0,09\,[s];\ W_4 = 0,0125\,[s].$$

Mean queue lengths $n_{i,q} = \frac{\rho_i^2}{1-\rho_i}$ [18]:

$$n_{1,q} = 3,2\,[Jobs];\ n_{2,q} = 0,129\,[Jobs];\ n_{3,q} = 0,9\,[Jobs];\ n_{4,q} = 0,05\,[Jobs].$$

Marginal probabilities $p_i(k) = (1-\rho_i)\,\rho_i^k$ for $(k_1, k_2, k_3, k_4) = (3,2,4,1)$[18]:

$$p_1(3) = 0,1024;\ p_2(2) = 0,063;\ p_3(4) = 0,0518;\ p_4(1) = 0,16.$$

Finally in **step 3** the mean overall response time is calculated by aggregating the mean number of jobs at every node and Little's Law [18]:

$$R = \frac{n}{\lambda} = \frac{1}{\lambda}\sum_{i=1}^{4} n_i = 1,545\,[s]$$

and the state probability for state $(k_1, k_2, k_3, k_4) = (3,2,4,1)$ is [18]:

$$p(3,2,4,1) = p_1(3) * p_2(2) * p_3(4) * p_4(1) = 0,0,0000534.$$

2.1.5 Closed Queueing Networks - Gordon-Newell Theorem

While in open networks with service request sources and sinks, the arrival rate is a free parameter of the source and the overall number of service requests in the system is dependent upon this arrival rate, in closed networks the overall number of service requests in the system is the free parameter and the arrival rate, more precisely the throughput, is a dependent value. Furthermore, in closed networks the overall number of service requests in the system is an integer in comparison to the resulting real value in an open network. In closed networks not a service request source is considered, but a global balance of the network with this constant number of service requests. Therefore a normalization constant is defined in the calculation of the steady-state probabilities. This section will give a general introduction into the handling of closed (Product Form) networks in the Queueing Theory. Further general discussions on the model of closed networks beyond this introduction could also be found later in section 4.1.1, where questions like: 'Who inserted the service requests in the system?' (initialization of a closed network) are raised.

Gordon and Newell [64] transfered and refined the results of Jackson to closed networks (while Jackson also already considered closed systems in [78]). Gordon Newell Networks have the same assumptions than Jackson Networks except, that no service request can enter or leave the system and therefore the number of service requests in the system is constant [18]:

$$K = \sum_{i=1}^{N} k_i \tag{2.10}$$

According to the **Gordon-Newell Theorem**, each network state in a Gordon Newell Network could be calculated using the following product-form expression [18, 64]:

$$p\left(k_1, .., k_N\right) = \frac{1}{G(K)} \prod_{i=1}^{N} F_i\left(k_i\right) \tag{2.11}$$

The normalization constant $G(K)$ is then defined as [18]:

$$G(K) = \sum_{\sum_{i=1}^{N} k_i = K} \prod_{i=1}^{N} F_i(k_i), \tag{2.12}$$

with $F_i(k_i)$ defined as [18]:

$$F_i(k_i) = \left(\frac{e_i}{\mu_i}\right) \frac{1}{\beta_i(k_i)}. \tag{2.13}$$

Where the visit ratios are defined as [18]:

$$e_i = \sum_{j=1}^{N} e_j p_{ji} \quad for \;\; i = 1, .., N. \tag{2.14}$$

and the function $\beta_i(k_i)$ is defined as [18]:

$$\beta_i(k_i) = \begin{cases} k_i! & k_i \leq m_i, \\ m_i! m_i^{k_i - m_i} & k_i \geq m_i, \\ 1 & m_i = 1. \end{cases} \tag{2.15}$$

Another definition of the function $F_i(k_i)$ for load dependent service rates is given as [18]:

$$F_i(k_i) = \frac{e_i^{k_i}}{A_i(k_i)}, \tag{2.16}$$

with $A_i(k_i)$ as [18]:

$$Ai(k_i) = \begin{cases} \sum_{j=1}^{k_i} \mu_i(j) & k_i > 0, \\ 1 & k_i = 0. \end{cases} \tag{2.17}$$

Where a constant service rate is a special case in this generalized function.

An algorithm for the computation of Gordon Newell Networks is separated into four steps [18]:

- *Step 1*: Calculation of the visit ratios e_i for all nodes.
- *Step 2*: Computation of the functions $F_i(k_i)$ for all nodes.
- *Step 3*: Computation of the normalization constant $G(K)$.
- *Step 4*: Computation of the state probabilities of the network and the required performance values.

Figure 2.5 illustrates an exemplary Gordon Newell Network (example 4.3-3 from [68], exemplary calculations from [18]).

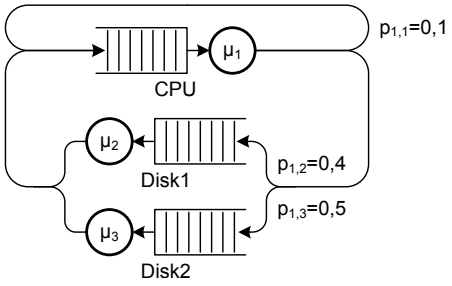


Figure 2.5: Sample Closed Network - Gordon Newell [68]

In this example the queueing discipline is FCFS at every node and the routing probabilities are defined as [68]:

$$p_{11} = 0,1;\ p_{12}0,4;\ p_{13} = 0,5;$$
$$p_{21} = 1,0;\ p_{22}0,0;\ p_{23} = 0,0;$$
$$p_{31} = 1,0;\ p_{32}0,0;\ p_{33} = 0,0.$$

The exponentially distributed service times are:

$$\mu_1 = 0,5;\ \mu_2 = 0,40;\ \mu_3 = 0,25.$$

There are 3 service requests in the system:

$$K = 3.$$

This network consists of 10 states:

$$(3,0,0);\ (2,1,0);\ (2,0,1);\ (1,2,0);\ (1,1,1);$$
$$(1,0,2);\ (0,3,0);\ (0,2,1);\ (0,1,2);\ (0,0,3).$$

In the first step the visit ratios e_i are determined as:

$$e_1 = e_1p_{11} + e_2p_{21} + e_3p_{31} = 1;$$
$$e_2 = e_1p_{12} + e_2p_{22} + e_3p_{32} = 0,4;$$
$$e_3 = e_1p_{13} + e_2p_{23} + e_3p_{33} = 0,5.$$

In the second step the functions $F_i(k_i)$ with $F_i(k_i) = \left(\frac{e_i}{\mu_i}\right)^{k_i}$ for $i = 1, 2, 3$ are computed:

$$\begin{aligned} &F_1(0) = 1;\ F_1(1) = 2;\ F_1(2) = 4;\ F_1(3) = 8;\\ &F_2(0) = 1;\ F_2(1) = 1;\ F_2(2) = 1;\ F_2(3) = 1;\\ &F_3(0) = 1;\ F_3(1) = 2;\ F_3(2) = 4;\ F_3(3) = 8. \end{aligned}$$

In the third step the normalization constant $G(3)$ is determined:

$$\begin{aligned} G(3) = \quad & F_1(3)F_2(0)F_3(0) + F_1(2)F_2(1)F_3(0) + F_1(2)F_2(0)F_3(1) +\\ & F_1(1)F_2(2)F_3(0) + F_1(1)F_2(1)F_3(1) + F_1(1)F_2(0)F_3(2) +\\ & F_1(0)F_2(3)F_3(0) + F_1(0)F_2(2)F_3(1) + F_1(0)F_2(1)F_3(2) +\\ & F_1(0)F_2(0)F_3(3) = 49. \end{aligned}$$

In the fourth step the different state probabilities $p(k_1, k_2, k_3) = \frac{1}{G(3)} \prod_{i=1}^{N} F_i(k_i)$ are calculated:

$$\begin{aligned} &p(3,0,0) = \frac{8}{49};\ p(2,1,0) = \frac{4}{49};\ p(2,0,1) = \frac{8}{49};\ p(1,2,0) = \frac{2}{49};\ p(1,1,1) = \frac{4}{49};\\ &p(1,0,2) = \frac{8}{49};\ p(0,3,0) = \frac{1}{49};\ p(0,2,1) = \frac{2}{49};\ p(0,1,2) = \frac{4}{49};\ p(0,0,3) = \frac{8}{49}. \end{aligned}$$

This leads to the following marginal state probabilities:

$$\begin{aligned} p_1(0) &= p(0,3,0) + p(0,2,1) + p(0,1,2) + p(0,0,3) = \frac{15}{49};\\ p_1(1) &= p(1,2,0) + p(1,1,1) + p(1,0,2) = \frac{14}{49};\\ p_1(2) &= p(2,1,0) + p(2,0,1) = \frac{12}{49};\\ p_1(3) &= p(3,0,0) = \frac{8}{49}; \end{aligned}$$

$$\begin{aligned} p_2(0) &= p(2,0,1) + p(1,0,2) + p(0,0,3) + p(3,0,0) = \frac{32}{49};\\ p_2(1) &= p(2,1,0) + p(1,1,1) + p(0,1,2) = \frac{12}{49};\\ p_2(2) &= p(1,2,0) + p(0,2,1) = \frac{4}{49};\\ p_2(3) &= p(0,3,0) = \frac{1}{49}; \end{aligned}$$

$$\begin{aligned} p_3(0) &= p(3,0,0) + p(2,1,0) + p(1,2,0) + p(0,3,0) = \frac{15}{49};\\ p_3(1) &= p(2,0,1) + p(1,1,1) + p(0,2,1) = \frac{14}{49};\\ p_3(2) &= p(1,0,2) + p(0,1,2) = \frac{12}{49};\\ p_3(3) &= p(0,0,3) = \frac{8}{49}. \end{aligned}$$

Finally, the performance values are derived, like the utilization $\rho_i = 1 - p_i(0)$:

$$\rho_1 = 0,69;\ \rho_2 = 0,35;\ \rho_3 = 0,69.$$

The mean number of service requests at a station $n_i = \sum_{k=1}^{3} k p_i(k)$:

$$n_1 = 1,27;\ n_2 = 0,47;\ n_3 = 1,27.$$

The throughput at every node $D_i = m_i \rho_i \mu_i$:

$$D_1 = 0,35;\ D_2 = 0,14;\ D_3 = 0,17,$$

and the mean response times $R_i = \frac{n_i}{D_i}$:

$$R_1 = 3,65;\ R_2 = 3,38;\ R_3 = 7,27.$$

This leads to the overall performance values of:

$$D = 0,35;\ R = 8,64.$$

2.1.6 Mixed Open and Closed Queueing Networks - BCMP Theorem

Baskett, Chandy, Muntz and Palacios (BCMP) [8] extended the results of Jackson and Gordon and Newell to open, closed and mixed networks with several job classes, different kinds of servers and state dependent arrival processes.

A BCMP network can consist of an arbitrary but finite number of queueing stations (service centers) of the following type [8]:

- Type-1: -/M/m - FCFS,
- Type-2: -/G/1 - PS,
- Type-3: -/G/∞,
- Type-4: -/G/1 - LCFS PR.

Where the different types of servers are defined as:

Type-1: -/M/m - FCFS [8]:

- first-come-first-served (FCFS) service discipline,
- all service requests get the same service time,
- the service time distribution is negative exponential,

- a state dependent service rate is possible (then: $m > 1$)[2].

Type-2: -/G/1 - PS [8]:

- single server, no queue,
- service discipline: processor sharing,
- each class of service request may have distinct service distributions,
- the service time distributions have Laplace transforms.

Type-3: -/G/∞ (IS) [8]:

- number of servers in the station $\geq$ maximum number of service requests,
- each class of service request may have distinct service distributions,
- the service time distributions have Laplace transforms.

Type-4: -/G/1 - LCFS PR [8]:

- single server,
- preemptive-resume last-come-first-served (LCFS PR) service discipline,
- each class of service request may have distinct service distributions,
- the service time distributions have Laplace transforms.

For open networks two kinds of arrival processes can be distinguished from each other [18]:

In the first case, the arrival process is Poisson and all service requests arrive in the network from one source with an overall arrival rate λ, which can depend on the number of jobs in the network. The probability, that an arriving service request is then routed from the source 0 to the service station i in class r is defined as $p_{0,ir}$, where [18]:

$$\sum_{i=1}^{N} \sum_{r=1}^{R} p_{0,ir} = 1, \tag{2.18}$$

where N is the overall number of service stations and R is the number of classes in the network.

In the second case, the arrival process consists of U independent Poisson arrival streams, where each source is assigned to a different chain. An arrival rate λ_U could be load dependent. A service request arrives at service station i with the probability $p_{0,ir}$, where [18]:

[2] A queueing station with more than one server and constant arrival rates is equivalent to a service station with one server and a load dependent service rate [18]:

$$\mu_i = \begin{cases} k_i \mu_i & k_i \leq m_i \\ m_i \mu_i & k_i \geq m_i \end{cases}$$

$$\sum_{r \in C_u;\, i=1}^{N} p_{0,ir} = 1 \; \forall \; u = 1,..U. \tag{2.19}$$

For this kinds of networks the BCMP Theorem states [8]:

For a network of service stations which is open, closed or mixed in which each service center is of type 1,2,3 or 4, the steady-state state probabilities are given by:

$$p(S_1,..,S_N) = \frac{1}{G\,(K)} d\,(S) \sum_{i=1}^{N} f_i(S_i), \tag{2.20}$$

where $G(K)$ is the normalization constant, $d(S)$ is a function of the number of service requests in the system and each $f_i(S_i)$ is a function that depends on the type of the service center i.

The function $d(S)$ is defined as [8, 18]:

$$d(S) = \begin{cases} \prod_{i=0}^{K(S)-1} \lambda(i) & \textit{open network with arrival process } 1, \\ \prod_{u=1}^{U} \prod_{i=0}^{K(S)-1} \lambda_u(i) & \textit{open network with arrival process } 2, \\ 1 & \textit{closed network.} \end{cases} \tag{2.21}$$

Where $K(S)$ is the overall number of service requests in the network and $K_u(S)$ is the overall number service requests in from the uth source.

The function $f_i(S_i)$ is defined as [8, 18]:

$$f_i(S_i) = \begin{cases} \left(\frac{1}{\mu_i}\right)^{k_i} \prod_{j=1}^{k_i} e_{is_{ij}} & \textit{Type } 1, \\ k_i! \prod_{r=1}^{R} \prod_{l=1}^{u_{ir}} \frac{1}{k_{irl}!} \left(\frac{e_{ir} A_{irl}}{\mu_{irl}}\right)^{k_{irl}} & \textit{Type } 2, \\ \prod_{r=1}^{R} \prod_{l=1}^{u_{ir}} \frac{1}{k_{irl}!} \left(\frac{e_{ir} A_{irl}}{\mu_{irl}}\right)^{k_{irl}} & \textit{Type } 3, \\ \prod_{j=1}^{k_i} \frac{e_{ir_j} A_{ir_j m_j}}{\mu_{ir_j m_j}}, & \end{cases} \tag{2.22}$$

with [18]:

s_{ij} Class of the service request in the j th position of the FCFS queue.

μ_{irl} The mean service rate of the node i in the class r at the lth phase ($l = 1,..,u_{ir}$) in a Cox distribution.

u_{ir} The maximum number of exponential phases of the class r in node i.

A_{irl} $A_{irl} = \prod_{j=0}^{l-1} a_{irj}$ is the probability, that a class-r service request in node i reaches the lth stage ($A_{ir1} = 1$ because of $a_{ir0} = 1$).

a_{irl} The probability, that a class-r service request at the node i moves to the $(j+1)$th phase.

k_{irl} Number of class-r service requests in phase l.

k_i $k_i = \sum_{r=1}^{R} k_{ir}$ overall number of service requests of all classes in node i.

In their paper [8] Baskett, Chandy, Muntz and Palacios provided also two simplifications of the equilibrium state probabilities for closed networks and open network, in which the arrival rate does not depend on the state of the model.

For a closed system the steady-state state probabilities are given by [8, 18]:

$$p(S_1, .., S_N) = \frac{1}{G(K)} \sum_{i=1}^{N} F_i(S_i), \tag{2.23}$$

where the normalization constant $G(K)$ is defined as [18]:

$$G(K) = \sum_{\sum_{i=1}^{N} S_i = K} \prod_{i=1}^{N} F_i(S_i) \tag{2.24}$$

and the function $F_i(S_i)$ is defined as [8, 18]:

$$f_i(S_i) = \begin{cases} k_i! \frac{1}{\beta_i(k_i)} \left(\frac{1}{\mu_i}\right)^{k_i} \prod_{r=1}^{R} \frac{1}{k_{ir}!} (e_{ir})^{k_{ir}} & \textit{Type } 1, \\ \frac{k_i!}{\prod_{j=1}^{k_i} \mu_i(j)} \prod_{r=1}^{R} \frac{1}{k_{ir}!} (e_{ir})^{k_{ir}} & \textit{Type } 1 \ (m_i = 1), \textit{ load dependent service rate } \mu_i(j), \\ k_i! \prod_{r=1}^{R} \frac{1}{k_{ir}!} \left(\frac{e_{ir}}{\mu_{ir}}\right)^{k_{ir}} & \textit{Type } 2, 4, \\ \prod_{r=1}^{R} \frac{1}{k_{ir}!} \left(\frac{e_{ir}}{\mu_{ir}}\right)^{k_{ir}} & \textit{Type } 3. \end{cases} \tag{2.25}$$

with [18]:

$$k_i = \sum_{r=1}^{R} k_{ir}, \tag{2.26}$$

$$\beta_i(k_i) = \begin{cases} k_i! & k_i \leq m_i, \\ m_i! m_i^{k_i - m_i} & k_i \geq m_i, \\ 1 & m_i = 1. \end{cases} \tag{2.27}$$

and [18]:

$$e_{ir} = \sum_{s \in C_q;\ j=1}^{N} e_{js} p_{js,ir}. \tag{2.28}$$

An example for a closed BCMP Queueing network is omitted, while the calculation of the normalization constant involves the calculation of all states of the network and could therefore

be very large. An optimized algorithm for the calculation of these networks is for example provided through the Mean Value Analysis, explained in the next subsection, where also a descriptive example is given.

For open BCMP networks with load-independent arrival and service rates the steady-state state probability simplifies to [8, 18]:

$$p(k_1, .., k_N) = \sum_{i=1}^{N} p_i(k_i), \tag{2.29}$$

where the individual steady-state state probabilities are defined as [8, 18]:

$$p_i(k_i) = \begin{cases} (1 - \rho_i)\, \rho_i^{k_i} & Type\ 1, 2, 4\ (m_i = 1), \\ e^{-\rho_i} \frac{\rho_i^{k_i}}{k_i!} & Type\ 3. \end{cases} \tag{2.30}$$

with [18]:

$$k_i = \sum_{i=1}^{R} k_{ir} \tag{2.31}$$

and [18]:

$$\rho_i = \sum_{i=1}^{R} \rho_{ir}, \tag{2.32}$$

$$\rho_{ir} = \begin{cases} \lambda_r \frac{e_{ir}}{\mu_i} & Type\ 1\ (m_i = 1), \\ \lambda_r \frac{e_{ir}}{\mu_{ir}} & Type\ 2, 3, 4., \end{cases} \tag{2.33}$$

where the ergodicity conditions ($\rho < 1$) holds for every server.

The mean number of service requests per class in every server is then provided as [18]:

$$n_{ir} = \frac{\rho_{ir}}{1 - \rho_{ir}}. \tag{2.34}$$

For the calculation of the steady-state state probabilities for Type-1 nodes with more than one server ($m_i > 1$), the formulas from table A.4 are used.

An algorithm for the calculation of the performance values of an open BCMP network with load-independent arrival and service rates could then be defined as follows [18]:

- *Step 1*: Compute the visit ratios e_{ir} for $i = 1, .., N$ and $r = 1, .., R$.
- *Step 2*: Compute the utilizations ρ_i for each node.
- *Step 3*: Compute the other performance values.
- *Step 4*: Compute the steady-state state probabilities for every node.
- *Step 5*: Compute the overall steady-state state probabilities.

An illustrative example, also used in [18], is defined as follows:

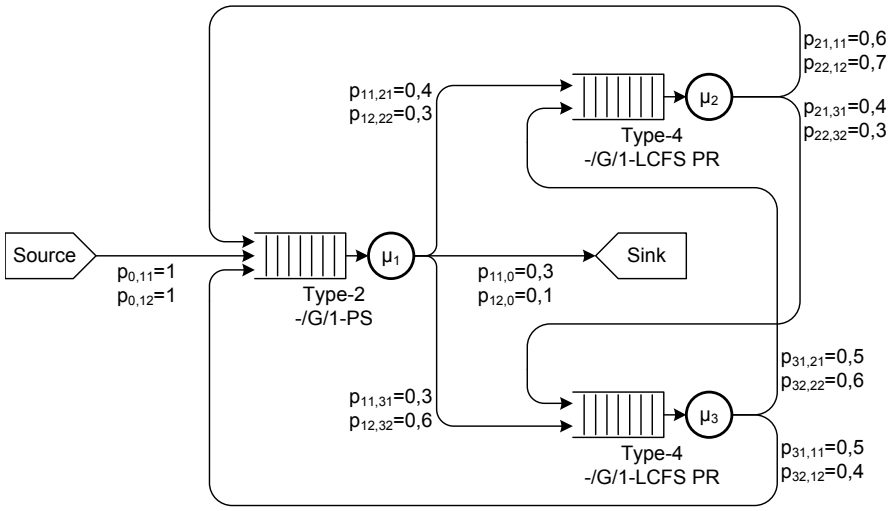


Figure 2.6: Sample Open Network - BCMP [18]

The network is an open network of three nodes ($N = 3$) and two service request classes ($R = 2$), shown in figure 2.6.

The service times are exponentially distributed and the service rates are defined as [18]:

$$\mu_{11} = \frac{1}{8s}; \quad \mu_{21} = \frac{1}{12s}; \mu_{31} = \frac{1}{8s};$$
$$\mu_{12} = \frac{1}{24s}; \quad \mu_{22} = \frac{1}{32s}; \mu_{32} = \frac{1}{36s}.$$

The arrival rates are also exponentially distributed as [18]:

$$\lambda_1 = \lambda_2 = \frac{1}{8}\frac{SRq}{s}.$$

In the example the demanded values are the mean number of service requests at every node n_{ir} and the probability for the state $(k_1, k_2, k_3) = (3, 2, 1)$

In the first step the visit ratios e_{ir} as $e_{ir} = p_{0,ir} + \sum_{s \in C_q,\ j=1}^{N} e_{js} p_{js,ir}$ are computed for $r = 1, .., R$ and $i = 1, .., N$ [18]:

$$e_{11} = p_{0,11} + e_{11}p_{11,11} + e_{21}p_{21,11} + e_{31}p_{31,11} = 3,333;$$
$$e_{21} = p_{0,21} + e_{11}p_{11,21} + e_{21}p_{21,21} + e_{31}p_{31,31} = 2,292;$$
$$e_{31} = p_{0,31} + e_{11}p_{11,31} + e_{21}p_{21,31} + e_{31}p_{31,21} = 1,917;$$

$$e_{12} = p_{0,12} + e_{12}p_{12,12} + e_{22}p_{22,12} + e_{32}p_{32,12} = 10;$$
$$e_{22} = p_{0,22} + e_{12}p_{12,22} + e_{22}p_{22,22} + e_{32}p_{32,32} = 8,049;$$
$$e_{32} = p_{0,32} + e_{12}p_{12,32} + e_{22}p_{22,32} + e_{32}p_{32,22} = 8,415.$$

In the second step the utilizations ρ_i are computed [18]:

$$\rho_1 = \lambda_1 \frac{e_{11}}{\mu_{11}} + \lambda_2 \frac{e_{12}}{\mu_{12}} = \rho 11 + \rho_{12} = 0,833;$$
$$\rho_2 = \lambda_1 \frac{e_{21}}{\mu_{21}} + \lambda_2 \frac{e_{22}}{\mu_{22}} = \rho 21 + \rho_{22} = 0,442;$$
$$\rho_3 = \lambda_1 \frac{e_{31}}{\mu_{31}} + \lambda_2 \frac{e_{32}}{\mu_{32}} = \rho 31 + \rho_{32} = 0,354.$$

In the third step the mean number of service requests at every node ($n_{ir} = \frac{\rho_{ir}}{1-\rho_i}$) are computed [18]:

$$n_{11} = 2,500;\ n_{21} = 0,342;\ n_{31} = 0,186;$$
$$n_{12} = 2,500;\ n_{22} = 0,500;\ n_{32} = 0,362.$$

In the fourth step the marginal probabilities ($p_i(k_i) = (1-\rho_i)\rho_i^{k_i}$ - Type 1,2,4 ($m_i = 1$)) are computed [18]:

$$p_1(3) = 0,0965;\ p_2(2) = 0,1093;\ p_3(1) = 0,2287.$$

Finally in the fifth step the steady-state state probability for the state $(k_1, k_2, k_3) = (3, 2, 1)$ is computed [18]:

$$p(3, 2, 1) = p_1(3)p_2(2)p_3(1) = 0,00241.$$

2.1.7 An Algorithm for Product Form Networks - Mean Value Analysis (MVA)

The Mean Value Analysis [118] is an algorithm for the calculation of the performance values of closed queueing networks with product form solutions. It is based on Little's Law [98] and the Theorem of the distribution at arrival time (Arrival Theorem, Reiser/Lavenberg-Theorem [93, 121]), which states, that in a closed Product Form Queueing Network, the probability mass function of the number of jobs seen at the time of the arrival at node i when there are k service requests in the network is equal to the probability mass function of the number of jobs at this node with one less job in the network [18].

This iterative algorithm is then defined as follows [18]:

In *step 1* the performance values are initialized:

For $i = 1, .., N$ and $j = 1, .., (m_i - 1)$:

$$n_i = 0;\ p_i(0|0) = 1;\ p_i(j|0) = 0. \tag{2.35}$$

Step 2 is an iteration over the number of service requests $k = 1, .., K$ with the corresponding sub steps:

Step 2.1 Computation of the mean response times of a service request at each node:

$$R_i(k) = \begin{cases} \frac{1}{\mu_i}\left(1 + n_i(k-1)\right) & Type - 1, 2, 4\ (m_i = 1), \\ \frac{1}{\mu_i m_i}\left(1 + n_i(k-1) + \sum_{j=0}^{m_i-2}\left((m_i - j - 1)\, p_i(j|k-1)\right)\right) & Type - 1\ (m_i > 1), \\ \frac{1}{\mu_i} & Type - 3, \end{cases} \tag{2.36}$$

with the conditional probabilities of:

$$\begin{aligned} p_i(0|k) &= 1 - \frac{1}{m_i}\left(\frac{e_i D_i(k)}{\mu_i} + \sum_{j=1}^{m_i-1}\left((m_i - j)\, p_i(j|k)\right)\right), \\ p_i(j|k) &= \frac{D_i(k)}{\mu_i \alpha_i(j)} p_i(j-1|k-1). \end{aligned} \tag{2.37}$$

and the function $\alpha_i(j)$ defined as:

$$\alpha_i(j) = \begin{cases} j & if j \leq m_i, \\ m_i & else. \end{cases} \tag{2.38}$$

In *Step 2.2* the overall throughput $D(k)$:

$$D(k) = \frac{k}{\sum_{i=1}^{N} e_i R_i(k)} \tag{2.39}$$

and the node throughput $D_i(k)$ is computed:

$$D_i(k) = e_i D(k). \tag{2.40}$$

The mean number of service requests at every node is computed in *Step 2.3*:

$$n_i(k) = e_i D(k) R_i(k). \tag{2.41}$$

The other performance values are computed using the known formulas.

The Mean Value Analysis is furthermore extended to multi class networks, mixed networks and networks with load dependent service rates. More information on this extensions could be found in [18].

An example of the closed Gordon Newell Networks, illustrated in figure 2.7, is used for exemplification.

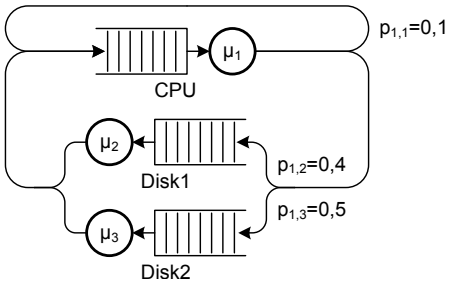


Figure 2.7: Sample Closed Network - MVA [68]

The performance parameters are:

$$\mu_1 = 0,5;\ \mu_2 = 0,40;\ \mu_3 = 0,25;$$
$$K = 3;$$
$$e_1 = 1;\ e_2 = 0,4;\ e_3 = 0,5.$$

In step 1, the n_i values are initialized. (The conditional probabilities $p_i(0|0)$ and $p_i(j|0)$ are not initialized, because there are no Type-1 $m_i > 1$ servers in the example):

$$n_1 = 0;\ n_2 = 0;\ n_3 = 0.$$

The first iteration ($j = 1$) of step 2 leads to the following results:

$$R_1 = 2[s];\ R_2 = 2,5[s];\ R_3 = 4[s];\ R = 5[s];$$
$$D_1 = 0,2\frac{1}{[s]};\ D_2 = 0,08\frac{1}{[s]};\ D_3 = 0,1\frac{1}{[s]};\ D = 0,2\frac{1}{[s]};$$
$$n_1 = 0,4;\ n_2 = 0,2;\ n_3 = 0,4;\ n = 1.$$

Second iteration ($j = 2$):

$$R_1 = 2,8[s];\ R_2 = 3[s];\ R_3 = 5,6[s];\ R = 6,8[s];$$
$$D_1 = 0,29\frac{1}{[s]};\ D_2 = 0,12\frac{1}{[s]};\ D_3 = 0,15\frac{1}{[s]};\ D = 0,29\frac{1}{[s]};$$
$$n_1 = 0,82;\ n_2 = 0,35;\ n_3 = 0,82;\ n = 2.$$

Third iteration ($j = 3 = K$):

$$R_1 = 3,65[s];\ R_2 = 3,39[s];\ R_3 = 7,29[s];\ R = 8,65[s];$$
$$D_1 = 0,35\frac{1}{[s]};\ D_2 = 0,14\frac{1}{[s]};\ D_3 = 0,17\frac{1}{[s]};\ D = 0,35\frac{1}{[s]};$$
$$n_1 = 1,27;\ n_2 = 0,47;\ n_3 = 1,27;\ n = 3.$$

2.2 Time Augmented Petri Nets

The dimension *time* is not considered in the original Petri Nets by Petri [113] and Holt and Commoner [75]. In this networks the causal relationships between the associated events or actions are to be investigated, as for example in the modeling of (business) processes or the development of asynchronous circuits. The quantitative analysis or steady state behavior of servers were not in the focus. Later on the Petri Nets were extended by the aspect time (as Time Augmented Petri Nets) in order to extend them for the quantitative analysis as an additional aim. Compared to Queueing Networks, the analysis of such Time Augmented Petri Nets is often executed on the level of the reachability graph, like illustrated in figure 2.8. While the Queueing Theory often abstracts from this level of detail through Product Form Solutions and defined equilibriums on the server level, in Time Augmented Petri Nets the quantitative behavior of system with complex (Non Product Form) control flows could be predicted with the trade-off of the possible state-space explosion problem, when considering the system on the level of continuous time Markov chains (CTMCs). A more extensive comparison in provided in section 4.4.

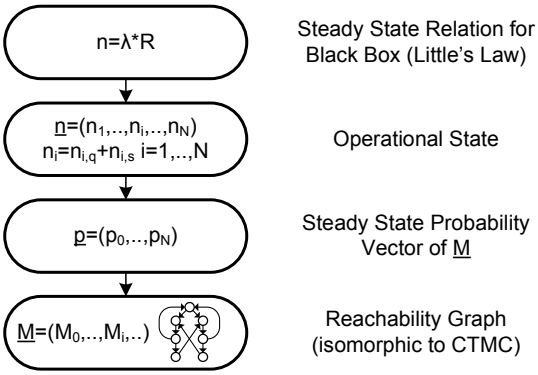


Figure 2.8: Range of Steady State Variables [140]

In this section an overview of some classifications and extensions of Time Augmented Petri Nets is given on the basis of [14].

2.2.1 Classification

The Time Augmented Petri Nets could be mainly divided into two classes, dependent which object type had a specified sojourn time [14]:

- Timed Places Petri Nets (TPPNs)
- Timed Transitions Petri Nets (TTPNs)

In Timed Places Petri Nets the sojourn times are connected to the places of the Petri Net. If a token enters a place, the corresponding transitions, for which this place is an input place, can only fire after a certain time interval.

On the other hand, in Timed Transitions Petri Nets the sojourn times are connected to the transitions.The transitions become enabled, but fire only after a certain time interval has elapsed. This group of Time Augmented Petri Nets could furthermore divided into two subclasses [14]:

- TTPNs with reservation (preselection models)
- TTPNs without reservation (race models)

In preselection models the transition selects all input tokens it needs to fire, when the transition is enabled, so that the tokens are unavailable for other possible concurrent transitions. When the corresponding time interval of the transition has elapsed, the input tokens are destroyed and the output tokens are created by the firing of the transition.

In race models the tokens are not reserved. In these models concurrent transitions get enabled parallel and the transition, which fires first, disables the other possible concurrent transitions.

Another differentiator is, if the mentioned times are defined deterministic or with a stochastic distribution.

In some cases also different firing policies of transitions are distinguished. While the considered Petri Nets have exponential distributed firing delays, this differentiation is irrelevant [14].

In the following some important Time Augmented Petri Net types are listed:

- In continuous time stochastic Petri Nets (SPN) [104, 106] (according to [14]) all transitions are augmented with a stochastic firing time interval.
- Generalized Stochastic Petri Nets (GSPN) [101, 102] introduce immediate transitions.
- In Deterministic Stochastic Petri Nets (DSPN) [34] the transitions are associated to deterministic as well as stochastic firing times.
- Queueing Petri Nets [9, 10, 14] combine Queueing Networks and Petri Nets within one net and in a mixed TPPN/TTPN.

2.2.2 Continuous Time Stochastic Petri Nets (SPN)

Continuous Time Stochastic Petri Nets (SPN) [104, 106] (according to [14]) extend a Petri Net by connecting a transition rate ω_i to every transition t_i.

While the firing delays in SPNs are exponentially distributed and therefore have a memoryless property, an SPN is isomorphic to a continuous time Markov chain (CTMC) [102]. Furthermore, a k-bounded SPN[3] system is isomorphic to a finite CTMC [102]. Thus the corresponding CTMC of an SPN could be obtained by applying the following rules [102]:

- The CTMC state space $S = s_i$ corresponds to the reachability set $RS(M_0)$ of the Petri Net associated with the SPN $(M_i \leftrightarrow s_i)$.
- The transition rate from state s_i (corresponding to marking M_i) to state s_j (M_j) is obtained as the sum of the firing rates of the transitions that are enabled in M_i and whose firings generate marking M_j.

[3] k-bounded [21]: SPN is k-bounded, if the number of tokens in each place is less or equal to k for all markings in $RS(M_i)$.

As a simplification, in this section, like in [102], the transitions are associated with single server semantics and a marking independent speed, but this has not to be the case in every system.

Based on this rules, the infinitesimal generator (state transition rate matrix Q) of the CTMC could be constructed from the SPN [102]. Every cell in this matrix is then defined as [102]:

$$q_{ij} = \begin{cases} \sum\limits_{T_k \in E_j(M_i)} \omega_k & i \neq j, \\ -q_i & i = j, \end{cases} \tag{2.42}$$

where:

$$q_i = \sum_{T_k \in E(M_i)} \omega_k, \tag{2.43}$$

with ω_k as firing rate of transition T_k and $E_j(M_i)$ as the set of transitions whose firings bring the net from M_i to M_j. If the SPN is ergodic, the steady state probability distribution vector π could then be found by solving the linear system [102]:

$$\pi Q = 0. \tag{2.44}$$

The probability, that a transition $T_k \in E(M_i)$ fires (first) in marking M_i has the expression [102]:

$$p(T_k|M_i) = \frac{\omega_i}{q_i}. \tag{2.45}$$

The average sojourn time in marking M_i is [102]:

$$SJ_i = \frac{1}{q_i}. \tag{2.46}$$

Through the steady state probability distribution vector π, the probability of being in a subset B of markings is defined as the sum of the steady state probabilities of the different markings [14]:

$$p(B) = \sum_{M_i \in B} \pi_i. \tag{2.47}$$

The throughput of a timed transition T_k is further defined by the mean number of firings in steady state as the firing rate ω_k multiplied by the sum of the steady state probabilities in which T_k is enabled [14]:

$$D_k = \omega_k \sum_{T_k \in E(M_i)} \pi_i. \tag{2.48}$$

In order to calculate the mean number of tokens enabling a transition T_k, also interpreted as n_k, the steady state probabilities π_i of all markings M_i enabling T_k are added:

$$n_k = \sum_{T_k \in E(M_i)} \pi_i. \tag{2.49}$$

The continuous time Stochastic Petri Net in figure 2.9 exemplifies an SPN and will be analyzed in this section. The example originates from [102].

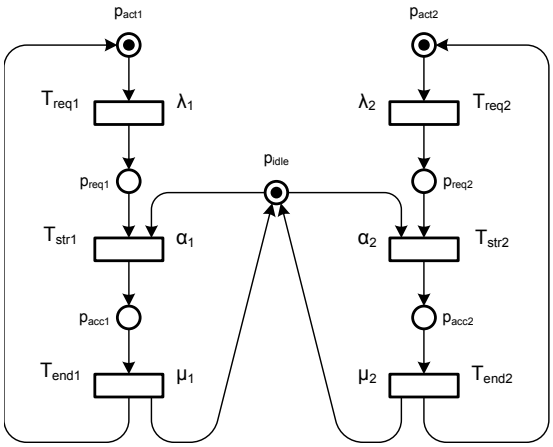


Figure 2.9: SPN Example [102]

The performance parameters of this example are defined in table 2.1.

Transition	Rate	Semantics
T_{req1}	$\lambda_1 = 1$	single server
T_{req2}	$\lambda_2 = 2$	single server
T_{str1}	$\alpha_1 = 100$	single server
T_{str2}	$\alpha_1 = 100$	single server
T_{end1}	$\mu_1 = 10$	single server
T_{end2}	$\mu_2 = 5$	single server

Table 2.1: SPN Example - Parameters [102]

In the first step of the calculation of this model, the corresponding reachability graph is set up, as illustrated in figure 2.10.

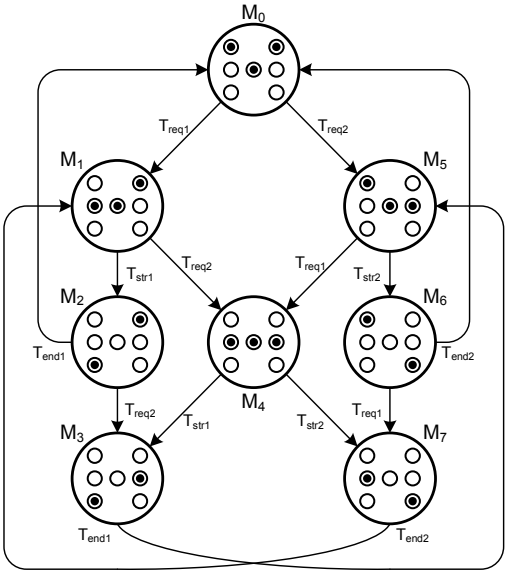


Figure 2.10: SPN Example - Reachability Graph [102]

After that the reachability set is derived as [102]:

$$
\begin{aligned}
M_0 &= p_{act1} + \quad p_{idle} + \quad p_{act2} \\
M_1 &= \quad p_{req1} + \quad p_{idle} + \quad p_{act2} \\
M_2 &= \quad p_{acc1} + \quad p_{act2} \\
M_3 &= \quad p_{acc1} + \quad p_{req2} \\
M_4 &= \quad p_{req1} + \quad p_{idle} + \quad p_{req2} \\
M_5 &= p_{act1} + \quad p_{idle} + \quad p_{req2} \\
M_6 &= p_{act1} + \quad p_{acc2} \\
M_7 &= \quad p_{req1} + \quad p_{acc2}
\end{aligned}
$$

This leads to the state transition rate diagram of the associated Markov chain, depicted in figure 2.11.

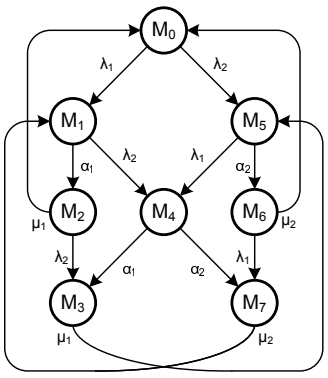


Figure 2.11: State Transition Rate Diagram of the SPN Example [102]

Knowing this state transition rate diagram, the state transition rate matrix Q is set up as [102]:

$$Q = \begin{bmatrix} -\lambda_1 - \lambda_2 & \lambda_1 & 0 & 0 & 0 & \lambda_2 & 0 & 0 \\ 0 & -\alpha_1 - \lambda_2 & \alpha_1 & 0 & \lambda_2 & 0 & 0 & 0 \\ \mu_1 & 0 & -\lambda_2 - \mu_1 & \lambda_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\mu_1 & 0 & \mu_1 & 0 & 0 \\ 0 & 0 & 0 & \alpha_1 & -\alpha_1 - \alpha_2 & 0 & 0 & \alpha_2 \\ 0 & 0 & 0 & 0 & \lambda_1 & -\alpha_2 - \lambda_1 & \alpha_2 & 0 \\ \mu_2 & 0 & 0 & 0 & 0 & 0 & -\lambda_1 - \mu_2 & \lambda_1 \\ 0 & \mu_2 & 0 & 0 & 0 & 0 & 0 & -\mu_2 \end{bmatrix}$$

If then $\pi Q = 0$ and $\sum_{i=0}^{7} \pi_i = 1$ and this linear system is solved, the steady state probabilities are [102]:

$$\begin{aligned} &\pi_0 = 0,61471, \quad \pi_1 = 0,00842, \pi_2 = 0,07014, \quad \pi_3 = 0,01556; \\ &\pi_4 = 0,00015, \quad \pi_5 = 0,01371, \pi_6 = 0,22854, \quad \pi_7 = 0,04876. \end{aligned}$$

An exemplary performance value could be the utilization of the shared memory. Therefore the expectation value of the markings, in which the place p_{idle} is occupied, is calculated as [102]:

$$E(M(p_{idle})) = \pi_0 + \pi_1 + \pi_4 + \pi_5 = 0,6370.$$

The utilization of the shared memory is then defined as [102, 128]:

$$\rho_{shared\ memory} = 1 - E(M(p_{idle})) = 0,3630.$$

Another exemplary performance value is the throughput of the transition T_{req1}, which could be interpreted as the throughput of the left subnet, calculated as $D_k = \omega_k \sum_{M_i \in A_k} \pi_k$, in which A_k is the subset of the result set in which T_j (in this case T_{req1}) is enabled [102]:

$$D_{req1} = \lambda_1(\pi_0 + \pi_5 + \pi_6) = 0,8570.$$

Compared to a solution of a closed queueing network calculated with the help of the Gordon-Newell Theorem, like shown in section 2.1.5, in this calculated example the interdependencies of the different transitions through the semaphore prevented the usage of a Product Form Solution. Therefore, the network had to be calculated on the level of reachability graphs. On the other hand, networks which are accessible to Product Form Solutions could also be calculated on the level of CTMCs, but this additional effort would not result in more precise calculations on the level of steady state probabilities, as for Product Form Networks, Product Form Solutions are correct.

2.2.3 Generalized Stochastic Petri Nets (GSPN)

Generalized Stochastic Petri Nets (GSPN) [101, 102] extend the SPN by introducing immediate transitions alongside the timed transitions. These immediate transitions fire in zero time, while the timed transitions fire after a random, exponentially distributed enabling time, like in the SPN [14].

If transitions in a GSPN are enabled concurrently, which transition fires is decided as [14]:

- If the concurrent transitions are timed transitions, the transition, which fires first, disables the other transitions (like in an SPN).
- If both timed and immediate transitions are enabled, only the immediate transition will be enabled (immediate transitions have priority).
- If only an immediate transition is enabled, it fires with probability 1.
- If immediate transitions are enabled in parallel, the random switch or switching distribution is often defined by firing weights ω_i. If then, for example, two transitions T_1 and T_2 are enabled in some marking M, the probability of firing T_1 is $\frac{\omega_1}{\omega_1+\omega_2}$.

The different states in a GSPN are then partitioned into tangible states $\hat{T}$ (all transitions enabled in this state are timed) and vanishing states $\hat{V}$ (at least one enabled transition in this state is immediate), where $\hat{T} \cap \hat{V} = \emptyset$.

In order to analyze the performance behavior of a GSPN, an embedded Markov chain could be considered. The corresponding transition state probability matrix is defined as an augmented matrix with respect to the sets of vanishing and tangible states [14]:

$$P = \begin{bmatrix} C & D \\ E & F \end{bmatrix} \tag{2.50}$$

Where in C are the transition probabilities from marking or state M_i to M_j, where M_i and M_j are vanishing states [14]:

$$C = (c_{ij}),\ c_{ij} = p(M_i \to M_j;\ M_i \in \hat{V},\ M_j \in \hat{V}). \tag{2.51}$$

D are the transition probabilities from vanishing states to transition states [14]:

$$D = (d_{ij}),\ d_{ij} = p(M_i \to M_j;\ M_i \in \hat{V},\ M_j \in \hat{T}) \tag{2.52}$$

and E and F accordingly [14]:

$$E = (e_{ij}),\ e_{ij} = p(M_i \to M_j;\ M_i \in \hat{T},\ M_j \in \hat{V}) \tag{2.53}$$

$$F = (f_{ij}),\ f_{ij} = p(M_i \to M_j;\ M_i \in \hat{T},\ M_j \in \hat{T}) \tag{2.54}$$

Every transition probability from marking of state M_i to M_j is then defined as the sum of all firing rates resp. firing weights ω_k, whose transitions T_k are enabled in marking M_i and which

firing produce M_j [4] divided by the sum of of all firing rates resp. firing weights ω_r of the transitions T_r, enabled in state M_i [5] [14, 102]:

$$p(M_i \to M_j) = \frac{\sum\limits_{T_k \in E_j(M_i)} \omega_k}{\sum\limits_{T_r \in E(M_i)} \omega_r} \tag{2.55}$$

The steady state distribution $\tilde{\pi}$ of the embedded Markov chain is given by [14]:

$$\tilde{\pi} P = \tilde{\pi} \tag{2.56}$$

and [14]:

$$\sum_{M_i \in \hat{T} \cup \hat{V}} \tilde{\pi} = 1. \tag{2.57}$$

The steady state distribution π of the original stochastic process could be derived from the steady state distribution $\tilde{\pi}$ of the embedded Markov chain by weighting the probability $\tilde{\pi}_j$ with the portion of time the process spends in marking M_j [14]:

If M_j is a vanishing marking, π_j is 0. If M_j is a tangible marking, the steady state probability π_j is the fraction of time the stochastic process spends in marking M_j. This is characterized by the mean time the stochastic process spends in this state [14]:

$$\frac{1}{\sum\limits_{T_k \in E(M_j)} \omega_k} \tag{2.58}$$

divided by the mean cycle time [14] for this marking:

$$\frac{1}{\tilde{\pi}_j} \sum_{M_s \in \hat{T}} \left(\frac{\tilde{\pi}_s}{\sum\limits_{T_k \in E(M_s)} \omega_k} \right) \tag{2.59}$$

This leads to the steady state distribution [14]:

$$\pi_j = \begin{cases} \dfrac{\tilde{\pi}_j \frac{1}{\sum\limits_{T_k \in E(M_j)} \omega_k}}{\sum\limits_{M_s \in \hat{T}} \left(\frac{\tilde{\pi}_s}{\sum\limits_{T_k \in E(M_s)} \omega_k} \right)} & M_j \in \hat{T}, \\ 0 & M_j \in \hat{V}. \end{cases} \tag{2.60}$$

Knowing the steady state distribution π, all other performance values could be derived like for an SPN.

[4] $E_j(M_i)$ denotes the set of transitions enabled in M_i whose firings produces M_j [102]

[5] $E(M_i)$ denotes the set of transitions enabled in marking M_i [102]

As an exemplary GSPN, the SPN example of subsection 2.2.2 has been slightly modified, that t_{str1} and t_{str2} are immediate transitions and therefore the token on place p_{idle} could be considered as a real semaphore. This is illustrated in figure 2.12.

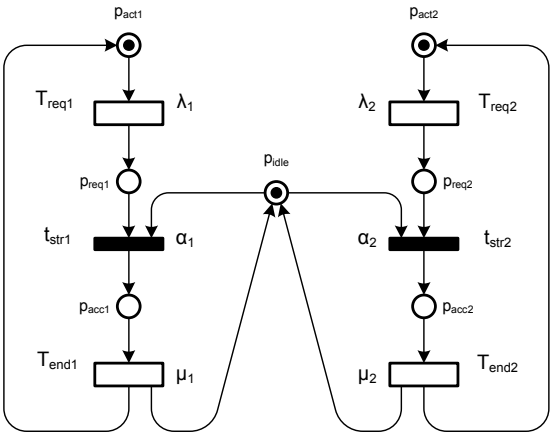


Figure 2.12: GSPN Example [102]

The performance parameters of this example are defined in table 2.2.

Transition	Rate/Weight	Semantics
T_{req1}	$\lambda_1 = 1$	single server
T_{req2}	$\lambda_2 = 2$	single server
T_{str1}	$\alpha_1 = 1$	immediate
T_{str2}	$\alpha_1 = 1$	immediate
T_{end1}	$\mu_1 = 10$	single server
T_{end2}	$\mu_2 = 5$	single server

Table 2.2: GSPN Example - Parameters

The new reachability graph, illustrated in figure 2.13, has slightly changed comparing to figure 2.10, because the old state M_4 is impossible. This state was deleted, because it was reached through the firing of transition T_{req2} resp. T_{req1}, while each of the transitions was concurrently enabled with transition T_{str1} resp. T_{str2}. The transitions T_{str1} and T_{str2} are immediate transitions in this model, so the timed transitions T_{req2} and T_{req1} will be disabled in this concurrent enabling. Furthermore, the state numberings i have been changed in order to be sorted in the sets of vanishing $\hat{V} = \{M_0, M_1\}$ and tangible states $\hat{V} = \{M_2, .., M_6\}$.

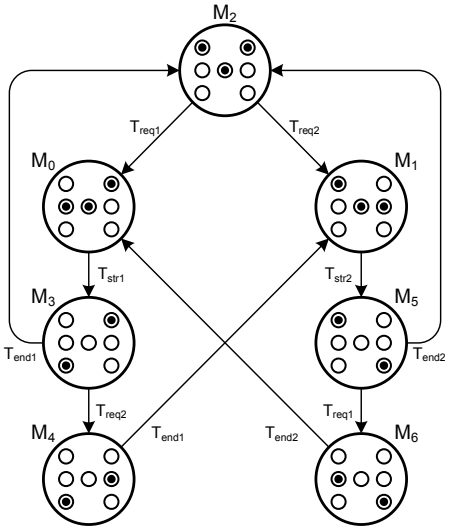


Figure 2.13: GSPN Example - Reachability Graph

The matrix P is then calculated as:

$$P = \begin{bmatrix} C & D \\ E & F \end{bmatrix} = \left[\begin{array}{cc|ccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \frac{\lambda_1}{\lambda_1+\lambda_2} & \frac{\lambda_2}{\lambda_1+\lambda_2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\mu_1}{\lambda_2+\mu_1} & 0 & \frac{\lambda_2}{\lambda_2+\mu_1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\mu_2}{\lambda_1+\mu_2} & 0 & 0 & 0 & \frac{\lambda_1}{\lambda_1+\mu_2} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

If the global balance equations $\tilde{\pi}P = \tilde{\pi}$ and $\sum_{M_i \in \hat{T} \cup \hat{V}} \tilde{\pi} = 1$ are then solved, $\tilde{\pi}_i$ is defined as:

$$\tilde{\pi}_0 = \frac{8}{63};\ \tilde{\pi}_1 = \frac{13}{63};\ \tilde{\pi}_2 = \frac{5}{18};\ \tilde{\pi}_3 = \frac{8}{63};\ \tilde{\pi}_4 = \frac{4}{189};\ \tilde{\pi}_5 = \frac{13}{63};\ \tilde{\pi}_6 = \frac{13}{378}.$$

This leads to the steady state distribution π (computed through formula 2.60):

$$\pi_0 = 0;\ \pi_1 = 0;\ \pi_2 = \frac{175}{277};\ \pi_3 = \frac{20}{277};\ \pi_4 = \frac{4}{277};\ \pi_5 = \frac{65}{277};\ \pi_6 = \frac{13}{277}.$$

While this example is also used as a comparative example in section 4.3, further performance values are calculated there.

2.2.4 Product Form Petri Nets

There are several proposals for product form solutions for Time Augmented Petri Nets in order to address the state-space explosion problem. The approach of Lazar and Robertazzi [94] is based on investigations of the reachability graph of the SPN. Henderson et al. [19, 20, 35, 46, 70, 72, 73] perform a structural analysis on the net level and Florin and Natkin [52] developed a product form solution for closed synchronized queueing networks [46, 52]. Furthermore,

Balbo, Bruell and Sereno [5, 6] proposed product form solutions for GSPNs. In this subsection the approaches are briefly explained. The product form solutions of Queueing Petri Nets [11, 12] are furthermore summarized in section 2.2.5.

Lazar and Robertazzi [94] characterize a product form Petri Net through the investigation of the corresponding reachability graph. They define an *isolated circulation* as [46, 94]:

Isolated Circulation An isolated circulation consists of the probability flow along a subset of edges of the state transition diagram for which there is a conservation of this flow at each adjacent state.

Through the identification of isolated circulations they define a product form criteria called *Duality Principle* [46, 94]:

Duality Principle There is a duality between the existence of local balance and the existence of isolated circulations. That is to say, the existence of local balance leads to isolated circulations and vice versa.

While the exploration of the state-space is a weakness of the approach [46], Lazar and Robertazzi also defined a product form criteria on structural level for safe nets[6] [46, 94]:

A Safe SPN, consisting of a number of task sequences that are comprised of a series of sequential sub-tasks, has product form solution for the equilibrium state probabilities if the state transition diagram can be naturally associated with a Cartesian coordinate systems, and if the transition diagram is comprised of integral building blocks (isolated circulations) and corresponding consistent set of local balance equations.

This criteria could also be formulated in terms of blocking at the state space level [46, 94]:

Consider a safe SPN consisting of a number of task sequences that are comprised of a series of sequential sub-tasks. The product form solution for the probabilities at equilibrium exists if and only if a task sequence is only allowed to proceed if there is a non-zero probability that it can return to its current state without the need for a state change in other task sequences.

Henderson, Lucic and Taylor [73] proposed a structural approach for the identification of product form Petri Nets. They consider the transitions of the stochastic Petri Net to be themselves states in a Markov chain, called *Routing process*. Each transition t has a unique input bag ($I(t)$) and the probabilistic routing allows for a number of different output bags ($O(t)$). Furthermore they define the set $R(T)$ [46, 73]:

$$R(T) = \bigcup_{t \in T} O(t) \cup \bigcup_{t \in T} I(t) \tag{2.61}$$

as the finite set of all vectors which are either output or input bags for the SPN. Then they define one step probabilities on the set $R(T)$ as [46, 73]:

$$p(r, r') = p(I(t), t, O_j(t)) \tag{2.62}$$

whenever there exists a transition t and a j such that $r = I(t)$ and $r' = O_j(t)$ (where no vector can be the input set of two or more transitions). Then they define the set F as [73]:

[6] Safe Net [46]: In a safe net, all places are 1-bounded.

$$F = \{f(\cdot) : f(r) > 0, \chi(r)f(r) = \sum_{r'} \chi(r')f(r')p(r', r), \forall r \in R\} \tag{2.63}$$

where $\chi(r) = \omega(r)$ if $r = I(t)$ and $\chi(r) = 1$ else [46].

Then, F needs to be nonempty, where [73]:

- For F to be nonempty, all vectors $I(t)$, $t \in T$ must be in positive recurrent communication classes of the routing process.
- For F to be nonempty, all $r \in R$ must be the input bag for some transition t and all $r \in R$ must be the output bag for some transition t.
- If F is nonempty, there exists a one to one correspondence between distinct elements of R and elements of F.

Through the one to one correspondence of R and T the set F could be defined as [46, 73]:

$$F = \{f(\cdot) : f(t) > 0, \omega(t)f(t) = \sum_{s} \omega(s)f(s)p(s, t), \forall t \in T\} \tag{2.64}$$

where $p(s, t) = p(I(t), I(s))$ with $s, t \in T$.

This leads to the product form theorem [46, 73]:

Assume, there exists a function $f(\cdot) \in F$ and a function $\{g(\cdot) : Z^P \rightarrow R\}$ which have the property, that for all $t \in T$ and $m + I(t)$ in the reachability graph

$$\frac{g(m + I(t))}{g(m + I(s))} = \frac{f(t)}{f(s)} \tag{2.65}$$

whenever $p(t, s) > 0$. Then the equilibrium distribution of the SPN is given by:

$$p(m) = G * g(m) \tag{2.66}$$

where G is a normalization constant.

In order to check if a SPN has a product form solution it is necessary [46]:

- to check the input output criteria and
- to find if $g(\cdot)$ exists (analytically or algorithmically (reachability graph)).

In [72] Henderson and Taylor further extended their results by analyzing a general discrete time model, finding joint equilibrium distributions at adjacent time points, allowing for arbitrary distributions of firing and enabling time periods and using the discrete time model as a basis for an embedded SPN analysis.

Boucherie and Sereno [19, 20], Donatelli and Sereno [46] and Coleman, Henderson and Taylor [35] further extended and generalized these results.

Through this research, a criterion based on minimal closed support T-invariants was established. Let $x_1, .., x_h$ denote the minimal support T-invariants[7], then [6, 20]:

Closed Set For $T' \subseteq T$ define $R(T')$, the set of input and output bags for the transitions in T', as $R(T') = \bigcup_{t \in T'} \{i(t) \cup O(t)\}$ is a closed set if for any $g \in R(T')$ there exist $t, t' \in T'$ such $g = I(t)$, as well as $g = O(t')$, that is if each output bag is also an input bag for a transition in T'.

Then the following structural constraint is the key to the identification of a product form SPN [6, 20, 70]:

Structural Constraint An SPN is said to be closed iff $\forall t \in T$ are covered by minimal closed support T-invariants, i.e. assume, that $\forall t \in T$ there exists an $i \in \{1, .., h\}$ such that $t \in ||x^i||$ and $||x^i||$ is a closed set.

If this structural constraint is true, there exists a positive solution of the traffic equations [20]. These traffic equations of the routing process [73] are the global balance equations of the corresponding Markov chain [70] and calculate the visit ratios $v(I(t_j))$ as follows [70]:

$$v(I(t_j)) = \sum_{t_h \in T} v(I(t_h)) p(I(t_h), I(t_j)) \, , \forall T_j \in T. \tag{2.67}$$

Following Haddad et al. [70], a positive solution for the traffic equations is not a sufficient condition to assert a product form solution for an SPN, but a first step in the finding of a product form solution [6]. In order to state, that an SPN has a product form solution, one other condition [35, 73] has to be fulfilled.

Product Form for Equilibrium Distribution of SPN [35, 70, 73] Let $f = v/\mu$ with v a solution for the traffic equations. The equilibrium distribution for the SPN has the form:

$$\pi(M) = \frac{1}{G} \prod_{i=1}^{|P|} y_i^{m_i} \; \forall m \in RS(M_0) \tag{2.68}$$

if and only if $Rank(C) = Rank(\left[C|w_f\right])$ where $\left[C|w_f\right]$ is the incidence matrix C augmented with the row w_f and G as a normalization constant. In this case the $|P|$-component vector $r = \left[log(y_1), .., log(y_{|P|})\right]$ satisfies the matrix equation $-r.C = w_f$.

with [70]:

$$w_f = \left[log\left(\frac{f(I(t_1))}{f(O(t_1))}\right), .., log\left(\frac{f(I(t_{|T|}))}{f(O(t_{|T|}))}\right) \right] \tag{2.69}$$

So it must be noted [70], that the condition $Rank(C) = Rank(\left[C|w_f\right])$ depends on the rates and not only on the structure of the net.

[7] T-invariant [20]: A vector $x \in \mathbb{N}_0^M$ is a T-invariant if $x \neq 0$ and $Ax = 0$. The *support* of a T-invariant x is the set of transitions corresponding to non-zero entries of x and is denoted by $||x||$, i.e. $||x|| = t \in T | x_t > 0$. A T-invariant x is *minimal*, if there is no other T-invariant x' such that $x'_m \leq x_m \; \forall m$. A support is minimal if no proper nonempty subset of the support is also a support of a T-invariant.

Based on this results, Haddad, Moreaux, Sereno and Silva [70] established a polynomial time algorithm ($O(|T|^2)$ to decide whether a SPN has a product form solution. Furthermore they developed a rate independent structural characterization of product form SPN. Also, they investigated some untimed properties and some reachability properties of product form SPNs.

Florin and Natkin [52] developed a product form solution for a special class of SPN called closed synchronized queueing networks. Closed synchronized queueing networks have the following properties [52]:

- The underlying Petri Net is a monovaluated place-transition net[8].
- The underlying Petri Net is bounded for its initial marking[9].
- The reachability graph of the underlying Petri Net is strongly connected (for ergodic properties).
- The stochastic Petri Net is a Markov Stochastic Petri Net.
- The firing rates of the transitions are marking independent.

The product form solution for closed synchronized Queueing Networks is based on results of Gordon-Newell [64] and matrix product form solutions.

Balbo, Bruell and Sereno [5, 6] extend the investigations on product form solutions for generalized Stochastic Petri Nets. They state, that every GSPN that satisfies the known structural constraint:

Structural Constraint [6] A GSPN is said to be closed iff $\forall t \in T$ there exists a minimal T-semiflow, such that $t \in ||x||$ and $||x||$ is a closed set.

and an additional criteria:

Free-Killing-Conflict [6] A GSPN is said to be a free-killing-conflict if any extended conflict set $ECS(t_i)$[10], that involves immediate transitions having the same priority level has the following property:

$$\bullet t_{i_a} \cap \bullet t_{i_b} \neq 0 \; or \; \bullet t_{i_a} \cap \circ t_{i_b} \neq 0 \; or \; \circ t_{i_a} \cap \circ t_{i_b} \neq 0 \; \forall t_{i_a}, t_{i_b} \in ECS(t_i) \tag{2.70}$$

Which means, that the firing of an enabled immediate transition disables all the other (enabled) transitions of its same $ECS(t_i)$ where $\bullet t_i = \{p_j | I(t_i, p_j) > 0\}$ is the *preset* of the transition t_i and $\circ t_i = \{p_j | H(t_i, p_j) > 0\}$ is the *inhibition set* of the transition t_i

admit a solution for its traffic equations and are therefore structurally suitable for a product form solution. To determine weather a GSPN admit product form solution, Balbo, Bruell and Sereno [5, 6] apply the results of [35, 70].

[8]Monovaluated Place-Transition Net [52]: In a monovaluated place-transition net, tokens can be added or removed from a place only one by one - non grouped arrival or departure from queue.

[9]Bounded [105]: A Petri Net is said to be bounded, if the number of tokens in each place could not exceed a finite number k for any marking reachable from the initial marking.

[10]Extended Conflict Set [6]: The extended conflict set $ECS(t_i)$ of a transition t_i includes all the transitions that are in structural conflict with t_i as well as those firings may disable t_i (indirect disable).

2.2.5 Queueing Petri Nets (QPN)

Queueing Petri Nets (QPN) [9, 10, 13, 14] combine Time Augmented Petri Nets and the Queueing Theory from the perspective of Time Augmented Petri Nets. In Queueing Petri Nets queues are integrated into colored GSPNs (CGSPN). In this kind of Petri Nets the type of *queueing places* is defined in addition to the *ordinary places*, in order be able to compactly define different scheduling and queueing strategies. The queueing places consist of a queue and a depository for tokens, as illustrated in figure 2.14.

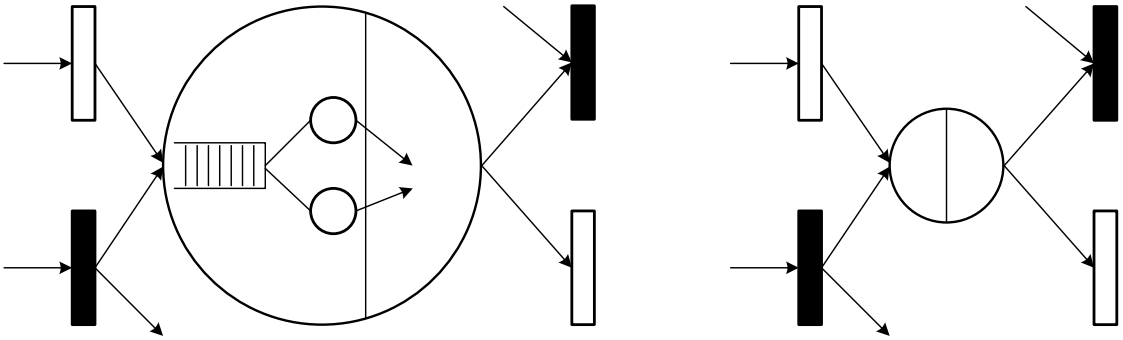


Figure 2.14: Queueing place and Queueing Place Shorthand Notation [14]

If a token is fired onto this place, the token is unavailable to all connected transitions until the service in the queue was not finished. When the service, according to the Kendall Notation of the queue (like described in section 2.1.3), is finished, the token is transmitted to the depository and is then available for all connected transitions. Through this extension of the colored GSPN, scheduling strategies could be integrated into the definition of the net without the need of a complex GSPN notation of the defined strategy [14].

An exemplary QPN is illustrated in figure 2.15.

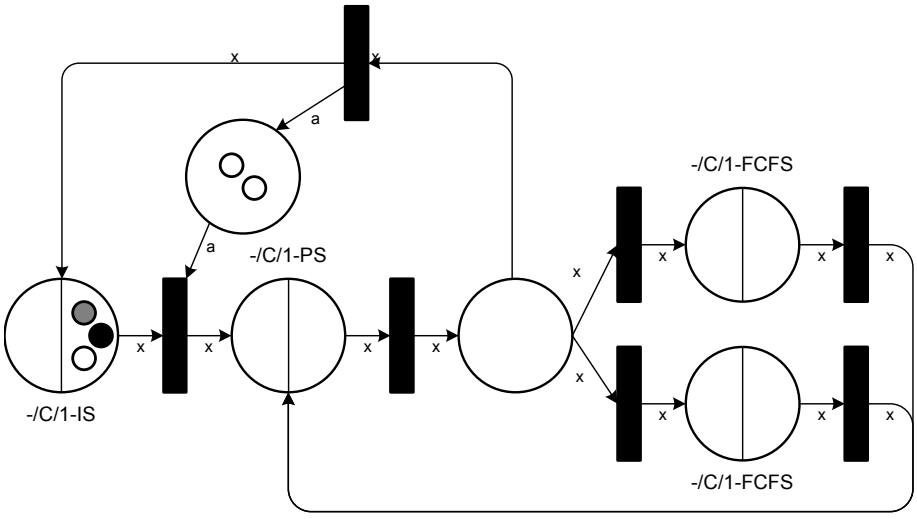


Figure 2.15: QPN Example [14]

For the analysis of quantitative system properties, QPNs must often be analyzed by inspecting the reachability set of the corresponding colored GSPN where the queue is modeled with CGSPN elements [14]. Though this limitation, they still suffer from the state space explosion problem.

Product Form Queueing Petri Nets

One approach to cope the state space explosion problem in QPNs, is to introduce product form solutions for Queueing Petri Nets [11, 12]. A QPN is defined by a triple $(CGSPN, P_1, P_2)$ where $CGSPN$ is a colored GSPN, P_1 is the set of queueing places and P_2 is a set of ordinary places ($P = P_1 \cup P_2$ is the set of places of the CGSPN). Furthermore, the $CGSPN$ is a 4-tuple (CPN, T_1, T_2, W), where CPN is the underlying Coloured Petri Net, $T_1 \subseteq T$ is the set of timed transitions, $T_2 \subseteq T$ is the set of immediate transitions ($T = T_1 \cup T_2$ is the set of transitions of the CPN) and $W = (\omega_1, .., \omega_{|T|})$ is an array whose entries are possibly marking dependent firing rates of timed transitions or firing weights of immediate transitions [11]. The QPNs, for which a product form solution was established, is then defined as follows [11]:

Product Form QPN [11] A product form QPN is a QPN with $P_1 \cup T_1 \neq 0$ satisfying the following restrictions:

1. All transitions are uncolored.
2. The input bags of immediate transitions are disjoint or equal and do not intersect with the input bags of timed transitions.
3. The relative firing frequencies of immediate transitions do not depend on the marking of the QPN.
4. Output places of immediate transitions are not input places of immediate transitions.
5. No two timed transitions have the same input bag.
6. The input bag of each transitions is the output bag of some other transition and vice versa.
7. The marking dependent firing rate of each timed transition $t \in T_1$ can be expressed as $r(M,t) = \frac{\varphi(M-I(t))\chi(t)}{\phi(M)}$, where φ, χ and ϕ are arbitrary non-negative functions.
8. Input transitions of queueing places have no further output places, output transitions of queueing places have no further input places and only single tokens arrive at and leave from a queue.
9. All queues in queueing places $p \in P_1$ are of product form type.

The solution of such product form QPNs follows ideas of Henderson et al. [35, 72, 73] and BCMP Networks [8]. Through restrictions 2-4 all immediate transitions could be eliminated. Restrictions 5-7 are the restrictions for product from SPNs. Through restriction 8 each queueing place could be replaced by its simplified interpretation, and through restriction 9 the global balance equations reduce to the defining equations for function f [11]:

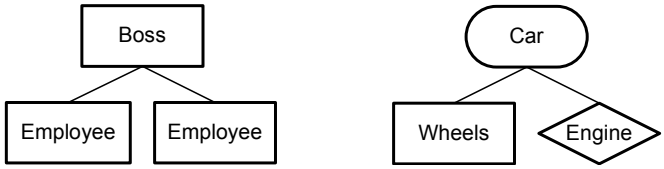
$$F = \left\{ f : T \to (0,\infty) : \chi(t)f(t) = \sum_{s \in T} \chi(s)f(s)p(s,t), \forall t \in T \right\} \tag{2.71}$$

So the product form QPN could be transformed into a product form CGSPN and furthermore an product form SPN (*SPN-skeleton*), and therefore product form Queueing Networks and product form SPNs are special cases of product form QPNs [11]. The calculation of the performance values could then be further optimized by aggregation and disaggregation methods like described in [11] and [12].

2.3 Quantitative Hierarchical Modeling

Hierarchical modeling is the key to cope with complexity in the modeling of quantitative systems. Therefore this section gives an overview of related work in the area of quantitative hierarchical modeling.

In order to classify the different approaches and to clarify the understanding of hierarchies, some definitions in are given in advance: There are different kinds of hierarchies: organizational hierarchies with super-/subordinate relations and abstraction hierarchies with compose/decompose relations [143]. In organizational hierarchies one instance is the superordinate to a hierarchically lower instance. As shown in figure 2.16(a), in this kind of hierarchies the elements are of the same type - both the *boss* and the *employee* are persons and they do not compose or decompose to each other. In contrast to this, in the abstraction hierarchy, some parts are composed to a whole. The elements are of different types, as illustrated in figure 2.16(b). A *car* is composed by it's parts, so the parts are different from the car and the car is different from the single parts. Also in the abstraction hierarchy the upper hierarchical instances not necessarily have to exist (as they could be *abstract*).



(a) Elements of same Type Organizational Hierarchy

(b) Elements of different Type Abstraction Hierarchy

Figure 2.16: Hierarchies

Furthermore, these two hierarchies could also exist in parallel. As an example, a service request, handled by a boss could be further decomposed to sub-requests, handled by the employee. In this case there is a relation between the different views, as the domain representation of the boss as the handler of the request is decomposed into the sub-requests handlers, represented by the employees. In this abstraction hierarchy the different elements are not the same, as the the boss is interpreted as the handler and the employees are interpreted as subhandlers. But nevertheless, the organizational hierarchy *boss-employees* still exists in parallel, as the different instances are not only abstract entities.

The decomposability in subsection 2.3.1 is an abstraction hierarchy, as parts of the systems are decomposed into sub-parts. Norton's Theorem, described in subsection 2.3.2, could also be used in the development of abstraction hierarchies (or model abstractions). The (sub-)parts or components of the system have to be carefully chosen in order to define correct hierarchies and not only a flat refinement of the system or a system with wrong hierarchical borders or hierarchy violations. The models and techniques shown in subsection 2.3.3 and 2.3.4 also try to decompose the models represented as Queueing Networks or Petri Nets into smaller subsystems in order to reduce the complexity of the model analysis. These hierarchies are model abstraction hierarchies. The Forced Traffic Flow Law, summarized in subsection 2.3.5, as well as the Layered Queueing Networks (LQN), summarized in subsection 2.3.6, with a decomposition of a system into components or nested requests is also aimed to be an abstraction hierarchy. But, as the right hierarchies are crucial for a correct system understanding, one has to be careful to model the right hierarchies and hierarchy transformations.

2.3.1 Decomposability

Simon and Ando [122] introduced decomposability and nearly-decomposable systems in order to support aggregation of variables in the analysis of large and complex systems. Through the aggregation of variables on different levels, the overall state-space of the model and therefore the complexity of the evaluation of such system could be reduced. Completely decomposable systems [38] are systems in which state variables could be classified in groups where:

- interactions within groups can be studied as if interactions among groups do not exist and
- interactions among groups can be analyzed without referring to the interactions within groups.

This hypothesis is correct [38], but often to rigorous. As mentioned, Simon and Ando [122] introduced nearly completely decomposable systems. In the analysis of such systems, there are good approximations [38] when interactions among groups are weak compared to the interactions within groups. Simon and Ando [122] showed that in such systems short-run dynamics and long-run dynamics can be distinguished:

- short-run dynamics represent the interactions of the variables within each subsystem and
- long-run dynamics represent the interactions among the subsystems.

Therefore, the subsystems could be analyzed by local equilibriums and the whole system could be analyzed by aggregate variables and a global equilibrium.

Courtois [38, 40] transferred the results of Simon and Ando to Queueing Networks and computer performance analysis. He also introduced a hierarchy of aggregate variables in order to cope with the complexity of large systems. Through the dissection of systems into subsystems with different models, the different models could be evaluated separately and through the representation of the subsystems in aggregate variables, these systems could be analyzed at different levels. Courtois also mentioned that, through this technique and the related state-space partitioning, it is possible to analyze the different subsystems with different methods like Queueing Theory, simulation and deterministic models.

In the hierarchical aggregation technique of Courtois there remains a known and limited approximation error [39, 126]. This error is in the same order of magnitude as the maximum degree of coupling between the different aggregation variables. As a result of this the aggregation variables or systems should be well conditioned and indecomposable [39]. Courtois also introduced multilevel aggregation [39].

A special case in the aggregation of matrices are nearly-completely decomposable matrices which are also block stochastic [39]. Here the eigenvalues of the aggregates matrices are summations of the eigenvalues of the original matrices, which yields to exact values for the steady state probabilities. Then the remaining error results only from the aggregation of stochastic matrices to subsystems.

Based on this results, Vantilborgh [126] introduced conditions under which the aggregation yields to exact results. He states and proves that the Perron-Frobenius eigenvector[11][12] of an

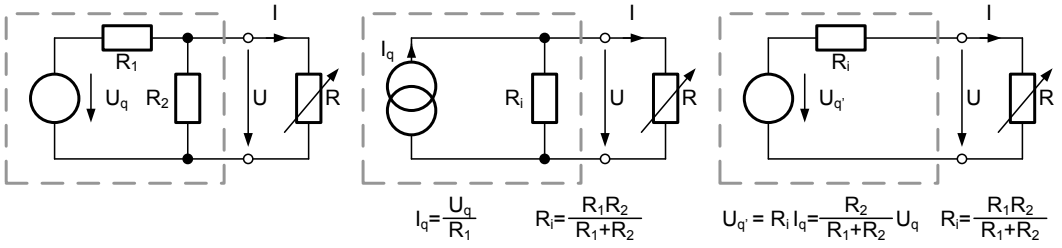
[11] Eigenvector [22]: For the square matrix A of type (n, n) the value λ is the eigenvalue and the vector $\vec{x}$ ($\vec{x} \neq 0$) is the eigenvector, calculated as: $A\vec{x} = \lambda\vec{x}$.

[12] Perron-Frobenius Theorem [111, 123]: The Perron-Frobenius Theorem deals with positive matrices ($A > 0$) and their eigenvalues.

aggregated matrix is correct if, and only if the Perron-Frobenius eigenvectors of the non aggregated base matrices are subparallel[13] to the Perron-Frobenius eigenvector of the aggregated matrix. The outcome of this is that the aggregative analysis of a system through the successive analysis of the subsystems yields exact results for all steady-state distributions, if and only if the Perron-Frobenius eigenvectors of the customer behavior matrices of the subsystems are subparallel to the Perron-Frobenius eigenvector of the customer behavior matrix of the aggregated system. Vantilborgh also proves that the equilibrium distribution of the aggregated system is equal to the conditional distributions on the servers of the subsystems conditioned on the customers served of queued at the servers, if and only if the Perron-Frobenius eigenvectors of the subsystems are subparallel to the Perron-Frobenius eigenvector of the aggregated system.

2.3.2 Norton's Theorem

Norton's Theorem of the electrical circuit theory states that in an electrical circuit of voltage and current sources and resistors (passive components) it is possible to replace a subsystem by a single current source and a parallel internal resistance with the same 'equivalent' behavior, as shown in figure 2.17. In this example figure 2.17(a) shows the original circuit, figure 2.17(b) shows the current source equivalent or Norton Equivalent Circuit and, additionally in figure 2.17(c) the voltage source equivalent or Thévenin Equivalent Circuit is shown.



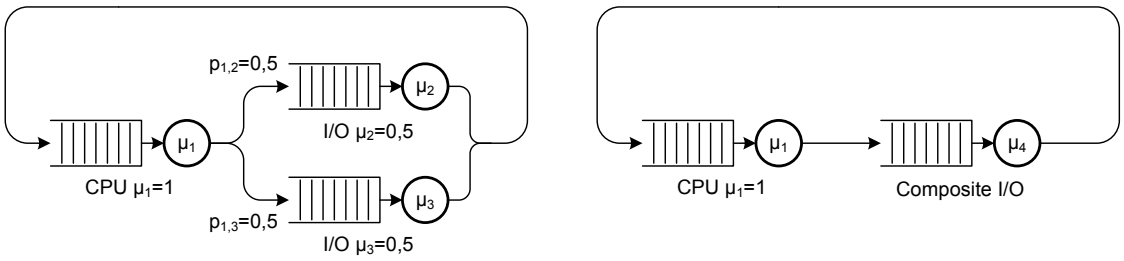
(a) Original Circuit (b) Current Source Equivalent (Norton Equivalent Circuit) (c) Voltage Source Equivalent (Thévenin Equivalent Circuit)

Figure 2.17: Norton's Theorem [41]

Referring to [81] this 'current source equivalent' was published in parallel in 1926 by Edward Lawry Norton [107] and Hans Ferdinand Mayer [103], who called this 'Ersatzschema' or 'Ersatzschaltung' (equivalent circuits). Both results are based on the studies on Hermann von Helmholz and Léon Charles Thévenin, who also were apparently unaware of the work of each other. Therefore the Norton Theorem sometimes is also called Helmholtz-Thévenin, Helmholz-Norton or Mayer-Norton Theorem [81].

Chandy, Herzog and Woo [30] adapted this results for the use in Queueing Networks in order to replace a subsystem by a single composite queue, as shown in figure 2.18.

[13]Subparallel [126]: An m-element vector v is subparallel to a vector $u = [u_1 u_2..u_n]$, $n > m$ if there exists a scalar k such that $v = k[u_1 u_2..u_n]$



(a) Original Network (b) Transformed Network

Figure 2.18: Norton's Theorem for Queueing Networks - Example [30]

They proved Norton's Theorem for closed Gordon-Newell Networks and showed that it is also applicable to open networks and networks which satisfy local balance. With this results it is possible to replace a subsystem by a composite queue in order to simplify the analysis of the rest of the system. They proved that for closed networks the queue length distribution for a queue in an equivalent network is the same as in the given network, and the queue length distribution at arrival for a queue in the equivalent network is also the same as in the given network. For open networks with exponential service times and Poisson arrivals they state that the queue length distributions for all queues in a subsystem in the equivalent network are the same as in the given network. They also show that Norton's Theorem holds for the class of networks which satisfy local balance. Here in a closed network the service rates at the composite queue is set equal to the throughput to the short with the same number of service requests in the short. For open networks all uninteresting queues in the subsystem are replaced by a single composite Poisson source which generates service requests of all classes, where each class is generated independently. The generation rates are set equal to the throughput through the short of the subsystem under consideration.

Balsamo and Iazeolla [7] extended Norton's theorem for the use of any number of queues and arbitrary interface between the subsystem and the rest of the system. They proposed a solution for BCMP-Networks [8] with one class of customers (extension to several classes is immediate [7]). They first constructed a closed network, called the *reduced network*, where the servers of the subnetwork are shorted (the service times are set to zero) and the service rate of the composite queue is set to the throughput of the original network with the same number of service requests in the network. They proved that the marginal probabilities for the queues and the queue length distributions at arrival on the queues are the same for the original and the equivalent network. They also described an algorithm for the solution of the reduced network and the equivalent network.

Walrand [127] proved that Norton's Theorem is also true for multiclass quasi reversible networks.

The results of the transmission of Norton's Theorem to Queueing Theory are interesting in the development of aggregation methods in the hierarchical modeling and analysis, as well as the development of model transformations. However, in the related work discussed in this section Norton's Theorem is used for the study of a subsystem without solving for the entire network by means of a model transformation. The idea is to replace the uninteresting parts of the entire model by the equivalent model and to study the behavior of the rest of the model then - hierarchies are not mentioned there. When considering hierarchies, some questions arise. Taking a deeper look in the model understanding, Norton's Theorem is clear to understand for electrical circuits, because electricity knows nothing about service requests and service request hierarchies - it is just a physical flow balance. But, when considering Queueing Networks with

service requests and service request hierarchies, these hierarchies and hierarchical transformations have to be considered. Service requests have to be served by specialized servers and in order to define sub-requests, service requests have to be transformed. Also at branches electricity is driven by resistances of the different paths, whereas the path of the service requests are defined by control flow decisions (or simplified by probabilities). But anyhow, the usage of Norton's Theorem in the Queueing Theory could lead to new ideas and formulas in the development of transformations, but, correct model and service request transformations always have to be considered in order to reach to meaningful results.

2.3.3 Formal Hierarchies and Combination of Models

Malhotra and Trivedi [100] propose a formal expression of abstraction hierarchies in model specification and solution. In this methodology the models are a composition of different hierarchical models. Every model is treated as a black box, and in the hierarchical overall model the input and output variables are combined with the help of an interconnection matrix. Though the treatment as a black box and the output and input connection many different types of sub models are possible. If parameters are passed from model M_i to model M_j, an order $\succ$ is defined. The relation $\succ\succ$ further defines the transitive closure between the models. If the overall model is acyclic (If $M_i \succ\succ M_j$ then not $M_j \succ\succ M_i$), the solution of the overall model is achieved through the solving of the model from the most-inner sub model to the overall model. If the overall model is non-acyclic, an iterative solution is possible for the solution of the overall model.

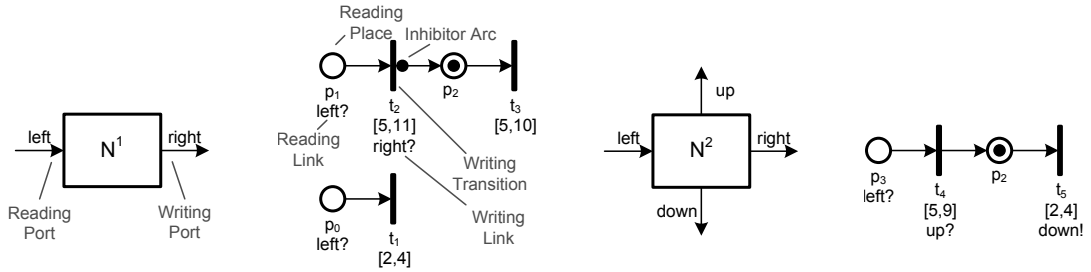
Bause and Buchholz [12] use aggregation and disaggregation in their Product Form Queueing Petri Nets [11], described in section 2.2.5. Their results for the exact aggregation are based on Norton's Theorem.

Balbo, Bruell and Ghanta [4] propose a technique in which Queueing Network models and Generalized Stochastic Petri Nets are combined in a hierarchical modeling approach. In their approach the hierarchical subsystems, which do not violate the BCMP-Theorem [8], are modeled as Product Form Queueing Networks (PFQN). Subsystems, which violate the BCMP assumptions, like systems with synchronization of processes or simultaneous possession of resources by a process, are modeled as Generalized Stochastic Petri Nets (GSPN [102]). Later the different results are combined to an overall solution. Through this combination it is avoided that the whole system is modeled as a GSPN model which results in a state space reduction. In this approach every subsystem is evaluated in isolation and the results are then aggregated. They refer to the decomposability approach of Courtois [38, 40] and the approximation error of this approach. They [4] state that, through the solving of the single sub models in isolation with the appropriate exact model, only the aggregation would lead to an approximative error.

2.3.4 Hierarchies in Time Augmented Petri Nets

Bucci and Vicario [23] propose Communicating Time Petri Nets (CmTPN). CmTPNs augment the basic model of Petri Nets with inhibitor arcs, the timing constraints of Time Augmented Petri Nets, and a module construct which permits the decomposition of a complex model into smaller sub models [23]. This module construct consists of *writing* and *reading ports* connected through *communication links* to transitions and places that are referred as *writing transitions* and *reading places*. An example in given in figure 2.19, showing two modules N^1 and N^2, where each module has a reading port *left* and a writing port *right*, respectively *up* and *down*. In the

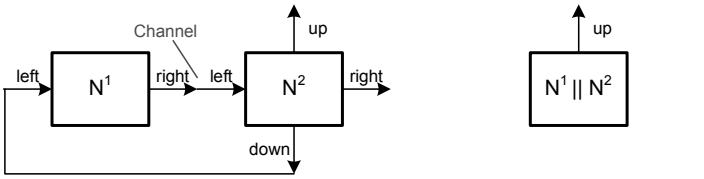
internal view it can be seen that through the reading links *left?* and the writing links *right?*, *up?* and *down?* these ports are connected to the reading places p_0, p_1 and p_3 as well as the writing transitions t_2, t_4 and t_5.



(a) Module N^1 (External View) (b) Module N^1 (Internal View) (c) Module N^2 (External View) (d) Module N^2 (Internal View)

Figure 2.19: Communicating Time Petri Nets Example - Modules [23]

Writing and reading ports of the different sub models can then be connected through *channels* to support the communication and interaction between the different sub models, as shown in figure 2.20, where the two modules of figure 2.19 are connected and composed in order to integrate them in a larger scenario.



(a) Connection Diagram (b) Composed Module

Figure 2.20: Communicating Time Petri Nets Example - Composition [23]

The analysis of such models consists of two steps: *unit analysis* and *integration analysis*. In the unit analysis the different subsystems are analyzed. The different sub models are separated from each other and communicate only through interfaces. Through that interfaces the environment of each sub model is defined. Then every sub model is analyzed similar to a Time Augmented Petri Net. In the integration analysis the whole model is computed by aggregating the results of the different subsystems.

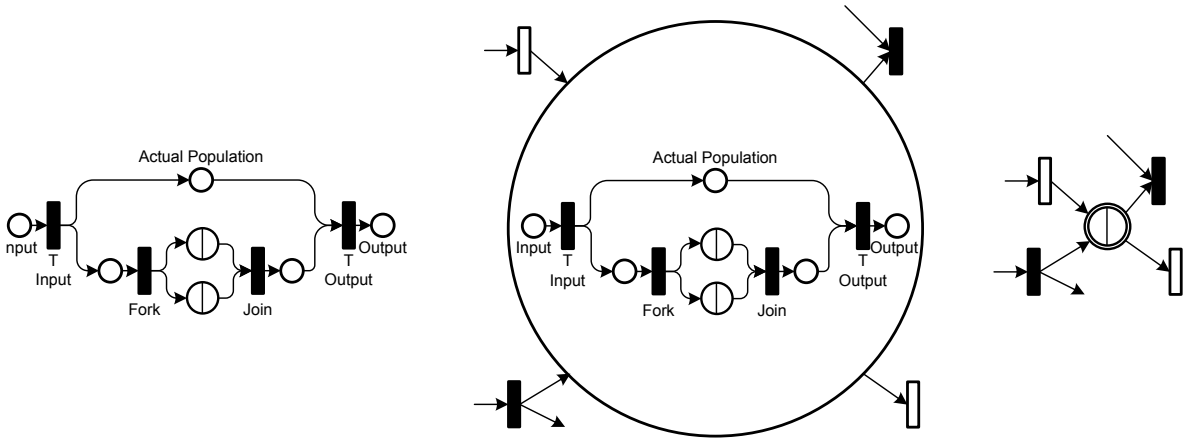
Haddad and Moreaux [69] combine aggregation and decomposition in order to evaluate Petri Nets. In their understanding aggregation reduces the state space by grouping states and solving the corresponding Markov chain on the set of state classes. In their decomposition method, they follow the decomposition of Plateau and Fourneau [114] (referring to [69]), in which the state space is a cartesian product of smaller state spaces. In their article they also propose the concept of *internal* and *external synchronization*, where internal synchronization means synchronization within one class and external synchronization means synchronization between several classes.

Buchholz [24] proposes the hierarchical structuring of Superposed Generalized Stochastic Petri Nets (SGSPN). In his work he presents a technique in which a hierarchical structure is generated in a preprocessing step in order to compute a compact generator matrix. Through this

approach the problem of unreachable states is avoided. In the approach *macro states* and the generation of a *macro transition system* are proposed. Macro states combine states of a component state space, and a set of macro states defines a partition of the component state space. Each macro state represents a set of detailed states. In the macro transition system the state transitions of one macro state to another (not locally) are defined. Through this macro transition system a *global reachability analysis* is performed. The unreachablility of a global macro state implies the unreachability of a detailed state, which allows the exclusion of unreachable states.

Freiheit and Zimmermann [60] propose a methodology for the automatic decomposition of models. Their results are based on the decomposition of Stochastic Petri Nets but could be extended to other modeling techniques. Their *divide and conquer approach* is divided into three steps: In the first step the system is decomposed into smaller subsystems. In the second step the dependencies between the different subsystems are derived and extracted in *low-level subsystems* and an aggregative *basic skeleton*. In the third step the performance values of the system are computed by an iterative approximation method.

Another approach to reduce the state-space explosion problem in Queueing Petri Nets (QPNs), beside the Product Form Queueing Petri Nets described in section 2.2.5, are hierarchically combined Queueing Petri Nets (HQPN) [15]. In HQPNs a timed place can contain a whole subnet. This subnet has a dedicated *input* and *output place*, where the tokens fired onto places in the subnet are delivered onto the input place and the output place is similar to the depository of a timed (queueing) place. An example is given in figure 2.21. In figure 2.21(a) an isolated HQPN subnet is specified. This subnet is then integrated into a larger subnet in figure 2.21(b). Finally, this larger subnet is then represented by a shorthand notation in figure 2.21(c).



(a) Subnet (b) Integrated Subnet (c) Shorthand Notation

Figure 2.21: Hierarchically combined Queueing Petri Nets (HQPN) - Example [15]

For the calculation of the performance values of the whole net the state spaces and transition matrices for each subnet are generated in isolation [15], and then the overall state space could be determined through a combination of the sub states of the subnets. In [15] the authors state that through the hierarchical decomposition the calculation of the performance values could be reduced by one order of magnitude. But, referring to memory constraints in the calculation [15], an introduction of several hierarchical levels would not extend the size of the

solvable problems, and so the reduction of the computation complexity is, as stated, *one* order of magnitude.

2.3.5 Forced Traffic Flow Law

The Forced Traffic Flow Law (also called Forced Flow Law) [43, 68, 79, 95] is one of the main laws for hierarchical decomposition. Through the Forced Traffic Flow Law the hierarchical service request transformation of the server requests into subrequests is possible, which enables the hierarchical decomposition. In the Forced Traffic Flow Law an arrival rate λ will be transformed into an arrival rate λ_i through the use of a traffic flow coefficient v_i:

$$\lambda_i = v_i * \lambda \tag{2.72}$$

In Denning and Buzen [43] the Forced Traffic Flow Law is used for the decomposition of the overall jobs into requests. They define units for the service request in the definition of rates like service rates. In their two level models the overall service request is called 'job' and the sub requests are called 'request'. In their definition of the operational analysis the traffic flow coefficient is called visit ratio and expresses the mean number of requests per job for a device.

Lazowska et al. [95] extend this and describe the specialization of the term 'request' on different levels of details. For example, a request at a disk is called disc access, and a request at the level of the entire system could be defined as a user-level interaction. They furthermore make use of the Forced Traffic Flow Law in combination with Little's Law for the calculation of performance values on different levels.

While the main idea of using the Forced Traffic Flow Law for hierarchical decomposition is very constructive, the main flaw in the approach of Denning and Buzen [43], as well as the approach of Lazowska et al. [95], is in the modeling only from the view of the static server structures. In these networks of servers and queues, as shown in the example in figure 2.22, the hierarchical control flow and hierarchical service request transformations are not modeled, like they could be modeled using (Time Augmented) Petri Nets. Through this inadequate representation the service requests have only a different naming at different servers, but there is no real transformation shown.

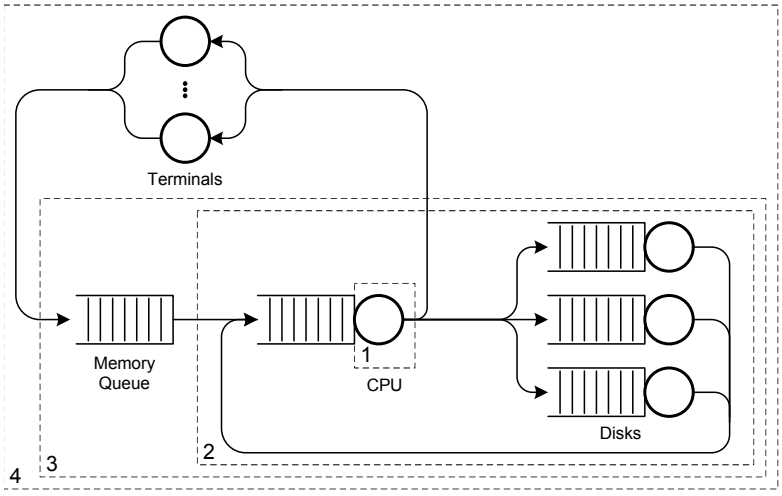


Figure 2.22: Memory Constrained System [95]

Although there are different levels (1-4) in the model in figure 2.22, the service request transformations are not visualized. Also the different visit ratios are not represented in this model. In this view the routing and admission of the overall request to the different servers is modeled. But, this view does not necessarily map to the real service request structures. Furthermore, as the service request transformations are not modeled, it is not clearly defined which kind of (sub-request) flows are in which part of the model. One could argue that inside every dashed box the corresponding service request type is existent, but then, in the example, the disk request is also in the queue of the CPU (which could be corrected by extending the level 1 box around the queue), but furthermore, the disk request is transfered back to the terminal. Where is the difference between the requests going out from the CPU and the requests to the Disks? In the mathematical model the flows could be correct, but viewing and questioning the model from the modeling view raises a lot of question like these.

Also in Haas and Zorn [68] the Forced Traffic Flow Law (there: 'Verkehrsflussgesetz') is used for the decomposition of a system into subcomponents. Through the decomposition the overall request is departed into subrequests at every component through the visit ratio V_i and the throughput D_i at every component could then be expressed as $D_i = V_i * D$. Furthermore, they define a distinction of the different throughputs for every service request class. But as in the former publications referenced in this subsection, the static server view is modeled as Queueing Networks of queues and servers without the modeling of the control flows, which raise the same questions.

2.3.6 Layered Queueing Networks (LQN)

Layered queues are an extended queueing network to handle nested multiple resource possession, where the nesting is defined by layers [59]. Referring to [59], their central idea is to model networks, where services may have nested services executed by another server, with nesting to any depth. While Layered Queueing Networks (LQN) [56, 59] subsume a lot of research in this area [59], is very interesting to investigate.

The Layered Queueing Networks (LQN) were introduced in 1996 by Franks et al. [56]. In [56] the basic LQN model and the LQN principles were discussed. Parallelism effects were added to the approach in [53]. Furthermore, the method of complementary delays [71] was integrated and extended to LQNs in [53]. In [54] two-phase servers are integrated into the LQN Mean Values Analysis calculation approach. Multiservers with several classes in LQNs were introduced in [55]. In [108] and [110] the replication of structures in LQNs is introduced. [109] concentrates on Dependability and the performance modeling of a quorum pattern. [59] gives an overview of the whole approach as an entry point for further investigations.

In LQN the layered service request structure is modeled with a specification of visits, customer classes with arrival rates or populations and devices with service disciplines. In LQN diagrams, as shown in figure 2.23, there is a distinction between logical servers and multiplexers, where the logical servers are named as software servers or tasks (parallelograms) and the multiplexers are named as hardware servers and are represented as circles. Tasks can make requests to any other entity.

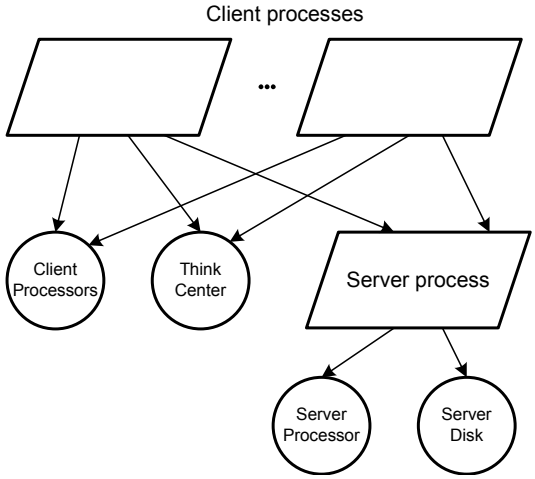


Figure 2.23: LQN Example - File Server Application - LQN [56]

The LQNs, the activities could be modeled in executions graphs which provide parallelism, sequence, branching and loops. An exemplary activity graph is shown in figure 2.24.

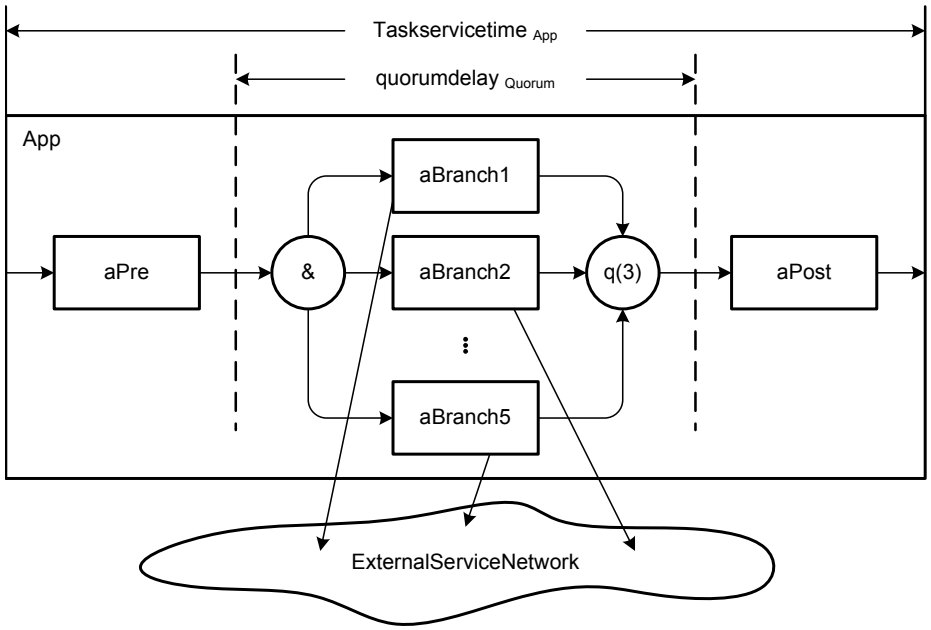


Figure 2.24: LQN Activity Graph Example - Quorum Consensus [109]

The dynamic behavior could also be modeled in sequence diagrams, as shown in figure 2.25. The different nested service calls to the different servers could also be interpreted as an organizational hierarchy, whereas here the service decomposition is mainly done from the view of the server structures and client-server / master-slave relations, which could be contradictory regarding to the real service request structures.

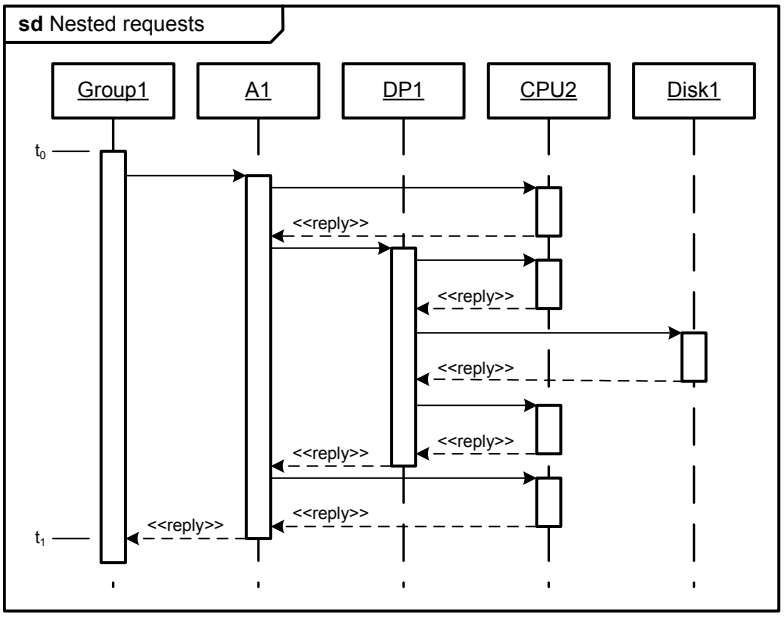


Figure 2.25: LQN Example - File Server Application - Sequence Diagram [59]

For the calculations of the performance values in LQN an LQN Solver (LQNS [57]) has been developed [59]. This solver derives the performance values by solving a set of related submodels, where each of the submodels is solved using a Linearizer algorithm of the Mean Values Analysis [59]. The algorithm of the LQNS is sketched as follows [59]:

LoadModel
ExtendModel
Topological Sort
Layerize (create and initialize layer submodels)
repeat
  Solve the layer submodels using Linearizer MVA
until converge or iteration limit
Save results

Where the solving of the submodels is sketched as [59]:

for all Clients do
  Calculate imported service and think times.
end for
for all Servers do
  Calculate imported mean and variance of service times.
end for
solve submodel using mixed − model MVA

2.4 Summary

The Queueing Theory, discussed in section 2.1, has strengths in the prediction of performance values, especially of Product Form Queuing Networks, through a sophisticated and mature mathematical background with a well defined set of formulas and fast algorithms. But, as the Queueing Theory focuses on the modeling of server structures and the request flow through this servers, the control flow is neglected in classical queueing networks. This leads to a lack of expressiveness in modeling power if there are a lot of control flows in the modeled system.

Time Augmented Petri Nets, described in section 2.2, model the systems from the perspective of the control flow and especially through Colored Generalized Stochastic Petri Nets or Queuing Petri Nets complex systems could be modeled. However, this often results in a state-space-explosion problem in the calculation of the performance values, and even in Product Form Petri Nets complexity is still a problem. Also through neglecting the server structures and only modeling the dynamic behavior and the control flow, the modeling of shared resources and specific scheduling strategies is problematic.

Hierarchies are the key to cope with complexity in the modeling of large systems. As described in section 2.3, there are some approaches to integrate hierarchies in quantitative modeling and evaluation. Anyhow, these approaches are rare and the important hierarchy law, the Forced Traffic Flow Law, is rather neglected.

The new methodology FMC-QE, described in the following chapters, tries to cope these problems in a hierarchical modeling from the perspective of the service requests. In section 4.4 the author will come back to some of the questions raised here and discuss them in comparison to FMC-QE.

Chapter 3

FMC-QE Fundamentals

This chapter describes the fundamentals of FMC-QE, the Fundamental Modeling Concepts for Quantitative Evaluation. It starts with a summary of the foundations of FMC-QE, FMC (the Fundamental Modeling Concepts) and FMC-eCS (the Fundamental Modeling Concepts extended for Communication Systems). After that some basic definitions are given in section 3.2. This includes the description of the term service request which is of main concern in FMC-QE. Main principles of hierarchical modeling in FMC-QE are also described beside the introduction of the main quantitative measures, defined in the scope of FMC-QE. In section 3.3 the graphical notations are introduced. This includes the service request structures, defined in Entity Relationship Diagrams, the static (server) structures, defined in Block Diagrams and the dynamic behavior including the control flows, defined in Petri Nets. After that, in section 3.4, the rules and formulas of the performance measures in the calculus and the derived Tableau are described. This includes fundamental laws, experimental parameters, the description of the Tableau including a short introduction into model transformations and a complexity analysis. In section 3.5 an Open Queueing Network is modeled with focus on the description of the transformations and the precision of the predictions for this class of problems. Model transformations in FMC-QE are only roughly discussed, while this topic is more focused by the work of Tomasz Porzucek [115, 116], another PhD-Student in the Research Group. His work is in the scope of the development of an FMC-QE Tool and in section 3.6 this is referenced.

3.1 Foundations

In this section the foundations of FMC-QE, the Fundamental Modeling Concepts (FMC) and the Fundamental Modeling Concepts extended for Communication Systems (FMC-eCS) are summarized.

3.1.1 Fundamental Modeling Concepts (FMC)

The Fundamental Modeling Concepts (FMC) are a modeling technique, developed to support the communication about information processing systems [92]. The beginnings of FMC are dated back to a workshop initiated by Siegfried Wendt in 1974 and were constantly evolved [92, 124, 130] and used [66, 82, 91]. The author of this thesis has received his FMC background through lectures within the software systems engineering studies and his master thesis, analyzing and modeling an Enterprise Resource Planning (ERP) System with FMC [90].

The human to human communication and the support in the understanding of information processing systems are in the focus of FMC and therefore the main concerns for this modeling technique are [92]:

- Abstraction,
- Simplicity,
- Universality,
- Separation of concerns and
- Aesthetics and secondary notation.

In this context abstraction means the ability to describe systems on different levels of abstraction in order to reduce the complexity of the models. Simplicity is achieved through restricting the modeling technique to a few fundamental concepts and notation elements in order to be able to create models ad-hoc for example in meetings. With FMC it is possible to model a broad range of systems without being bound to a specific paradigm. Separation of concerns for reducing the complexity and the ability of describing different aspects of a system are achieved through the three-dimensional modeling space of *compositional structures*, *behavior* and *data / value structures*. In FMC aesthetics and secondary notation are supported though modeling and visualization guidelines and easy formation of the graphical patterns [2, 92].

In FMC the systems are modeled in three different dimensions [92]:

- Compositional Structures,
- Behavior and
- Data / Value Structures.

The compositional structures are modeled in Block Diagrams, referring to [92], based on the German industrial standard DIN6620 [44]. FMC uses Petri Nets [75, 113] in order to describe the behavior of systems. Data and value structures are described in Entity Relationship Diagrams, originally defined by Chen [31]. In the following the different diagram types are shortly described:

Compositional Structures - Block Diagram

The compositional (static) structures are represented in Block Diagrams [92]. FMC Block Diagrams are bipartite graphs with the distinction of active and passive system components. Active components (Agents) are represented in rectangles, passive components (channels and storage) are represented as round elements. The active components process information, while on passive components, information is stored or observed. The passive components are further distinguished in non-volatile storages and volatile channels. The components are linked through directed and undirected edges, which represent write, read and read/write accesses. An example of an FMC Block Diagram is given in Figure 3.1.

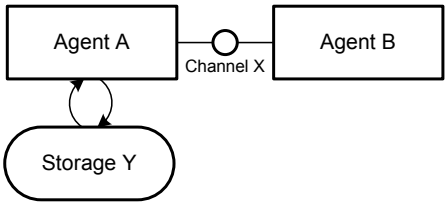


Figure 3.1: FMC Block Diagram - Example

In the example there are two active components *Agent A* and *Agent B* connected through a channel *X*. Additionally, *Agent A* has a modifying (read/write) access to storage *Y*.

Control Structures - Petri Net

The control structures of the modeled system are described in Petri Nets [92]. These bipartite *Event Condition* [75] nets are used to describe the sequence of actions and events [92] observed in the system. These actions happen in a certain temporal order (partial order because of concurrency) dependent on the state of the system and system specific rules [92]. Through Petri Nets these rules and the causal order of the events or actions could be described. In FMC actions are represented by a *transition* of a Petri Net, graphically represented by a rectangle. The circles in a Petri Net are the *places*. Places could be marked (with a dot inside) or remain empty. Places and transitions are linked through directed arcs. The corresponding action of the transition is performed if the transition *fires*. This is possible if [92]:

- all its input places (connected to transition - arc ends at transition) are marked and
- all its output places (connected to transition - arc starts at transition) are unmarked (empty) (strict transition rule [105]).

After the firing [92]:

- the corresponding action has been performed,
- all input places are unmarked and
- all output places are marked.

These rules could also be extended to places with a capacity greater than one and arc weights greater than one. Therefore see [92, 105].

If, in a case of a conflict (two transitions are ready to fire (*enabled*) and share on input place), the alternatives depend on a specific condition (predicate), this condition is written next to the input arc and the firing will depend on this predicate [92]. In the distinction of *operational* and *control states* (see 3.1.1) the Petri Net models the control flow and a certain marking represents the control state of the system. Operational states of the system (observable on channels or storages) are modeled though conditions or described in the actions of the transitions.

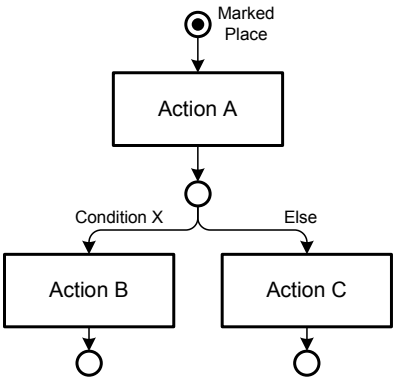


Figure 3.2: FMC Petri Net - Example

In the example FMC Petri Net, shown in Figure 3.2, *Action A* will be performed first. Then, depending if *Condition X* is true or not, either *Action B* or *Action C* will be performed.

Value Structures - Entity Relationship Diagram

After describing the compositional structures in Block Diagrams and the behavior and control flow in Petri Nets, the value structures in the storages and channels (operational state) are defined in FMC in Entity Relationship Diagrams [92], adapted from Chen [31]. In this also bipartite graph entities are represented as round nodes (with labels and attributes inside possible) and the relations as rectangles. At the links between the relation and the entity a number (cardinality) defines how many times an instance of an entity takes part in the relation. This is further supported by arrows in the relationship-rectangle (1 to 1: $\leftrightarrow$; 1 to n: $\rightarrow$; n to m *no arrow* [1]). The roles of an entity in a relation could also be noted at the links to the relation. Entity sets could be partitioned through embedding the partitioning entities in the partitioned entity or through a triangular shape connected to the different entities.

In the example in Figure 3.3 there are the entities *Person*, *Location* and *Country*. A person has the attributes *Name* and *Gender*, a location *Name* and *Area* and a country *Name*. The set of persons is partitioned into *Male* and *Female* and the set of locations is partitioned into *Cities*, *Towns* and *Villages*. The person *lives in* up to *n* different locations as a *Habitant* and a location has up to *m* habitants calling this location their *Place of Residence*. Furthermore, the location is *Part of 1* country which consists of *n* locations. A location could be (*0,1*) a *Capital of* a country and every country has *1 Capital*.

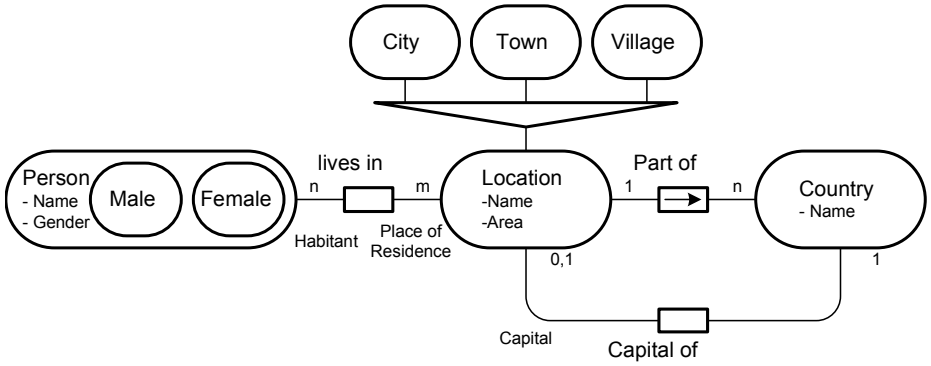


Figure 3.3: FMC Entity Relationship Diagram - Example

Operational State vs. Control State

In FMC [92, 124, 131] the distinction between operational and control states [63, 129] is important. The criterion for this distinction is based on semantics (the purpose of the variables) [92]:

Distinction Criteria - Operational and Control States [92, 131]: Whether a state variable has to be classified as an operational variable or a control variable, depends on how the domain of that variable is defined.

This is then further specified by:

Criteria - Operational State [92, 131]: The domain of an operational variable can be specified without itemizing each transition between the possible values, explicitly.

and:

Criteria - Control State [92, 131]: The domain of a control variable can only be specified by explicitly itemizing each value and each permissible transition between those values. The appropriate means of representation is a graph.

[92] and [124] furthermore compare operational and control variables as shown in table 3.1.

	Operational Variables	Control Variables
Criteria of distinction	Explicit itemization of each value and each transition between possible values unnecessary	Every value and every transition needs to be itemized explicitly
Convenient Representation	Algebraic Notation, E/R-Diagram	Graphs like Petri Nets
Power of the domain	Virtually unlimited	Limited by the human need to understand the control flow
Extensibility of the domain	Easy: for instance, extending the domain of some counter variable	Difficult: extensions are modifications of the control flow
Naming the variable's domain	Meaningful labels always possible	Labeling has no meaning: only single states have dedicated meaning

Table 3.1: Operational vs. Control Variables [92, 124]

3.1.2 Fundamental Modeling Concepts extended for Communication Systems (FMC-eCS)

In FMC-eCS, the Fundamental Modeling Concepts extended for Communication Systems [135, 145], the hierarchical modeling with a special regard on integrity, consistency and critical actions is in the main concern. It is based on FMC and can be seen as a preliminary step to FMC-QE, while the hierarchical modeling and the treatment of service requests is already in the viewpoint. In FMC-eCS there is a great importance attached to terms and definitions and therefore the main definitions are given in this subsection. As in FMC (described in section 3.1.1), and later in FMC-QE, the distinction of operational and control states is also regarded in FMC-eCS. Therefore, the extended definitions on operational and control states in FMC-eCS are also provided. As the modeling of service request handling and hierarchical modeling in FMC-eCS is the main bridge to FMC-QE, this will also be provided in this section.

Beside the references from Werner Zorn [135, 136, 145], this subsection is also based on a draft version of the PhD-thesis of Reinhard Höllerer [74].

Terms and Definitions

Following the fundamental ideas of mutual exclusion, loosely connected processes and semaphores of Dijkstra [45], in FMC-eCS the integrity of content and the handling and modeling of critical actions and critical locations are important, therefore these terms are defined as followed [136, 145]:

Critical Action / Critical Section Sequence of actions on critical contents on a critical action-field where at least one inconsistent system state could be reached.

A critical section is illustrated in figure 3.4. In an implementation, where the critical section is guarded by a semaphore, the transition at the beginning of the critical section could be associated to the *P-Operation* [45] and the transition at the end could be associated to the *V-Operation* [45]. Another example for a critical action could be a transaction in a database system which has to be executed completely (or not at all), otherwise the data in the system could be inconsistent. In this case the database is consistent before and after the transaction and in between it could be in-consistent, which is guarded by the transaction operations.

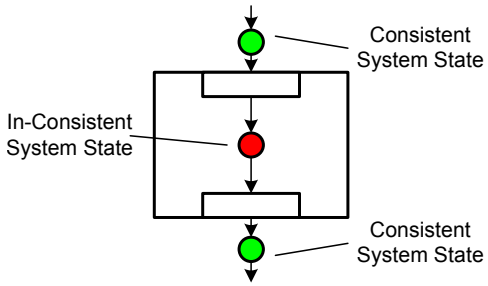


Figure 3.4: Critical Section [145]

Critical Location / Critical Actionfield Location (channel or storage), whose content is attackable by a third party (agent with no agreement concerning any interaction)

Figure 3.5 illustrates a critical actionfield. An example for such a critical location or actionfield could be a data transmission channel, where the data is transferred from *Agent A* to *Agent B*. In between the agents, the content (data) could get corrupted by noise on the channel, which is the *Third Party* in this case.

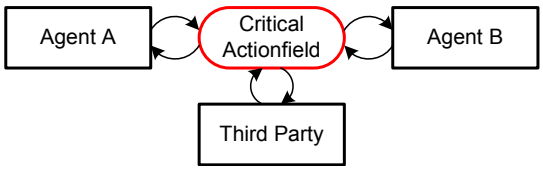


Figure 3.5: Critical Actionfield [145]

Critical Content / Critical Values Content, whose integrity is mandatory.

Integrity Absence of damages to crucial features of contents including information about such features.

In this notion there could be [136]:

1. Critical values on critical actionfields,
2. Critical values on un-critical actionfields,
3. Un-critical values on critical actionfields,
4. Un-critical values on Un-critical actionfields,

whereas only in the first case (critical values on critical actionfields) critical actions are necessary.

FMC-eCS furthermore distinguishes between consistent and in-consistent system states as followed [136]:

Consistent System State A system state is consistent if inside the modeled system a specific subject will only allow assertions which do not lead to any objections.

In an FMC-eCS Petri Net, as shown in figure 3.4, this state is illustrated by a token on a green place.

In-Consistent System State A system state is in-consistent if inside the modeled system the same subject will allow assertions which, when viewed from different perspectives, agent may object to. The objections may be lodged against existing as well as future system states if their occurrence is dependent on the interaction of external agents.

In FMC-eCS this is illustrated by a token on a red place, like shown in figure 3.4.

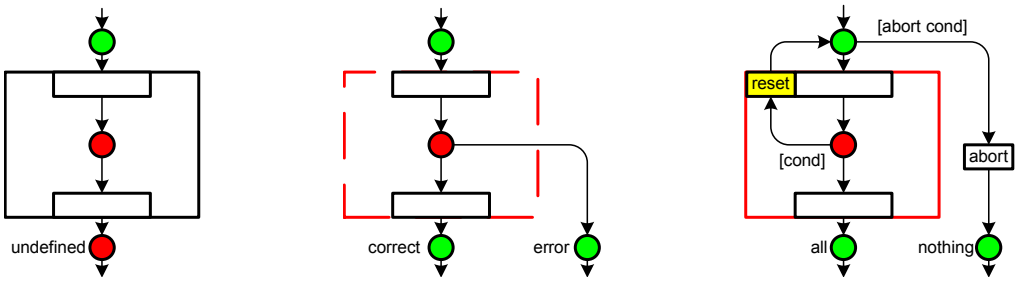
Types of Critical Actions

In FMC-eCS there is a distinction between unsecured, partially secured and secured critical actions as [136]:

Unsecured Critical Action Critical actions whose all or nothing execution is not guaranteed.

Partially Secured Critical Action Critical action whose future system state is known as correct or incorrect (without any correction)

Secured Critical Action Critical action which shall execute completely or not at all. In secured critical actions all third party actions have to be guarded against.



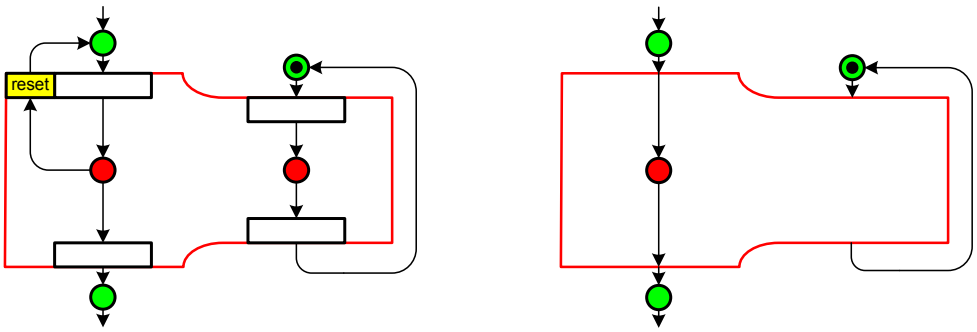
(a) Unsecured (b) Partially Secured (c) Secured

Figure 3.6: Types of Critical Sections [145]

An unsecured critical action is illustrated in figure 3.6(a). An example could be a receiver of data, who just receives the data, but does no know (*undefined*) if the data is corrupted or not. In a partially secured critical action, like shown in figure 3.6(b), the receiver would then know (for example through a parity bit) if the data is *correct* or not (*error)*. In a secured critical action, like illustrated in figure 3.6(c), in the example, the data would be received correctly (*all*) or the data would be corrupted (known through the parity bit) and then deleted (*nothing*) or re-sent by the sender.

Joint Actions

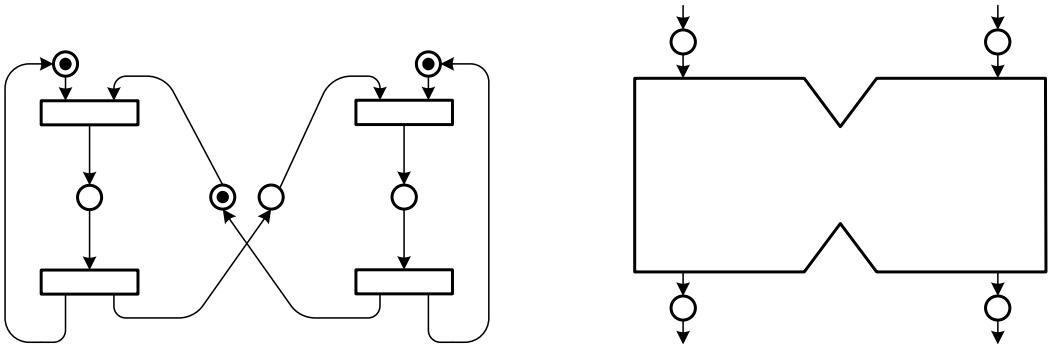
Critical actions are often defined among cooperating agents, where the consideration of both agents is important to know the overall states of the system. One example is the modeling of a client server scenario, where one critical action is dependent from another critical action, like shown in figure 3.7(a), with a corresponding abstract short notation in figure 3.7(b).



(a) Client/Server (b) Short Notation

Figure 3.7: Joint Action - Client/Server [135]

Another example of joint actions is a producer/consumer, illustrated in figure 3.8, where again figure 3.8(a) shows the Petri Net (simplified) and figure 3.8(b) shows the corresponding short notation.



(a) Producer/Consumer (b) Short Notation

Figure 3.8: Joint Action - Producer/Consumer [135]

In addition to the client/server and producer/consumer example, several other scenarios are possible. A summary of the described examples and further scenarios are illustrated in figure 3.9.

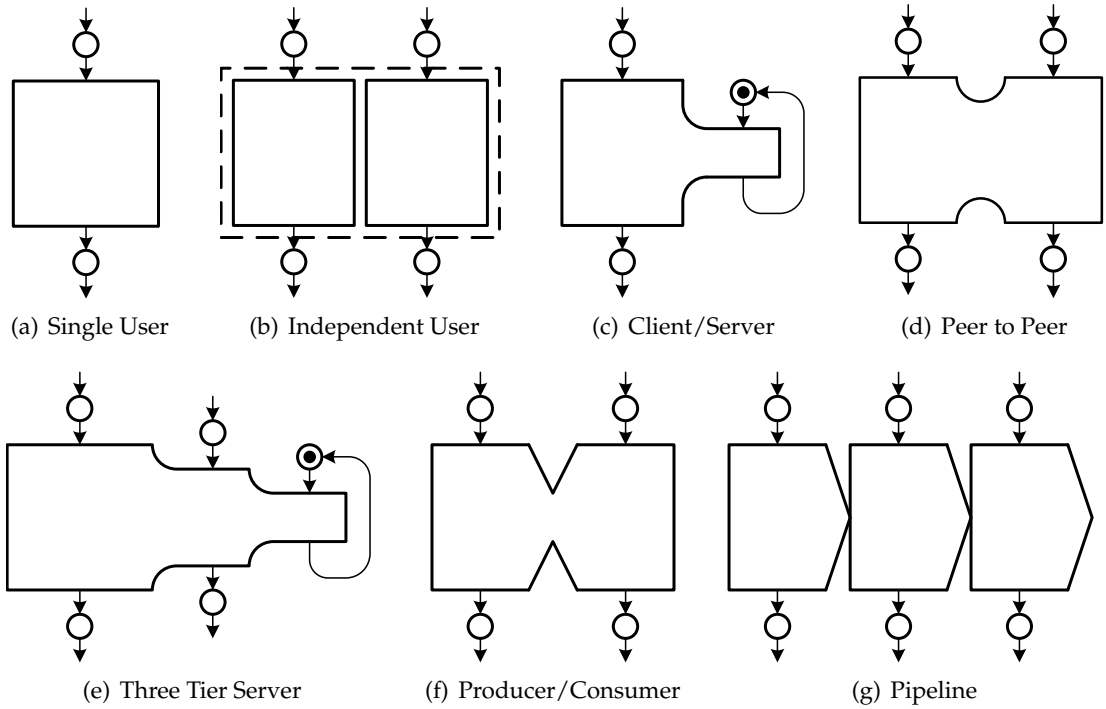


Figure 3.9: Joint Action - Short Notations [135]

Operational and Control States in FMC-eCS

Like in FMC, as described in section 3.1.1 in FMC-eCS, there is a distinction between operational and control states. In combination with the distinction into consistent and inconsistent states this leads to the following definitions [74]:

Consistent Control State A consistent system state is a state at the beginning or the end of a critical section, where the corresponding atomic operation has not started yet or is already finished.

Inconsistent Control State An inconsistent control state is a state inside the critical section in which it is unclear if the critical section could be successfully passed or the system has to be reseted to the consistent start state. The actual behavior could then be dependent from agents which are not in a protocol-relation to the actual processing agents (third parties).

Consistent Operational State A system is in a consistent operational state if there are no conflicting statements on the system state possible.

Inconsistent Operational State A system is in an inconsistent operational state if conflicting statements on the same system state at the same time are possible.

Furthermore, the resetting of the control automation of the system implies the resetting of the operational automation of the system.

Hierarchical Service Handling of Critical Actions

In FMC-eCS the hierarchical modeling is of main concern as in FMC-QE. A service request on one hierarchical level could create another service request on another hierarchical layer, whereas the service request on one layer is then transformed into service requests in the other hierarchical layer. If the service request at the lower layer is fulfilled, the service response is submitted back to the higher layer where this response is then handled. Through this view, also the error handling could be modeled on every fine grained hierarchical level to visualize it in the model.

If this view on the system is combined with the different types of critical actions and the distinction in operational and control states, the following three service handling classes are defined [74, 146]:

Unreliable Service In the unreliable service, as in figure 3.10 it is unclear, if the result of the critical action on layer $n+1$ is correct, therefore the result has to be checked again at layer n in order to prove if there were no errors and to reset in case of errors.

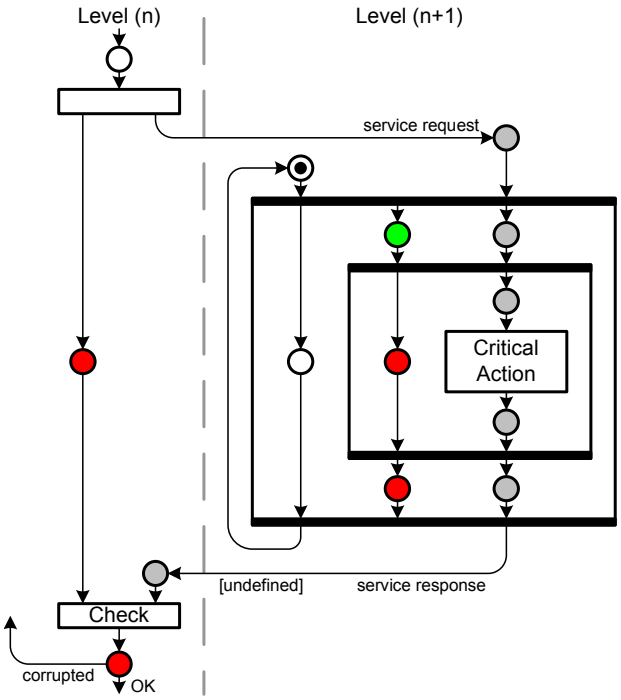


Figure 3.10: Unreliable Service [146]

Checked Service In a checked service, as in figure 3.11, the lower layer $n+1$ can notify the layer n if the critical section was passed successfully and the result is correct or if there was a problem concerning the service execution and the result is corrupted. Here the layer $n+1$ has no error handling, but through the error detection at this layer there is no need for an error detection at layer n, only an error handling is needed.

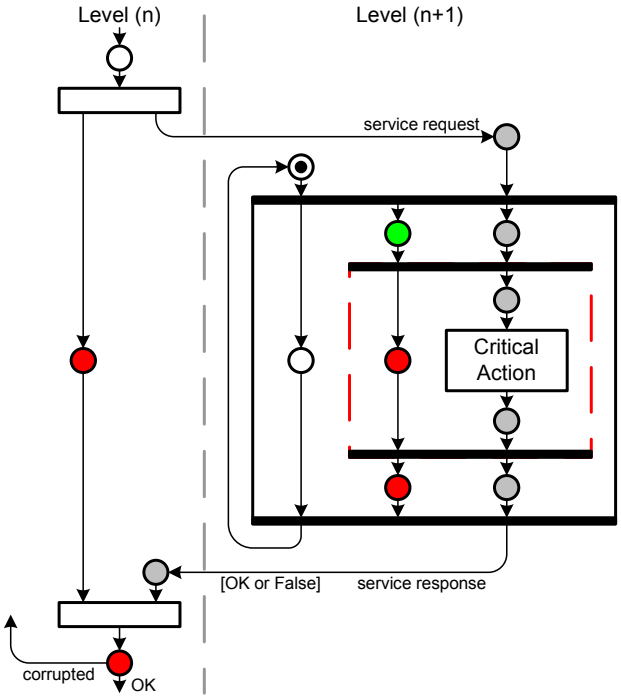


Figure 3.11: Checked Service [146]

Transactional Service In the transactional service in figure 3.12 the critical section in layer $n+1$ is passed all or nothing. On layer n the system state is consistent. Only in the case of nothing the state is still consistent, but the critical section possibly has to be passed again.

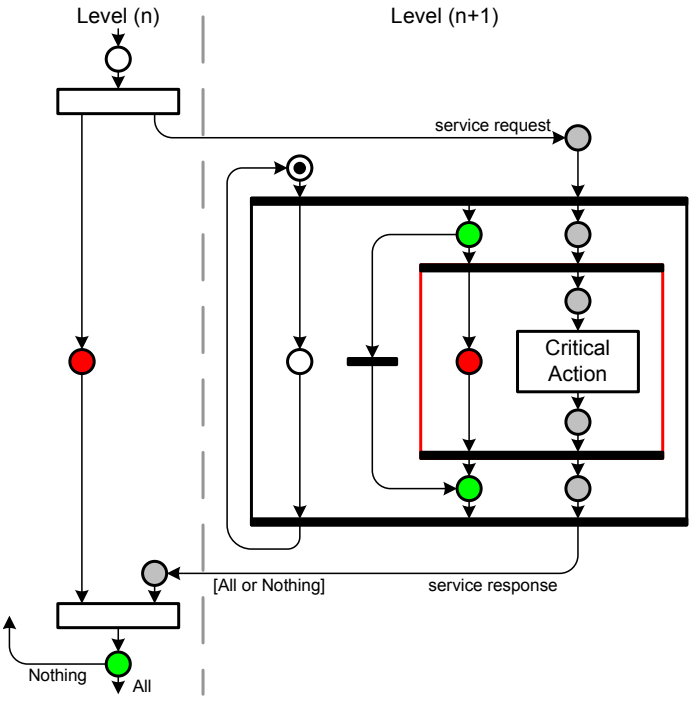


Figure 3.12: Transactional Service [146]

At the time the FMC-eCS models in figures 3.10 - 3.12 have been developed the development of FMC-QE was already in progress. Therefore, these two developments influenced each other and there are a lot of similarities from these models to the ones later explained in section 3.3.3, like the distinction of operational (gray) and control tokens (red - inconsistent state, green - consistent state and white - control tokens like idle or busy) as well as the intermediate transitions at the borders of the transitions, here amongst others interpreted as the semaphore operations $p(s)$ and $v(s)$, in FMC-QE interpreted as admission control and departure control.

3.2 Basic Definitions

This section provides basic definitions in FMC-QE. This includes the description of service requests, the hierarchical modeling and main quantitative measures.

3.2.1 Service Request

The modeling of service requests is a central concern in FMC-QE. While in Queueing Theory the central interest is the static structure of the server system and the dynamic behavior is implicit in the static structure and in Timed Petri Nets the interest lies in the dynamic behavior and the static structure beneath is implicitly modeled, in FMC-QE the system is modeled from the three dimensional view of the service request. Coming from the definition of the service request structures, both the static and the dynamic behavior are shown. Because of the special interest in the service requests further definitions are given in this section.

Some of the following definitions do not have to be modeled for every system, but once all of the FMC-QE performance analysts have to think about this topic in order to model their systems in a right way.

In physics and engineering sciences nearly all measurements or variables are given as a tuple of value (here notated in {}) and units ([]). In the Queueing Theory these units for service requests are almost ignored. So for a service rate the unit is one per second, but then it is hard to distinguish between different service request types and it is not possible to define hierarchical service requests. In FMC-QE service request units are defined. A service request SRq_i is the Service Request of type i. To start with a simple example, the request SRq_i could be to fill a car's tank. Then the unit of this request is the filling of the tank of car XY. Because of the different sizes of the tanks of the different cars, there has to be a unified service request N_i^e with value 1 per definition. In this example it would be filled one liter gasoline in a tank of a car (in this case there is no distinction between different fuel-types). So with a capacity of 50 liters the request would be defined as: $1[SRq_i] = 50[N_i^e]$. So the normalized service request N_i is a multiple of the unified service request N_i^e. By that definitions it is possible to map different service requests (cars with different tank sizes) to one service station.

In the modeling and evaluation of systems the first transformation from service requests to normalized service requests is often done intuitively and is therefore abstracted. When speaking about the evaluation of systems, the normalized service request N_i is often intuitively taken and when the distinction between SRq_i and N_i is not of special interest, the service request is taken as synonym to the normalized service request.

In the evaluation of FMC-QE models it is important to distinguish between service requests or jobs in the server which are in service ($N_{i,s}$) and the ones in the queue ($N_{i,q}$). So the service requests in the service station (queue + server) are:

$$N_i = N_{i,q} + N_{i,s}. \tag{3.1}$$

In the evaluation it is also useful to define $n_i = \{N_i\}$ as the number of service requests in a station and analogous $n_i = n_{i,q} + n_{i,s}$ for the number of requests in the queue and in service.

If a service request SRq_i is fulfilled, the corresponding service response is denoted as SRs_i. An unnormalized service response is further defined as:

$$SRs_i = \{SRs_i\}\,[SRs_i]. \tag{3.2}$$

In order to define a mapping between different service responses and servers, the normalized service request is denoted as:

$$N_i^r = \{N_i^r\}\,[N_i^r] \tag{3.3}$$

with the unified service response $N_i^{e,r}$.

3.2.2 Hierarchical Service Requests

The hierarchies of the service request structures are in the main focus in FMC-QE, because the service request in the origin of every service provisioning process [143] and the hierarchical decomposition is the key to cope with complex systems, whereas hierarchies are defined as:

Hierarchy [143]: A hierarchy implies a service request decomposition on one layer into different service requests on a lower layer.

The systems are modeled from three different hierarchical views: the service request structures, the (logical) server structures and the dynamic control flow behavior. Furthermore, the service requests are strictly modeled as a tuple of $\{Value\}$ and $[Unit]$:

$$SRq_i = \{SRq_i\}\,[SRq_i] \tag{3.4}$$

in order to make the hierarchical decomposition possible through the service request transformation on the hierarchical borders of the model. On the basis of canonical conventions SRq_i is transformed to N_i and therefore:

$$N_i = \{N_i\}\,[N_i] \qquad SRq_i = \{N_i^e\}\,[N_i^e]\,. \tag{3.5}$$

The fundamental laws of the quantitative evaluation used in FMC-QE are Little's Law [98, 143]:

$$N_i^{[bb]} = \lambda_i^{[bb]} * R_i^{[bb]} \tag{3.6}$$

and the Forced Traffic Flow Law [43, 68, 79, 95, 143]:

$$\lambda_i^{[bb]} = v_i^{[bb]} * \lambda_{parent(i)}^{[bb-1]} \tag{3.7}$$

In the scope of hierarchies [143] Little's Law defines relations inside an hierarchical level $[bb]$[1] (horizontal). According to Little's law, the number of service requests i on the hierarchical level

[1]The notation $[bb]$ of the hierarchical level in FMC-QE follows the notation of the lexicographical level (LL) in [25, 26]

$[bb]$ is a product of the arrival rate $\lambda_i^{[bb]}$ of service requests N_i per time unit and the response time $R_i^{[bb]}$.

The Forced Traffic Flow Law defines inter-hierarchical relations and defines therefore vertical relations between the hierarchical levels. It is the key to the hierarchical modeling in FMC-QE and was therefore extended [143] with hierarchical levels (definition in [143]: $\lambda_i^{[bb]} = v_i^{[bb]} * \lambda_{parent(i)}^{[bb-1]}$, classical definition [43, 68, 79, 95]: $\lambda_i = v_i * \lambda$). It defines that an arrival rate $\lambda_{parent(i)}^{[bb-1]}$ of service requests $N_{parent(i)}$ per time unit on hierarchical level $[bb-1]$ will be transformed into an arrival rate $\lambda_i^{[bb]}$ of service requests N_i on the hierarchical level $[bb]$. This transformation is done with the help of the traffic flow coefficient $v_i^{[bb]}$. As already mentioned, in FMC-QE the service requests are modeled as a tuple of $\{Value\}$ und $[Unit]$. Therefore, the Traffic Flow Coefficient is not only a scalar of $\{Value\}$ but also the basis for the service request transformation from the unit service request $N_{parent(i)}^{e[bb-1]}$ to a number of unit service requests $N_i^{e[bb]}$:

$$v_i^{[bb]} = \left\{ v_i^{[bb]} \right\} \left[v_i^{[bb]} \right] = \left\{ v_i^{[bb]} \right\} \left[\frac{N_i^{e[bb]}}{N_{parent(i)}^{e[bb-1]}} \right] = \left\{ v_i^{[bb]} \right\} \frac{\left[N_i^{e[bb]} \right]}{\left[N_{parent(i)}^{e[bb-1]} \right]} \tag{3.8}$$

Here the service requests $N_{parent(i)}^{e[bb-1]}$ on the hierarchical level $[bb-1]$ are the hierarchical parents ($parent(i)$) of the service request $N_i^{e[bb]}$ on the hierarchical level $[bb]$.

A simple example for a hierarchical service request could be a grocery list with the items one liter milk, ten eggs and one loaf of bread. The hierarchical parent is then the service request *Buy everything from the grocery list*. This is then decomposed into one request *Buy one liter milk*, ten requests *Buy one egg* and one request *Buy one loaf of bread* with the corresponding transformations from the overall request to the sub-requests.

In comparison to [43], the traffic flow coefficient enables now a real hierarchical decomposition of the service requests into sub-requests and is not only a visit ratio of the mean number of requests per job for a device in a somehow black box manner. In [43] it was asked: *Jobs generate an average of 5 disk requests and disk throughput is measured as 10 requests/second. What is the system throughput?* With the help of FMC-QE the system can be analyzed on multiple hierarchical levels and the question would not only be answered by the ratio, but also by questions like: *And from what sub-request(s) were these disk requests required?*

3.2.3 Quantitative Measures in FMC-QE

In this subsection the quantitative measures used in FMC-QE are described. The definitions are near to the definitions in the Queueing Theory, given in section 2.1, but are redefined here in order to refine them to the hierarchies and service requests in FMC-QE.

Arrival Rate λ

$\lambda_i^{[bb]}$ denotes the mean arrival rate of (normalized) service requests arriving at server i. It is defined as $\lambda_i = N_i / \Delta t$, where Δt is the mean inter arrival time. Furthermore, in FMC-QE the overall top level arrival rate $\lambda^{[1]}$ is calculated as the bottleneck arrival rate $\lambda_{bott}^{[1]}$ multiplied by a correction factor f (desired bottleneck utilization).

Traffic Flow Coefficient v

As discussed in the last subsection, the Forced Traffic Flow Law is a central law to define hierarchies in FMC-QE. Therefore the traffic flow coefficients are also defined in the model. They are defined as $v_i^{[bb]}$ as the absolute traffic flow coefficient of service request i on hierarchy level $[bb]$, $v_{i,int}^{[bb]}$ as the relative traffic flow coefficient of service request i on hierarchy level $[bb]$ to the next hierarchical level $[bb+1]$ and $v_{parent(i)}^{[bb-1]}$ as the absolute traffic flow coefficient on the next hierarchical level ($[bb-1]$) relative to service request i.

Multiplicity m

With the parameter multiplicity m the number of parallel servers in a server station is defined. It is possible that $m = \infty$. In this case every service request has a dedicated server. An example could be the user in a client server scenario, where every user-input request has a dedicated user.

There are several multiplicities defined in an FMC-QE model. In order to support parallelism on logical service request level, the parameters $m_i^{[bb]}$, $m_{parent(i)}^{[bb-1]}$ and $m_{i,int}^{[bb]}$ are defined. $m_i^{[bb]}$ is the absolute multiplicity of the server of service request i on hierarchy level $[bb]$. $m_{parent(i)}^{[bb-1]}$ is the absolute multiplicity of the server on the higher hierarchical level ($[bb-1]$). $m_{i,int}^{[bb]}$ is the relative multiplicity of the server of service request i on hierarchy level $[bb]$ to the next hierarchical level $[bb-1]$. So $m_i^{[bb]} = m_{parent(i)}^{[bb-1]} * m_{i,int}^{[bb]}$.

On the multiplexer server level the number of parallel servers is defined by m_j.

Multiplex Coefficient m_{mpx}

In a multiplex scenario, where one multiplexer server is shared among several logical servers, the multiplex coefficient $m_{i,mpx}^{[bb]}$ defines the fraction the logical server receives form the multiplexer.

Service Time X

The service time $X_i^{[bb]}$ is the mean time needed by the logical server i to process a service request i. This service request i is located on hierarchical layer $[bb]$.

Service Duration Y

The service duration is the mean elapsed time for the handling of a service request. While the service time $X_i^{[bb]}$ is an input parameter of the model, the service duration, denoted as $Y_i^{[bb]}$, is a resulting value. If a basic server station is not handled in a multiplex, the service duration equals the service time, if the basic server station is multiplexed, the service duration is longer than the service time, because in this case the service time is the time the service request is actually handled and the service duration is the overall elapsed time from the beginning of the processing till the end (including the breaks for the other multiplexed servers). Furthermore, the service duration is also a value for hierarchical server stations, while the service time is only a parameter for the basic server stations.

Service Rate μ

The service rate $\mu_i^{[bb]}$ is a measure for the maximal number of service requests a server could process in a given time period. It will be represented by $\mu_i^{[bb]} = SRq_i^{[bb]}/\Delta t$. According to the discussion about service requests and normalized service requests in chapter 3.2.1, the normalized service rate is denoted by $\mu_i^e = N_i^{e,[bb]}/X_i^{e,[bb]}$. For the sake of simplicity it will henceforth be assumed normalization as default and the superscripted 'e' will be omitted, except when needed to remind of the units.

Utilization ρ

The utilization of a server is denoted by $\rho_i^{[bb]}$. The Utilization is defined as $\rho_i^{[bb]} = \frac{\lambda_i^{[bb]}}{\mu_i^{[bb]} * m_i^{[bb]}}$ respectively.

Departure Rate D

The departure rate is the mean rate fulfilled service requests (service responses $SRs_i^{[bb]}$ or $N_i^{r[bb]}$) leave the server. This is denoted by $D_i = \frac{N_i^{r[bb]}}{\Delta t}$ for mean departure rate of normalized service responses i leaving the server with Δt as mean inter departure time.

In steady state the departure rate equals the arrival rate with the note of the transformation of the service request to the according service response $D_i = \frac{\lambda_i [N_i^{e,r}]}{[N_i^e]}$.

Mean Number of Service Requests n

The mean number of service requests in a service station is denoted by $n_i^{[bb]}$ with $n_i^{[bb]} = \left\{ N_i^{[bb]} \right\}$. It is the sum of the service requests in service $n_{i,s}^{[bb]}$ and the service requests queued $n_{i,q}^{[bb]}$.

Waiting Time W

The waiting time $W_i^{[bb]}$ denotes the mean time a service request i is queued in a service station.

Response Time R

The response time $R_i^{[bb]}$ is defined as the time interval between the arrival and the departure of a specific service request at a service station. So the response time is the sum of the waiting time and the service duration: $R_i^{[bb]} = W_i^{[bb]} + Y_i^{[bb]}$.

3.3 Graphical Representation

FMC-QE follows the idea of FMC of modeling different aspects of the systems in different diagram types. Therefore, the performance analyst models the quantitative aspects of the the static architecture of the systems in Block Diagrams, the quantitative dynamic behavior in Petri Nets and the relation between the two plans and the service request structure in Entity Relationship Diagrams.

3.3.1 Service Request Structures

As already described in chapter 3.2.1, the service requests are the key to FMC-QE models. The definition of a service request tree in FMC-QE is modeled in Entity Relationship Diagrams. Referring to figure 3.13, a service request is an entity with a name and the attributes action and server. The name describes the Service Request in a semantic manner. Due to the discussion about service requests and normalized service requests in this diagram all service requests are considered to be normalized and from now on a service request is normalized by definition.

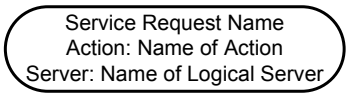


Figure 3.13: Service Request Entity

Beside the definition of the service requests the purpose of the Entity Relationship Diagram is the mapping between the later described dynamic structure in the Petri Net and the server structure in the Block Diagram. Because of this the action which links to the corresponding transition in the Petri Net and the corresponding logical server in the Block Diagram is stored. Of course, in the process of modeling the three diagrams are constructed in parallel, and so in a first iteration of the modeling of the service request sometimes these references are not given.

The strength of FMC-QE is the hierarchical modeling. A global or higher level service request can be decomposed into other service requests, down to the basic server requests (operational service request), and so a tree structure can be defined. A control service request is a hierarchical service request which controls the process of the child service requests. This child service request can be decomposed into other service requests, or it can be an operational service request, which can be processed by a server. By definition, control service requests are timeless and only the operational service requests consume service time. The meta structure of this tree is shown in figure 3.14. $[bb]$ identifies the hierarchical level in the service request tree. The root has level $[1]$ and children or the subtrees have level $[bb + 1]$ and so on. In another layout the hierarchical level $[bb]$ could also be noted at the side of the diagram with swim lanes separating the different layers, as illustrated in figure 3.17.

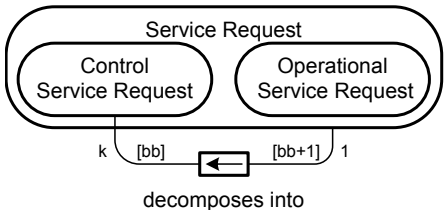


Figure 3.14: Entity Relationship Diagram Tree Metastructure

The traffic flow coefficient, defined in section 2.1.2, is a parameter of the service request and is represented as the cardinality or the composition relation in the Entity Relationship Diagram as shown in figure 3.15.

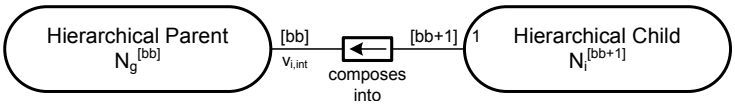


Figure 3.15: Traffic Flow Coefficient v

The service request tree is related to an external source and sink (*External World* - more discussions in section 3.4.2), which is by definition linked to the root of the tree. As illustrated in figure 3.16, both of the requests are on level $[1]$ and the traffic flow coefficient $v_{1,int} = v_1$ is 1 per definition.

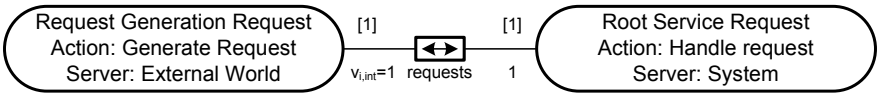


Figure 3.16: External Service Request Generator

For a better illustration of this diagram type a simple barbershop example is modeled in figure 3.17. In this barbershop the customer is served by washing the hair, cutting the hair in one of two haircuts (branch in Petri Net), dying the hair or making a perm and collecting the money.

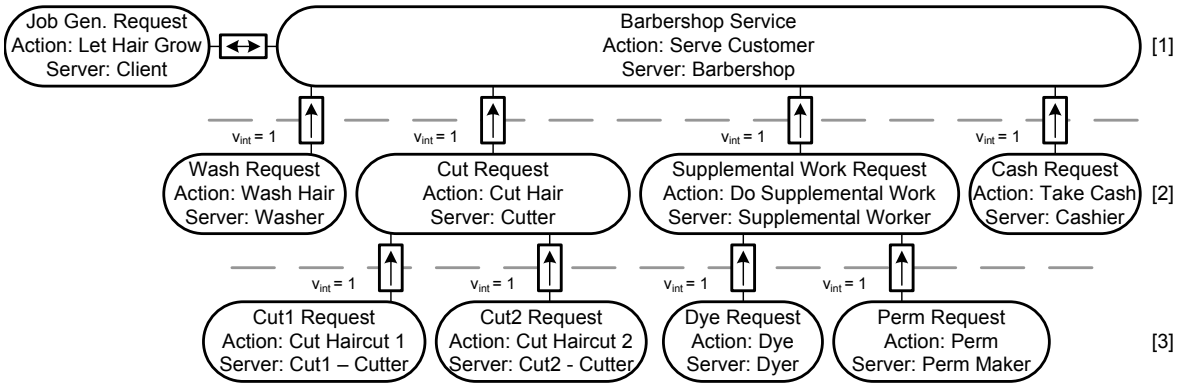


Figure 3.17: Barbershop - Service Request Structure

3.3.2 Static Structures

In FMC, as described in section 3.1.1, the static structure diagrams (Block Diagrams) represent the compositional structure of systems. This is done by using elementary components of type agent as an active component and storage as well as channels as passive components, together with arcs as connectors.

In the quantitative extensions of FMC-QE this diagram type is also used and extended. In FMC-QE it is another view on the system, and especially in the static structure diagrams it is the architectural or the specification view. In this diagram type the logical servers, following the service request structure and the multiplexer servers, following the static structure of the real systems, as well as queues and channels are considered. This diagram type is closely related to

the Queueing Theory, but in contrast to the Queueing Networks this diagram only shows the static structure. The dynamic behavior is not considered here, this is the purpose of the Petri Nets. In the Block Diagrams system specifications are given like service times of servers or buffer sizes of queues, which are partly independent from the actual service request scenario.

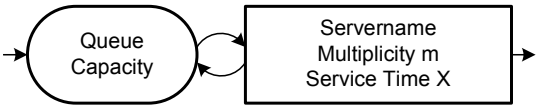


Figure 3.18: Basic Server Station

The most simple combination in a Block Diagram is a server together with a queue shown in figure 3.18. This combination is called basic server station (*BSSt*). A net as a combination of different basic server stations would be similar to a Queueing Theory Net. A parameter of the queue would be the capacity of the queue K. In most cases for the sake to simplicity the queue size is assumed to be infinite (∞). A Server would be parameterized with a name, its multiplicity and a service time or a service rate.

Often the communication in real systems is not implemented via shared memory or files (Queues). The communication is implemented via channels. For this fact the queueing station, shown in figure 3.19, is defined. The admission control puts the service request into the internal queue and after the server served the request, the departure control releases the result. This view could be mapped to a programmed environment, where a procedure call would be represented by the channel. The admission control is the handler which takes the call. The server is the procedure and the return value is handled by the departure control. The departure control is timeless by definition and is executed immediately.

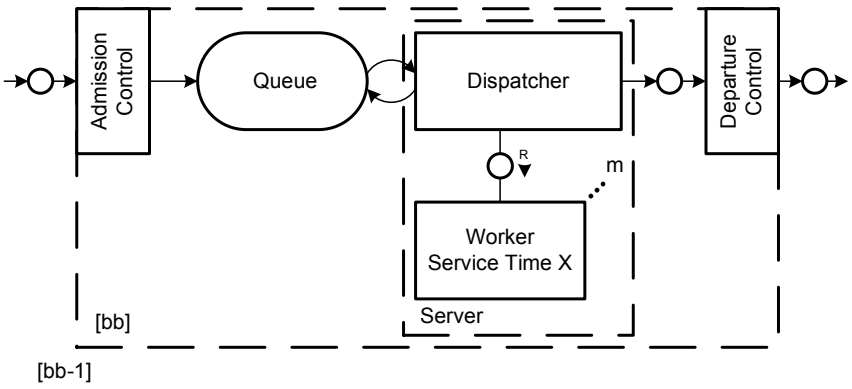


Figure 3.19: Queueing Station

The second task of admission control and the departure control is the hierarchical transformation of the service requests. The admission control transforms the service request of level $[bb-1]$ into the service request in the hierarchical level of the queueing station $[bb]$. The departure control is responsible for the re-transformation. In FMC-QE the dispatcher would be timeless and the workers would consume the service times.

For some cases it is helpful to define a server with infinite parallel servers, as shown in Figure 3.20. This server has no queue because every incoming request has a dedicated server. In FMC-QE the external source is defined as an infinite server.

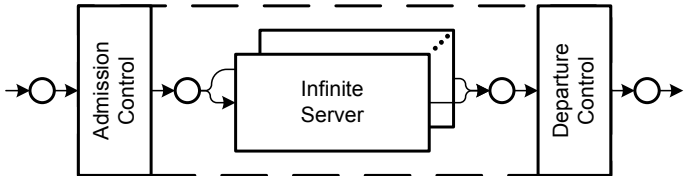


Figure 3.20: Infinite Server

The main strength of FMC-QE is the ability to define hierarchies. Often the system description gets complex because hierarchies neglected and the system is modeled flat. Of course a diagram like figure 3.21 looks very complex at the first glance, but knowing and modeling the system in a fine grained manner, can help for a better understanding of the system and errors and inconsistencies in the modeling are found almost immediately.

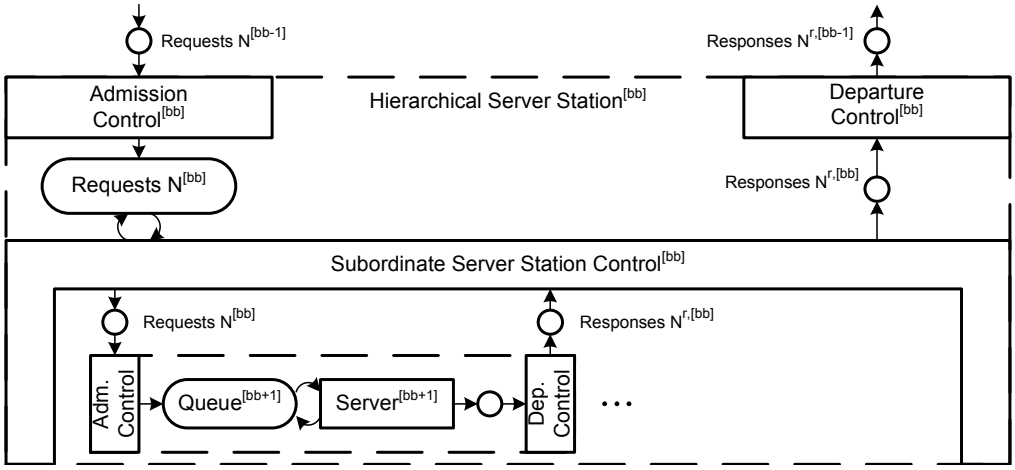


Figure 3.21: Hierarchical Server Station

These general considerations about the inner structure of the servers were in the minds of the FMC-QE developers when implementing the methodology. Of course for larger systems this kind of diagrams, as shown in figure 3.21, would become to large. Therefore, a short notation was developed. In this short notation every logical server (Hierarchical Server Station or Basic Server Station) correspondent for the handling of a service request is represented by one rectangle annotated by a name and a multiplicity m. Also the hierarchical level $[bb]$ of the corresponding service request is denoted. An exemplary hierarchical server station with two basic server stations is illustrated in figure 3.22.

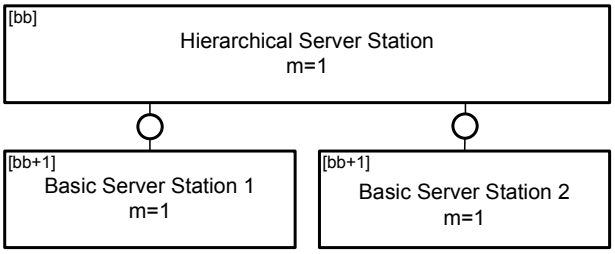


Figure 3.22: Hierarchical Server Station - Short Notation

While the logical servers follow the structure of the service request, the multiplexer servers are also modeled in the Block Diagram. As illustrated in figure 3.23, the multiplexer servers are also named and have a multiplicity attribute.

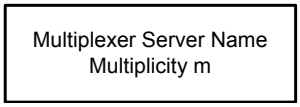


Figure 3.23: Multiplexer Server

The mapping between the logical and the multiplexer servers is modeled through a matrix. At the connection points of the matrix the measured service time X is notated. This is the time the multiplexer server would need to handle a service request of the logical basic server when no other service requests is handled by the multiplexer server. An exemplary mapping is illustrated in figure 3.24. It is also possible that a multiplexer server handles more than one logical server (multiplex).

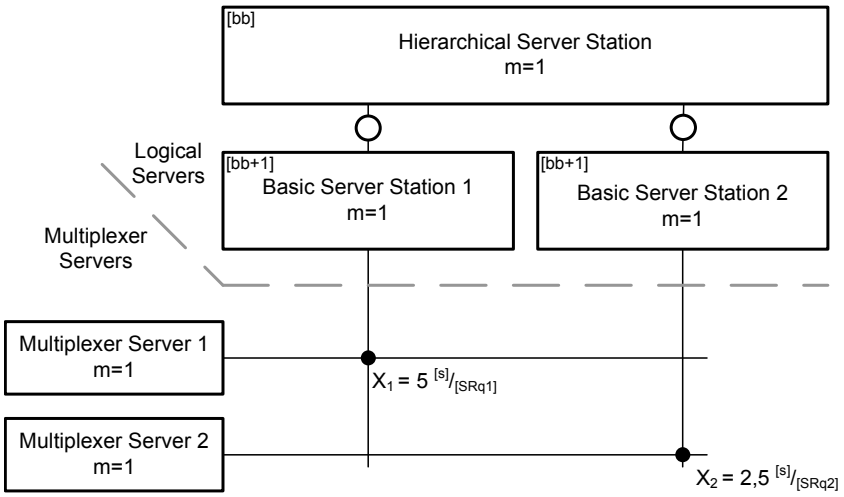


Figure 3.24: Mapping between Logical and Multiplexer Servers

Figure 3.25 illustrates the static structure of the barbershop example. The logical servers on top are handled by three multiplexer servers, the *Barber Boss*, the *Barber* and the *Apprentice*, whereas the boss cuts the haircut 1 and makes the perm, the barber cuts the haircut 2 and dyes the hair and the apprentice washes the hair and collects the money.

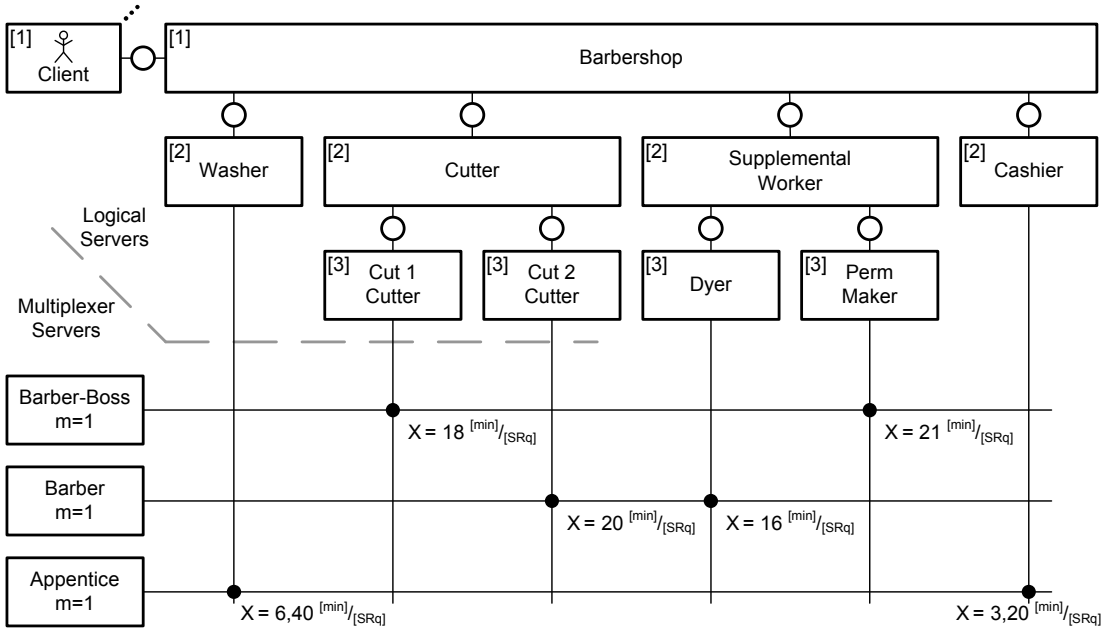


Figure 3.25: Barbershop - Static Structures

3.3.3 Dynamic Structures

The dynamic behavior in FMC-QE is represented in Petri Nets. Within a FMC-QE Petri Net the inner most structure is called controlled operational transition. This transition consists of an operational transition (*execute operation*), an input and an output place for the operational transition and a control loop, shown in figure 3.26

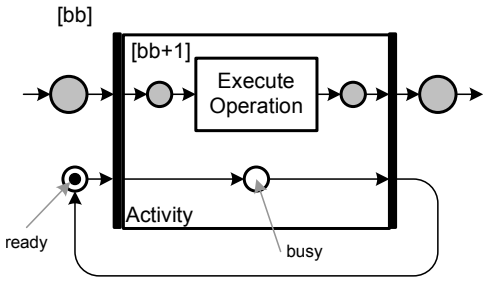


Figure 3.26: Controlled Operational Transition

The operational transition represents the processing of a basic service request in a server. The tokens in the input-place of this transition represent the service requests in process and the tokens in the output place represent the service responses. The places are presented in grey because the tokens are considered to be colored tokens like in Colored Petri Nets [80]. The loop at the bottom of the controlled operational transition represents the control flow of the server. If the control token (*ready*) is in front of the transition, the server is ready and active to process a request, and if the token is on the place inside the transition, the server is busy and not able to process another service request.

The controlled operational transition could be integrated into a representation of the activities a queueing station (admission control, queue service requests, process service requests and the

departure control) shown in figure 3.27. In this figure the controlled operational transition is abstracted to the transition called activity.

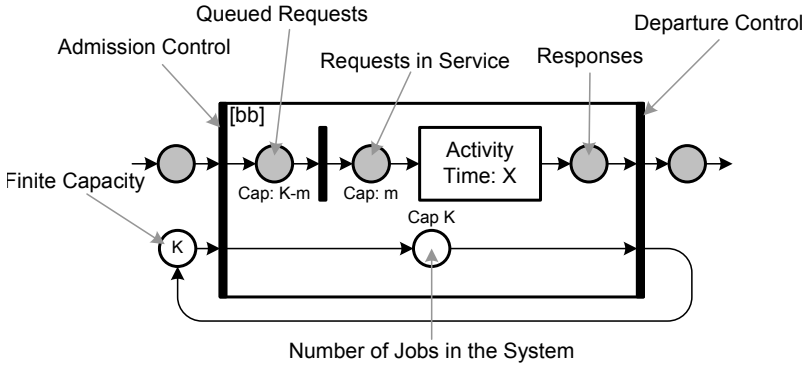


Figure 3.27: Dynamic Behavior of a Queueing Station

The admission control is represented in the black timeless transition at the left. At this time the admission control discards no service requests. In hierarchical systems the admission control and the departure control perform the service transformation.

Because of the firing rules of a Time Augmented Petri Net, there is a distinction between the service requests in service and the ones queued. The token stays before transition in the service time, and at the end the transition fires timelessly and the token 'jumps' from the input place to the output place. The service requests in service are represented in the tokens in the input place of the operational transition. Due to the multiplicity m defined in the Block Diagram of the corresponding server, the place would have the capacity m. The queued service requests are represented in the place top-left. This place has the capacity $K - m$ also imported from the Block Diagram. The transition between the queue place and the *in service* place is a timeless transition which represents the start of the service processing.

If the corresponding server has a multiplicity m of more than 1, the activity or the controlled operational transition could be activated more than once. In this case the inner structure of the activity could be modeled as shown in figure 3.28. A timeless dispatcher sends the request to the corresponding controlled operational transition. This figure is only shown to illustrate the FMC-QE view on the multiplicity and has not to be modeled for every scenario.

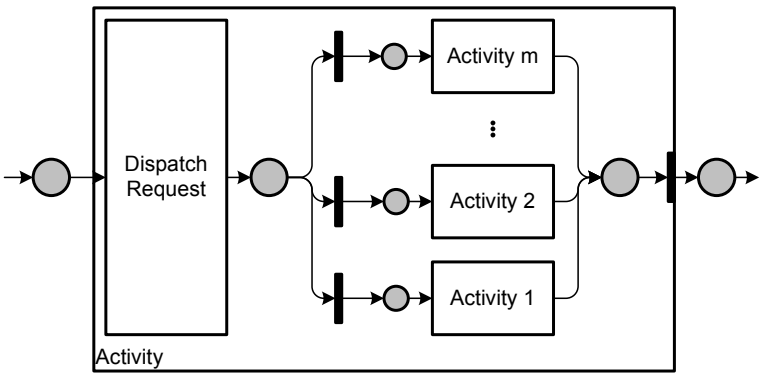


Figure 3.28: Parallel Server - Activity Refined

Coming back to figure 3.27, it is not necessary to define the capacities and queue sizes here again, first the diagrams could be inconsistent, and second the Petri Net gets more complex.

But anyway, the performance analyst is free to import this parameters here in order to have the information together in one diagram. But as mentioned, the multiplicity and the service times are defined in the static plans and are only imported and not defined here.

The black rectangle at the right in figure 3.27 represents the departure control. It fires per definition timelessly and immediately, so in the steady state there are zero tokens in the response place.

While tokens in the five gray places in figure 3.27 on the top represent the operational states (service requests and responses), the tokens in the two place at the bottom represent the control states to the system [124, 131, 137]. The tokens in the place in front of the start transition (bottom left) represent the capacity of the server. A token in this place represents that the server is active to get a request. This parameter is also not defined in this diagram, it is imported from the Block Diagram. The tokens in the place at the bottom middle represent the number of service requests in the server. The performance analyst could use Colored Petri Nets in order to define the operational states of the system.

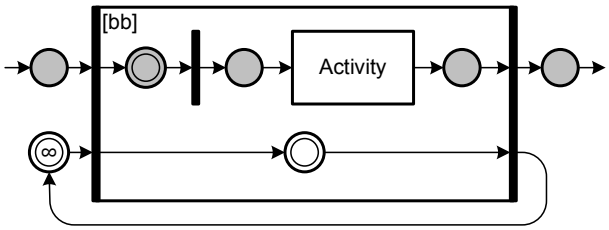


Figure 3.29: Infinite Queues

If the queue has an infinite size, the station could enqueue every service request arriving at the station, and therefore, in this case the activation place has an infinite size with an infinite amount of tokens in it, as depicted in figure 3.29. One could argue, that this infinite place with infinite tokens is irrelevant for the processing of the Petri Net, but nevertheless the place is drawn in order to illustrate the activation of the station.

For some cases, like modeling the external world (service request generator), there is the need of an infinite server and an activity, which could be activated infinite times. In this cases the structure defined in figure 3.30 could be used. The place in front of the activity has an infinite capacity and so the station is activated infinite times again.

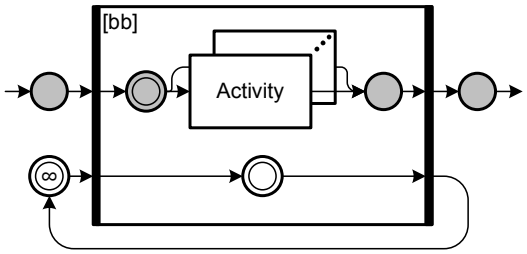


Figure 3.30: Infinite Server

A hierarchical transition is shown in figure 3.31. At the black border transitions at the left and the right a service transformation is processed. At the left border the service request $N^{[bb]}$ is transformed into v different sub requests $N^{[bb-1]}$. At the right border transition the sub service responses are reassembled.

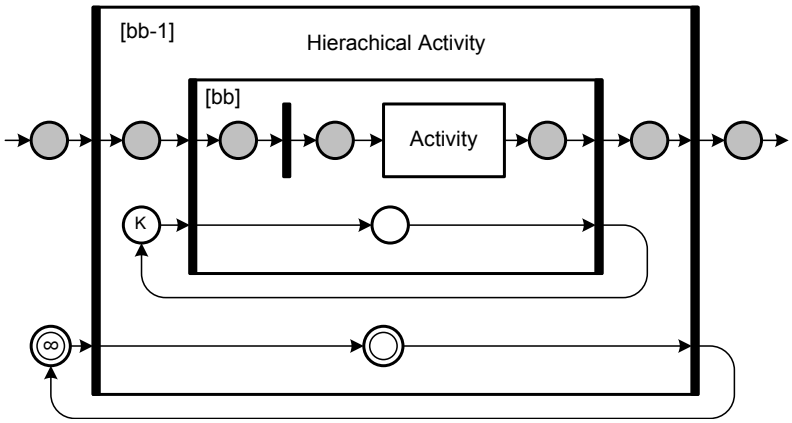


Figure 3.31: Hierarchical Transition

Again, the tokens in infinite capacity places at the bottom represent the number of service requests in the sub system. Only in exceptional cases, when a limited number of service requests are allowed to be in the sub system, this place would have a finite capacity.

In the branch, illustated in figure 3.32, every job in level $[bb-1]$ has to be delivered to one of the activities A or B. Inside the level $[bb-1]$ there is no job transformation. The transformation is at the incoming transitions of activity A and B.

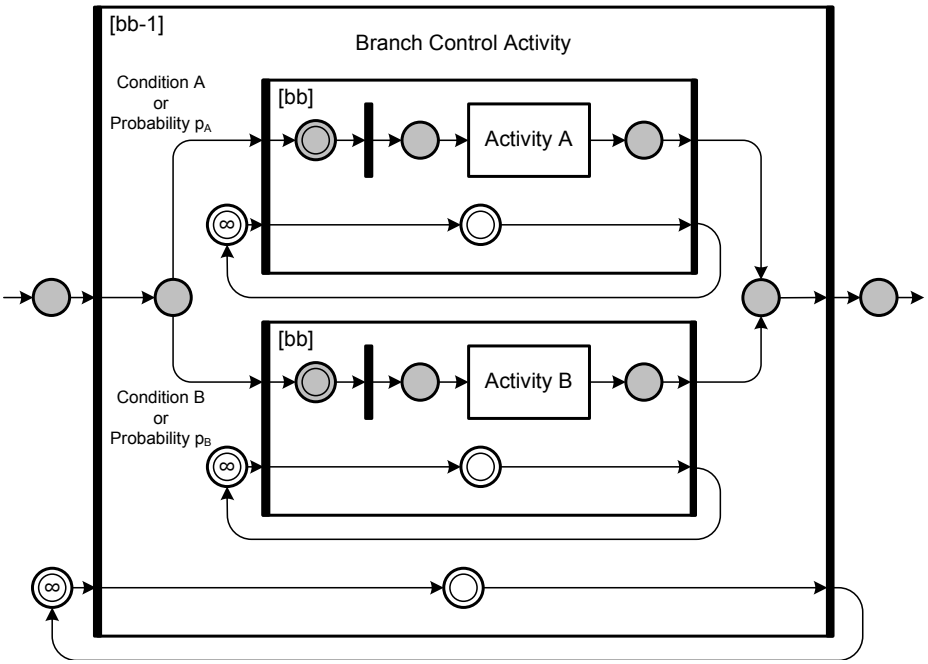


Figure 3.32: Branch

An important remark to the parallel service transition in figure 3.33 is that the traffic on all three outgoing arcs of the service request transformation transitions at the left are the same. The service request is delivered to both of the activities A and B. The transformation into the service requests of the activities A and B ($N^{[bb-1]}$ to $N^{[bb]}$) is not done at this time, this is done in the next step, the incoming transition of the activities (admission control).

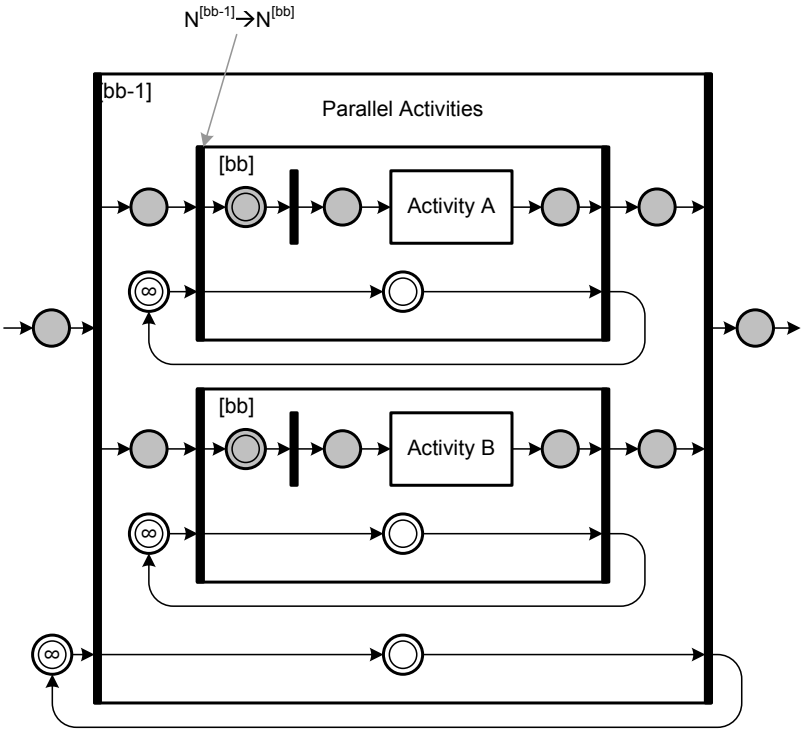


Figure 3.33: Parallel Activities

Beside the parallel activities in figure 3.33, a serial processing of activities, shown in figure 3.34, is also possible.

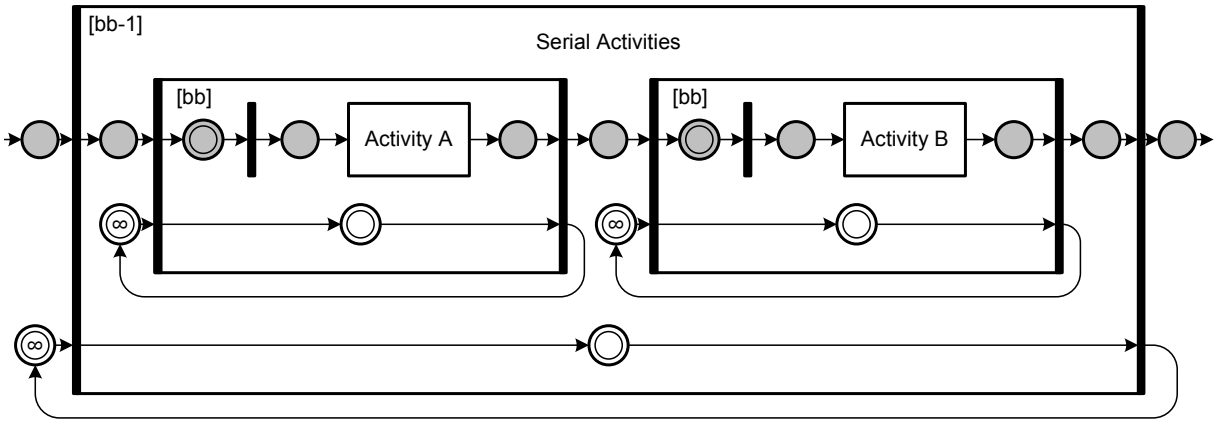


Figure 3.34: Serial Activities

One could argue that in a serial processing of service requests in a hierarchical manner, the service responses should be redirected to the hierarchical parent after each step and then the hierarchical parent delivers the next sub-request to the next serial handler. For the sake of simplicity this request-response handling is simplified in figure 3.34 to a sequence in which every serial sub-request handler delivers the service response to the next sub-handler. In this scenario the service requests from the hierarchical parent to the hierarchical sub-handlers could also be envisaged as a list of service requests in which every sub-handler marks its corresponding ser-

vice request as *done* and hands the list of service requests with this requests marked as done (*service response*) to the next sub-handler.

In FMC-QE there is a different distinction between open and closed nets. By definition the transition on level [1] is linked to the external load generator, interpreted as a service request source and a service response sink, which could be mapped to an open net, together with a time interval, which defines the time between a response and a request of a single customer. If this time interval would be zero, the solution could be compared to a closed net. There are some differences in the calculation of closed and open nets and this hybrid approach, discussed in section 3.4.2 and 4.1. In figure 3.35 the most simple FMC-QE Petri Net is shown. A service request generator and one activity are connected. In order to get to more complex solutions, the Controlled Operational Transition could be replaced by more complex nets.

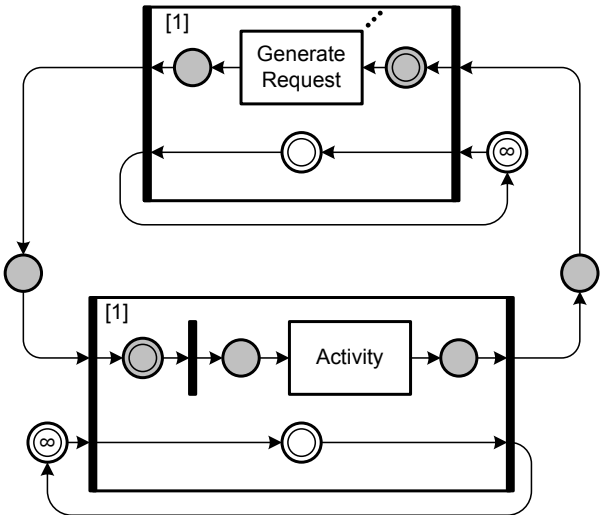


Figure 3.35: Most Simple FMC-QE Petri Net

If parallelism on logical server level should be explicitly modeled, like the modeling of threads, a sub-net could be replicated and the multiplicity could be illustrated by dots as shown in figure 3.36. An example of this is used in the case study in section 5.3.

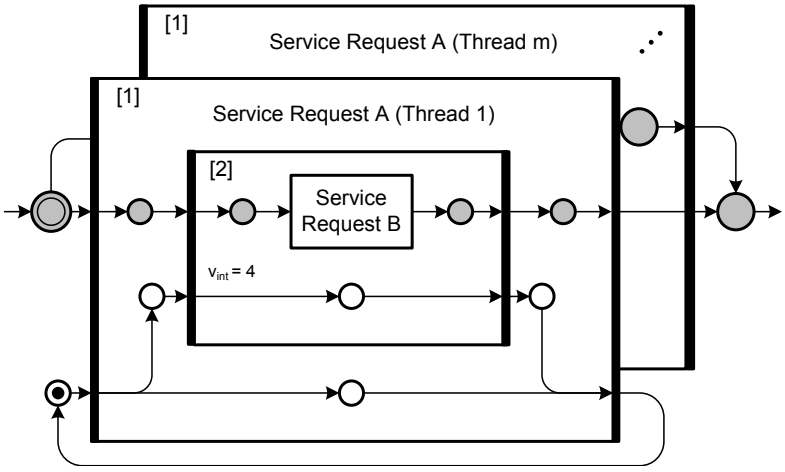


Figure 3.36: Parallelism on Logical Server Level - Threads

In figure 3.37 the barbershop is used again to illustrate an example of the diagram type. In this model first the hair is washed, then cut in haircut 1 or 2, afterwards in a supplemental step either dyed, permed or left as it is (NOP) and finally, the money is collected from the customer.

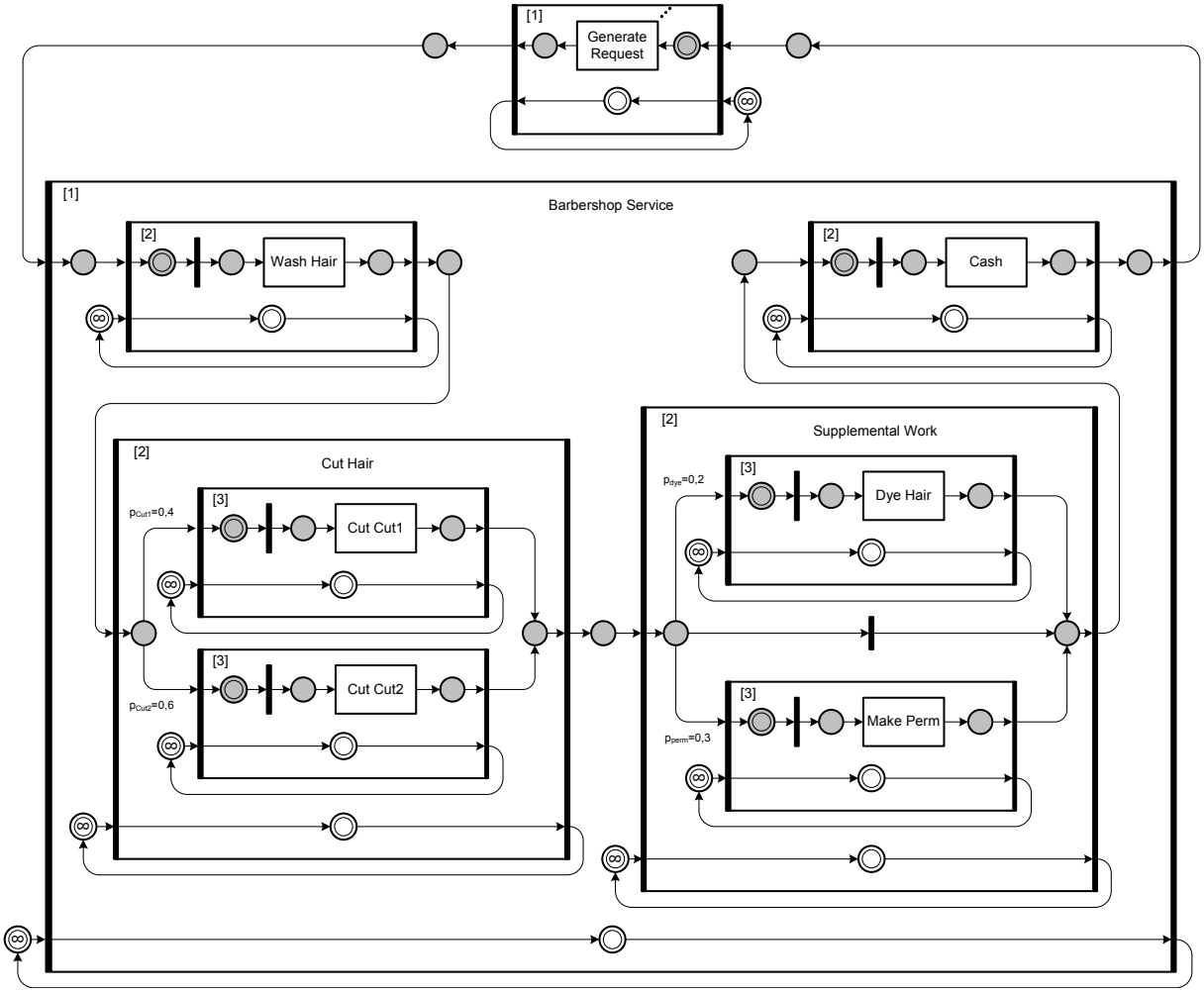


Figure 3.37: Barbershop - Dynamic Behavior

3.4 Calculus

After modeling the system in the three dimensions of FMC-QE, the *service request structures*, the *server structures* and the *dynamic behavior*, the predictions of the performance values are calculated with the help of the FMC-QE Calculus. In this section the parameters, values and formulas of the calculus are explained. Alongside this the FMC-QE Tableau, a representation of the Calculus, is shown for every section. This part is structured as follows:

In the first subsection 3.4.1 the main laws of the calculus, Little's Law [98] and the Forced Traffic Flow Law [79], are recapitulated.

In the experimental parameters section of the Calculus the experimental parameters are defined, as the name implies. These are the overall number of service requests $n_{ges}^{[1]}$, the derived possible bottleneck throughput $\lambda_{bott}^{[1]}$, the desired bottleneck utilization f and the overall arrival rate $\lambda^{[1]}$. Subsection 3.4.2 describes this *Experimental Parameters* section and the external load generation (load model) in FMC-QE.

In subsection 3.4.3 the *Service Request Section* is described. In this section the performance parameters of the service request, like hierarchy levels, probabilities, traffic flow coefficients and arrival rates are defined.

In the *Server Section*, described in subsection 3.4.4, the performance parameters of the logical servers including their mappings to the multiplexer servers are stored.

The main calculations of the performance values are processed in the *Dynamic Evaluation Section*. This section, divided into the calculations for operational (parallel and infinite) and control (hierarchical, serial, parallel, branch and loop) service requests, is explained in subsection 3.4.5.

The performance parameters of the multiplexer servers are defined in a separate section, this *Multiplexer Section* in explained in 3.4.6.

Subsection 3.4.7 discusses the complexity of the approach.

Table 3.2: Tableau Example (see Appendix - Table B.1)

Experimental Parameters	
$n_{ges}^{[1]}$	30
$\lambda_{bott}^{[1]}$	3,750
f	0,600
$\lambda^{[1]}$	2,250

Service Request Section								Server Section						Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
2	1	Cash	1,00	1,00	1,00	1,00	2,250	Cashier	1	1	1	1	0,056	1,000	18,000	0,125	0,018	0,008	0,125	0,056	0,143	0,063
3	2	Blow-Dry	0,50	1,00	1,00	0,50	1,125	Blow-Dryer	1	∞	∞	4	0,350	1,000	2,857		0,000	0,000	0,394	0,350	0,394	0,350
2	3	Dry	1,00	1,00	1,00	1,00	2,250	Dryer	1	1	1				∞		0,000	0,000	0,394	0,175	0,394	0,175
3	4	Perm	0,30	1,00	1,00	0,30	0,675	Permer	1	1	1	3	0,350	0,467	1,333	0,506	0,519	0,769	0,506	0,750	1,025	1,519
3	5	Dye	0,20	1,00	1,00	0,20	0,450	Dyer	1	2	2	2	0,333	0,125	0,375	0,600	0,675	1,500	1,200	2,667	1,875	4,167
2	6	Supp. Work	1,00	1,00	1,00	1,00	2,250	Supp.	1	1	1				3,750		1,194	0,531	1,706	0,758	2,900	1,289
3	7	Cut2	0,60	1,00	1,00	0,60	1,350	Cut2-Cutter	1	1	1	2	0,333	0,750	2,250	0,600	0,900	0,667	0,600	0,444	1,500	1,111
3	8	Cut1	0,40	1,00	1,00	0,40	0,900	Cut1-Cutter	1	1	1	3	0,300	0,533	1,778	0,506	0,519	0,577	0,506	0,563	1,025	1,139
2	9	Cut	1,00	1,00	1,00	1,00	2,250	Cutter	1	1	1				3,750		1,419	0,631	1,106	0,492	2,525	1,122
2	10	Wash	1,00	1,00	1,00	1,00	2,250	Washer	1	3	3	1	0,111	1,000	9,000	0,083	0,000	0,000	0,250	0,111	0,250	0,111
1	11	Barbershop Serv.	1,00	1,00	1,00	1,00	2,250	Server	1	1	1				3,750		2,631	1,169	3,581	1,592	6,212	2,761
1	12	Job Generation	1,00	1,00	1,00	1,00	2,250	Generator	1	1	1		10,572		0,095		0,000	0,000	23,788	10,572	23,788	10,572

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Appentice	5	0,167
2	Barber	1	0,267
3	Barber Boss	1	0,225
4	Customer	∞	

The FMC-QE Tableau is a hierarchical balance sheet used for the performance evaluation of FMC-QE models. The Tableau consists of three linked tables. The mathematical formulas of the Calculus are used to provide support for the hierarchical modeling. The exemplary Tableau,

shown in table 3.2, is the corresponding Tableau of the barbershop example, modeled in section 3.3. In order to integrate the handling of infinite servers, this example is extended by the blow-drying of the hair by the customer.

3.4.1 Fundamental Laws

In this subsection the two fundamental laws used in FMC-QE, Little's Law and the Forced Traffic Flow Law, are described on a high level as a recap of the more detailed discussion on hierarchies and the fundamental laws in section 3.2.2 as a preliminary for the FMC-QE Tableau. Also the hierarchical levels ($[bb]$) and indices (i) are integrated in the notation of the laws.

Little's Law [98] is one of the fundamental laws in FMC-QE. It is the basis of the performance evaluation within a hierarchical level (horizontal). The law defines the dependence between the mean number of jobs in a system $n_i^{[bb]}$, the mean arrival rate $\lambda_i^{[bb]}$ and the mean response time $R_i^{[bb]}$ [98, 125]:

$$n_i^{[bb]} = \lambda_i^{[bb]} * R_i^{[bb]} \tag{3.9}$$

The **Forced Traffic Flow Law** (FTFL) [43, 68, 79] is the second fundamental law in FMC-QE. It is the key to the hierarchical modeling (vertical). Through this law and the use of a traffic flow coefficient $v_i^{[bb]}$, the FMC-QE performance analyst is able to transform an arrival rate at a hierarchical border. The internal arrival rate is computed as a product of the external arrival rate and the traffic flow coefficient [43, 68, 79]:

$$\lambda_{i,int}^{[bb]} = v_i^{[bb]} * \lambda_{i,ext}^{[bb-1]} \tag{3.10}$$

Beside the transformation of the value the service request itself is transformed (unit-transformation). The external arrival rate $\lambda_{i,ext}^{[bb-1]}$ has the unit [external service requests per time unit] and the internal service request has the unit [internal service requests per time unit]. A detailed description of this service request transformation could be found in section 3.2.2.

3.4.2 Experimental Parameters

In this section the *Experimental Parameters* of the Calculus are described. Therefore, first the parameters and the relations between the parameters are described, after that the load model in FMC-QE is described. Finally an exemplary cutout of a tableau is shown.

The parameters of this part are:

- $n_{ges}^{[1]}$ – Overall number of top level ([1]) service requests in the whole model (in the system and external);
- $\lambda_{bott}^{[1]}$ – Maximum bottleneck arrival rate;
- f – Desired bottleneck utilization;
- $\lambda^{[1]}$ – Arrival rate of top level ([1]) service requests;

$n_{ges}^{[1]}$ defines the overall number of top level service requests ($SRq^{[1]}$) in the model. This includes the service requests in the system ($n_{sys}^{[1]}$) and the service requests in the external world (clients preparing the service request or following other activities) ($n_{ext}^{[1]}$).

The parameter $\lambda_{bott}^{[1]}$ defines the arrival rate, where the bottleneck of the system would have a utilization of 1. This parameter is derived as the minimum service rate of every logical server ($\mu_i^{[bb]}$) multiplied by the number of corresponding parallel logical servers ($m_i^{[bb]}$) normalized to top level service requests ($\frac{1}{v_i^{[bb]}}$):

$$\lambda_{bott}^{[1]} = \min\left(\frac{\mu_i^{[bb]} * m_i^{[bb]}}{v_i^{[bb]}}\right) \forall\, i \tag{3.11}$$

In order to avoid arrival rates greater than the maximum bottleneck arrival rate (which would be a conflict to the stability criteria - $\rho < 1$), the parameter f is established. This parameter defines the desired bottleneck utilization and allows a simple arrival rate adjustment.

The overall arrival rate of top level ([1]) service requests generated by the load generator (*Outside World*, *Client*, *Customer*) is defined as $\lambda^{[1]}$. $\lambda^{[1]}$ is derived as the product of f and $\lambda_{bott}^{[1]}$:

$$\lambda^{[1]} = f * \lambda_{bott}^{[1]} \tag{3.12}$$

In some cases, due to the modeled systems, $\lambda^{[1]}$ is directly defined. Then f and $\lambda_{bott}^{[1]}$ are not used.

External Load Generation / Load Model

In FMC-QE there is no differentiation between closed and open models like in traditional Queueing Theory or Time Augmented Petri Nets. Therefore, the most basic FMC-QE model is represented in figure 3.38.

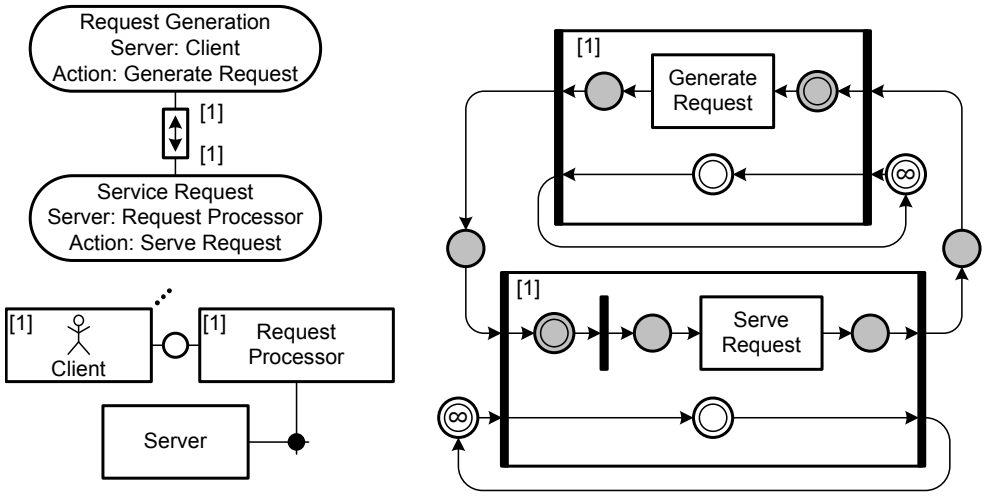


Figure 3.38: Basic FMC-QE Model

The external load generation is always represented by a number of clients or load generators in the Block Diagram which generate the level [1] service requests. In the Petri Net the behavior of the load generation clients are described as infinite servers because by definition every service request has a dedicated client, so the client would behave like an infinite server. In the Entity Relationship Diagram every top level service request has a 1to1 relation to a service request generation.

In comparison to closed systems, the external time is not defined as zero, because when modeling systems, it nearly always makes no sense to enqueue a service response immediately as a new service request. The model here is different. First of all, there is a differentiation between the service request which enters the system and the service response. The client does not only handle the request generation but also the response. The second difference is that in FMC-QE models, both, the arrival rate ($\lambda^{[1]}$) and the mean overall population ($n_{ges}^{[1]}$), could be defined. The relation between these parameters is defined through the General Response Time Law [27, 43]:

$$R_{sys}^{[1]} = \sum_{i=1}^{N} R_i^{[bb]} v_i^{[bb]} \tag{3.13}$$

and the Interactive Response Time Formula [43]):

$$R_{sys}^{[1]} = \frac{n_{ges}^{[1]}}{\lambda^{[1]}} - X_{ext}^{[1]} \tag{3.14}$$

modified to:

$$X_{ext}^{[1]} = (\frac{n_{ges}^{[1]}}{\lambda^{[1]}}) - R_{sys}^{[1]} \tag{3.15}$$

Steady State Systems - Notation In order to explain the different load scenarios, figure 3.39 shows the notation of a steady state system illustration (stationary process). The cycling service requests or service responses are represented as one moving dot on a circle. The time the dot is in the system (the service request is served - R_{sys}), it is moving in the in the lower *system* part. The time the service request is in the external client or the external client, it moves in the upper *external* part (X_{ext}). The distance between the dots can be interpreted as the inverse of the overall arrival rate $1/\lambda^{[1]}$ (inter arrival time).

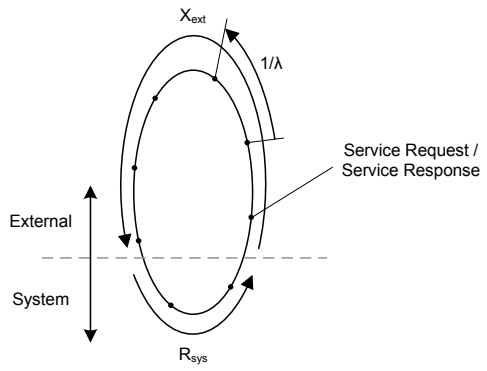


Figure 3.39: Steady State Systems - Notation

Steady State Systems - Scenarios Depending on what parameters are fixed and which ones are variable, different scenarios could be defined. Here the different, simple but representative scenarios are explained through examples, namely Paternoster, Soup Kitchen and Rollercoaster.

In the *Paternoster* example the arrival rate $\lambda^{[1]}$ and the response time $R^{[1]}$ are fixed and the total number of cycling service request n_{ges} is variable. When adding a new service request to the model, the paternoster ellipse (external time X_{ext}) will be bigger or smaller, as illustrated in figure 3.40, because the arrival rate and also the interarrival time $1/\lambda^{[1]}$ is fixed.

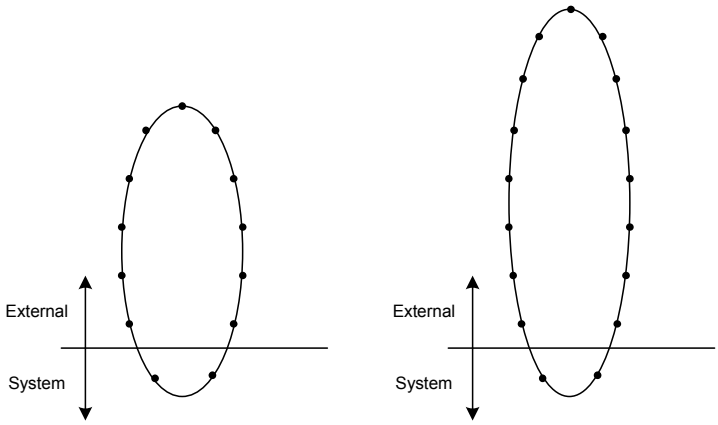


Figure 3.40: Steady State Systems - Paternoster

An example of this type could be the situation in an agency where you have to take a number in order to be served. The agency would know the response time R_{sys} and the number of customers in service n_{sys}, so they would provide the new customers with the waiting time, for that the new customers could leave the agency and could come back after that time. In that case the waiting time would be mapped to the external time X_{ext}.

The *Soup Kitchen* model, shown in figure 3.41, is different. Here the external time is always the same (people are receiving lunch every day). In this model the arrival rate is not a parameter, but a value, when adding new service requests to the model the arrival rate will grow.

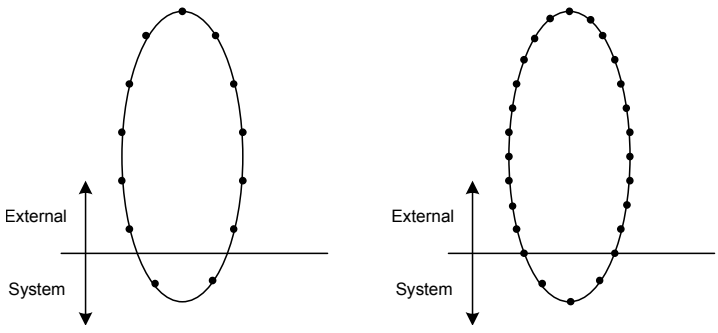


Figure 3.41: Steady State Systems - Soup Kitchen

In the third example *Roller Coaster* the external service time X_{ext} is zero. In the example the kids using the roller coasters immediately enqueue again after they took the ride, as illustrated in figure 3.42. The model can be compared to closed models. In comparison to the paternoster, in both the Soup Kitchen and the roller coaster model more customers generate a higher arrival rate and as a result of this longer queues and longer response times. But, there is a significant difference between the soup kitchen model and the roller coaster. In the soup kitchen there is still an external source and the overall number of cycling service requests has a mean value (population) but is not fixed. In the closed roller coaster there is no external source. The population is fixed and there is an internal synchronization in the model. An iterative algorithm for the calculation of closed models has also been developed and is described in section 4.1.

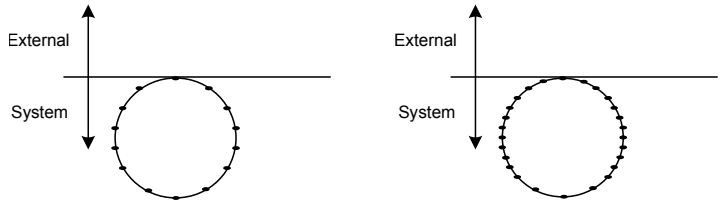


Figure 3.42: Steady State Systems - Roller coaster

For the paternoster and soup kitchen model figure 3.43 shows an example of the broad range of possible configurations of the mean population n_{ges} and the arrival rate λ. In the chart the twelve graphs are the corresponding curves for $1 \leq n_{ges} \leq 12$, where the most left curve is the curve for $n_{ges} = 1$ and the most right curve is the curve for $n_{ges} = 12$.

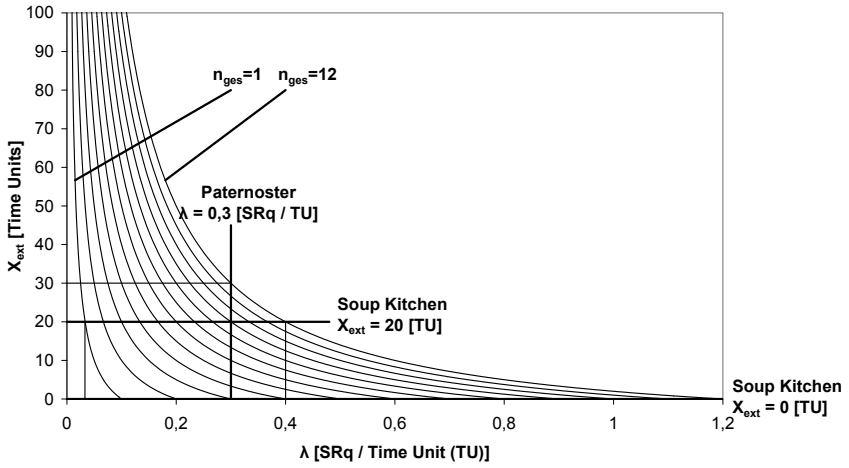


Figure 3.43: Chart External Service Time

For an exemplary paternoster (in a paternoster the arrival rate is the free parameter and the external service time is a resulting value) with $\lambda = 0,3\frac{[SRq]}{[TU]}$, the minimal population is $n_{ges} = 3$ (then $X_{ext} = 0[TU]$) and for $n_{ges} = 12$, $X_{ext} = 30[TU]$.

For soup kitchen scenarios the chart is read in the different direction (X_{ext} is the free parameter and λ is the resulting value). In the example a soup kitchen with $X_{ext} = 20[TU]$, the resulting arrival rates are in the range from $\lambda = 0,033\frac{[SRq]}{[TU]}$ for $n_{ges} = 1$ to $\lambda = 0,4\frac{[SRq]}{[TU]}$ for $n_{ges} =$

12. Accordingly, for $X_{ext} = 0[TU]$ the corresponding arrival rates are at the corresponding intersection points with the λ-axis of the curves 1 to 12.

More on the *Roller Coaster* scenario and the corresponding discussion of Closed Queueing Networks could be found later in section 4.1.

An exemplary *Experimental Parameters* section of the Tableau is shown in figure 3.3.

Table 3.3: Tableau Example - Experimental Parameters

Experimental Parameters	
$n_{ges}^{[1]}$	30
$\lambda_{bott}^{[1]}$	3,750
f	0,600
$\lambda^{[1]}$	2,250

3.4.3 Service Request Section

The *Service Request Section* describes the service request hierarchies with levels, indices, names, probabilities traffic flow coefficients and the arrival rates at the different hierarchical levels. The parameters in this part are:

- $[bb]$ – Hierarchical level
- i – Index
- $SRq_i^{[bb]}$ – Service request labeling
- $p_{parent(i),i}$ – Routing probability from service request parent to to service request i
- $v_{parent(i)}^{[bb-1]}$ – Absolute traffic flow coefficient on the higher hierarchical level (parent)
- $v_{i,int}^{[bb]}$ – Relative traffic flow coefficient to the higher hierarchical level
- $v_i^{[bb]}$ – Absolute traffic flow coefficient
- $\lambda_i^{[bb]}$ – Arrival rate of service requests i

The hierarchy level $[bb]$ is derived from the service request structure definition in the Entity Relationship Diagram. The top-most service request and the service request generator have hierarchy level $[1]$. The hierarchy level is then incremented by 1 for every refinement level.

The index i is a unique index for every logical service request type. The numbering is established through traversing the service request tree in an adapted postorder traversal (right, left, root). Through this traversal method the order of the service request table is also set up. The job generation, which is connected to the root of the service request tree, is the one with the highest index ($i_{JobGenerator} = i_{Root} + 1$) in this notation.

Every service request is named in the column SRq_i. This name is imported from the corresponding entity in the Entity Relationship Diagram.

The routing probability from the hierarchically higher request $(parent(i))$ to this service request ($p_{parent(i),i}$) is derived from the Petri Net. It is possible that the sum of the probabilities in one hierarchical part is less than 1, because the NOP transitions in a branch are not shown in the

corresponding Tableau. As an example: in the barbershop example, as shown in figure 3.4, the supplemental work is separated in *Dye* ($p_{Dye} = 0,3$), *Perm* ($p_{Perm} = 0,2$) and do nothing - *NOP* $p_{NOP} = 0,5$, so in this example, in the Tableau, the probabilities of the children of the *Supplemental Work* sum up to $0,5$.

The parameters $v_{parent(i)}^{[bb-1]}$, $v_{i,int}^{[bb]}$ and $v_i^{[bb]}$ specify the traffic flow coefficients of the service request. These parameters are defined in the corresponding Entity Relationship Diagram of the model and then imported in the Tableau. In detail, the parameter $v_{parent(i)}^{[bb-1]}$ imports the absolute value of the traffic flow coefficient on the next hierarchical level, the parameter $v_{i,int}^{[bb]}$ defines the value of the traffic flow coefficient relative to the next hierarchical level, and the parameter $v_i^{[bb]}$ defines the absolute value of the traffic flow coefficient of this service request multiplied by the probability of the service request:

$$v_i^{[bb]} = v_{parent(i)}^{[bb-1]} * v_{i,int}^{[bb]} * p_{parent(i),i} \tag{3.16}$$

An example: in a car service the *Change Wheels* service request is decomposed into four service requests *Change Wheel*. If the absolute value of the traffic flow coefficient of the service request *Change Wheels* would be 1 ($v_{ChangeWheels} = 1$) then the absolute traffic flow coefficient of the *Change Wheel* service request would be 4 ($4(v_i^{[bb]}) = 1(v_{parent(i)}^{[bb-1]}) * 4(v_{i,int}^{[bb]}) * 1(p_{parent(i),i})$) (under the assumption, that $p_{ChangeWheels} = 1$).

The last parameter in this part is the individual arrival rate of the service request. The arrival rate $\lambda_i^{[bb]}$ is computed as the overall arrival rate λ multiplied by the traffic flow coefficient of the service request:

$$\lambda_i^{[bb]} = \lambda * v_i^{[bb]} \tag{3.17}$$

The unit of the arrival rate is $[Service\ Requests_i^{[bb]} / Time\ Unit]$. The traffic flow coefficient does not only describe the value transformation, it also describes the transformation of the top level service request ($SRq[1]$) to the service request on this hierarchical level ($SRq_i^{[bb]}$).

An exemplary *Service Request Section* of the corresponding Tableau is shown in table 3.4.

Table 3.4: Tableau Example - Service Request Section

Service Request Section							
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$
2	1	Cash	1,00	1,00	1,00	1,00	2,250
3	2	Blow-Dry	0,50	1,00	1,00	0,50	1,125
2	3	Dry	1,00	1,00	1,00	1,00	2,250
3	4	Perm	0,30	1,00	1,00	0,30	0,675
3	5	Dye	0,20	1,00	1,00	0,20	0,450
2	6	Supp. Work	1,00	1,00	1,00	1,00	2,250
3	7	Cut2	0,60	1,00	1,00	0,60	1,350
3	8	Cut1	0,40	1,00	1,00	0,40	0,900
2	9	Cut	1,00	1,00	1,00	1,00	2,250
2	10	Wash	1,00	1,00	1,00	1,00	2,250
1	11	Barbershop Serv.	1,00	1,00	1,00	1,00	2,250
1	12	Job Generation	1,00	1,00	1,00	1,00	2,250

3.4.4 Server Section

The *Server Section* of the FMC-QE Calculus defines the performance parameters of the different logical servers. In this part logical multiplicities (like processes and threads) and the mapping to the multiplexer servers are defined. The parameters in this part are:

- $Server_i$ – Labeling of the logical server
- $m_{parent(i)}^{[bb-1]}$ – Absolute value of the logical multiplicity on the next hierarchical level (parent)
- $m_{i,int}^{[bb]}$ – Value of the logical multiplicity relative to the next hierarchical level
- $m_i^{[bb]}$ – Absolute value of the logical multiplicity
- Mpx_i – Corresponding multiplexer server
- $X_i^{[bb]}$ – Measured service time for every request at the corresponding multiplexer

$Server_i$ labels the corresponding logical server of each service request. This parameter is originally defined in the Entity Relationship Diagram and the Block Diagram of the model.

The multiplicity of the logical service requests (logical server multiplicity) is hierarchically defined through the parameters $m_{parent(i)}^{[bb-1]}$, $m_{i,int}^{[bb]}$ and $m_i^{[bb]}$. This logical multiplicity could be compared to the multiplicity of threads and processors. It defines how many service requests could be logically processed in parallel. It addition to this, the multiplicity of the multiplexer servers (in the example: processors) would be defined in the *Multiplexer Section*. This logical multiplicity of the service request server structure is originally defined in the corresponding Block Diagram of the model. The definition is very similar to the definition of the traffic flow coefficients. $m_{parent(i)}^{[bb-1]}$ denotes the multiplicity of the hierarchical parent of the logical server, $m_{i,int}^{[bb]}$ specifies the value of the multiplicity relative to the next hierarchical level and $m_i^{[bb]}$ defines the absolute value of the multiplicity:

$$m_i^{[bb]} = m_{parent(i)}^{[bb-1]} * m_{i,int}^{[bb]} \tag{3.18}$$

The mapping of a logical basic server (server of a operational service request - the one which actually does the work) to a multiplexer server is defined through the parameter Mpx_i. This parameter is originally defined in the Block Diagram through the mapping matrix. This parameter is empty for non leaf nodes (*Control Service Requests*) of the service request tree.

The parameter $X_i^{[bb]}$ defines the measured service time for each service request processed at the corresponding multiplexer server. This value is taken under the assumption that the multiplexer server does not process other logical service requests at that moment (no multiplex). Of course, FMC-QE with the Tableau is suitable to handle multiplex scenarios, but for the sake of normalization this assumption is made. In multiplex scenarios the measurements have to be measured on dedicated (non multiplexer) servers or have to be normalized. In the model this value is also defined in the matrix of the Block Diagram.

An exemplary corresponding *Server Section* of the Tableau is shown in table 3.5.

Table 3.5: Tableau Example - Server Section

Server Section					
$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$
Cashier	1	1	1	1	0,056
Blow-Dryer	1	∞	∞	4	0,350
Dryer	1	1	1		
Permer	1	1	1	3	0,350
Dyer	1	2	2	2	0,333
Supp.	1	1	1		
Cut2-Cutter	1	1	1	2	0,333
Cut1-Cutter	1	1	1	3	0,300
Cutter	1	1	1		
Washer	1	3	3	1	0,111
Server	1	1	1		
Generator	1	1	1		10,572

3.4.5 Dynamic Evaluation Section

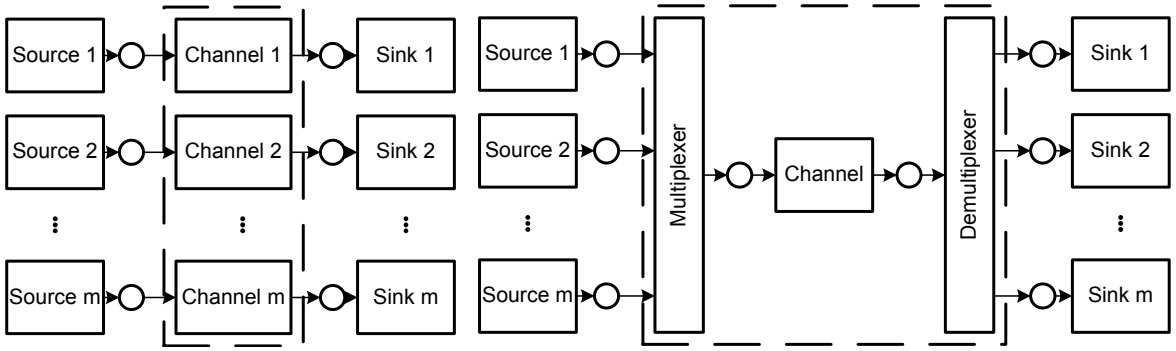
The performance values of the service request structures are computed in the *Dynamic Evaluation Section*. The values in this part are:

- $m_{i,mpx}^{[bb]}$ – Multiplex coefficient
- $\mu_i^{[bb]}$ – Service rate
- $\rho_i^{[bb]}$ – Utilization
- $n_{i,q}^{[bb]}$ – Mean number of queued service requests
- $W_i^{[bb]}$ – Mean waiting time
- $n_{i,s}^{[bb]}$ – Mean number of service requests in service
- $Y_i^{[bb]}$ – Mean service duration
- $n_i^{[bb]}$ – Mean number of service requests in the station
- $R_i^{[bb]}$ – Mean response time

In the next paragraphs the different formulas for the calculation of the performance values of an operational service request (leaf node - Basic Server Station *BSSt*) and a control service request (non leaf node - Hierarchical Server Station *HSSt*) are described.

Multiplexer - Operational Service Request

In FMC-QE it is possible that a multiplexer server is in a role of more than one logical server (real multiplex). The multiplexer server then is partitioned into several fractions, corresponding for every logical server. This fraction is called multiplex coefficient ($m_{i,mpx}^{[bb]}$). A multiplexer in signal transmission is shown in figure 3.44.



(a) User View (b) System View

Figure 3.44: Multiplexer/Demultiplexer [130]

The corresponding activities of each of the multiplexed logical servers are handled then in parallel which results in a longer server duration Y (as the service time X stays constant). An example: as illustrated in figure 3.45, the service durations Y_i and Y_j for the handling of the a service request i and service request j handled by one multiplexer Mpx are twice as long as the corresponding service times X_i and X_j as the multiplexer handles both activities in parallel and each logical server receives half of the multiplexer ($m_{i,mpx} = m_{j,mpx} = 0,5$).

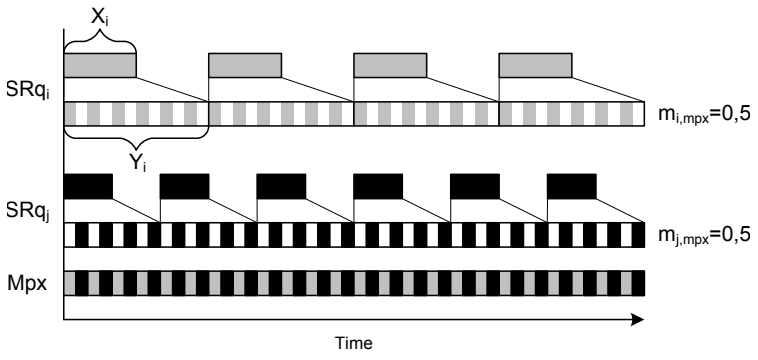


Figure 3.45: Multiplex Example [146]

In such a multiplexer scenario, as illustrated in figure 3.46, only the service requests in service are transferred to the multiplexers, the different queues for every type of service request are still at the different logical servers.

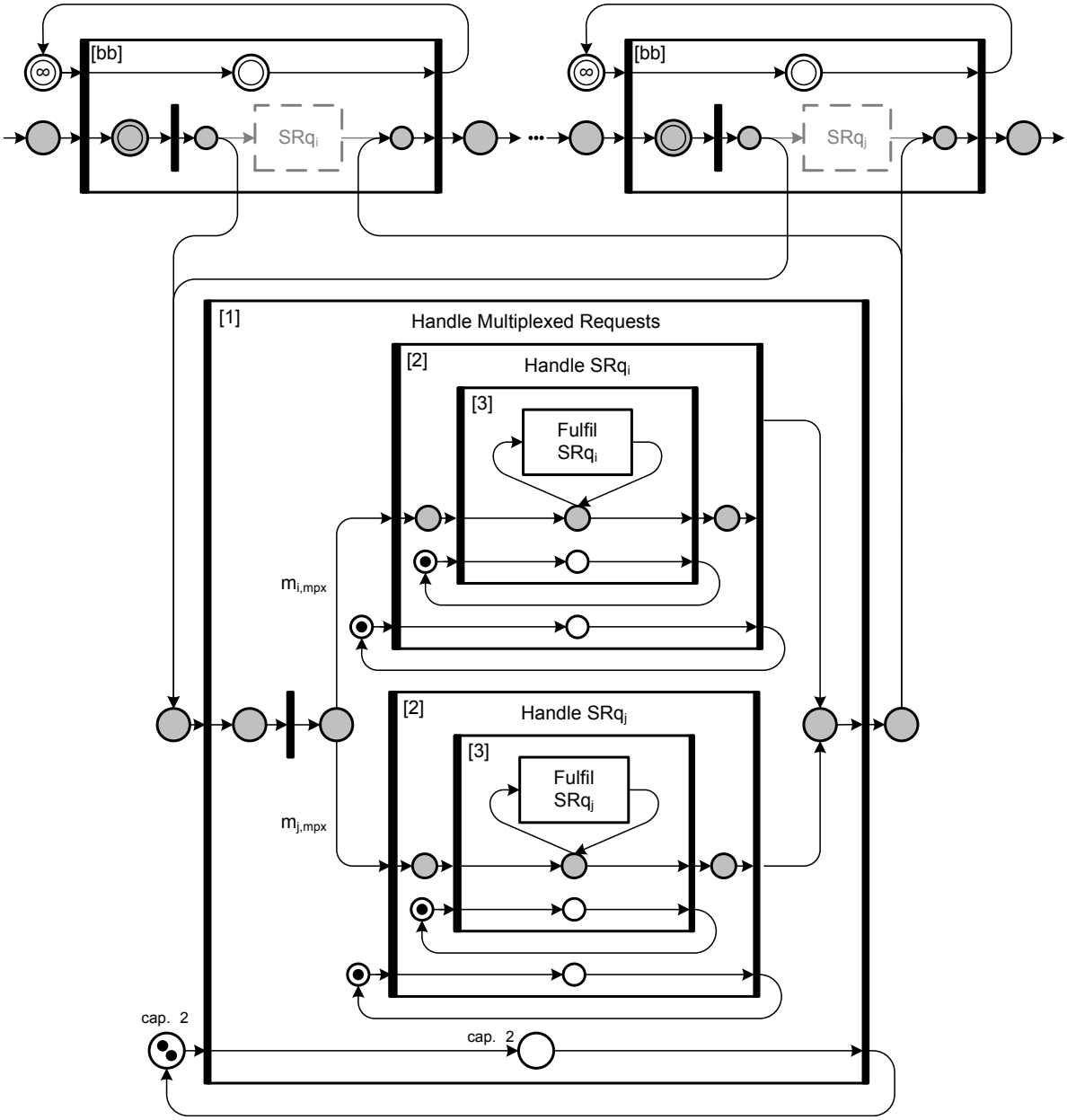


Figure 3.46: Multiplex - Service Requestor's View [146]

If the sum of all logical servers handled by a multiplexer server is less or equal than the sum of all multiplexer servers ($\sum_{\forall\, SRq_i\, of\, Mpx_j} m_i^{[bb]} \leq m_j$), every logical service request is handled by a dedicated multiplexer server - multiplex coefficient is one ($m_{i,mpx}^{[bb]} = 1$). In this case redundant multiplexer servers ($m_{j,idle} = m_j - \sum_{\forall\, SRq_i\, of\, Mpx_j} m_i^{[bb]}$) would idle. The scenario of infinite servers (both, the logical and the corresponding multiplexer servers) is a special case of equal number of logical and multiplexer servers and the multiplex coefficient is also one for this scenario ($m_{i,mpx}^{[bb]} = 1$).

If the sum of logical servers handled by a multiplexer server is greater than the sum of all multiplexer servers ($\sum_{\forall\, SRq_i\, of\, Server_j} m_i^{[bb]} > m_j$), the multiplexer servers have to work in multiplex mode. The service rates in which a logical service request operates is smaller because the mul-

tiplexer servers are handling other service request types in parallel. The logical servers receive only a fraction of the multiplexer server which is defined as the multiplex coefficient $m_{i,mpx}^{[bb]}$. The multiplex coefficient is calculated as a fraction of this logical service request time $X_i^{[bb]}$ and the overall amount of service time of a multiplexer server (multiplexer) needs to handle all of the corresponding service request of a top level service request X_j multiplied by the fraction of the multiplicity of multiplexer servers (m_j) and the number of logical servers.

To summarize the calculation of the multiplex coefficient:

$$m_{i,mpx}^{[bb]} = \begin{cases} 1 & if \ \sum_{\forall \, SRq_i \, of \, Mpx_j} m_i^{[bb]} \leq m_j \\ \frac{X_i^{[bb]} * v_i^{[bb]}}{X_j} * \frac{m_j}{m_i^{[bb]}} & if \ \sum_{\forall \, SRq_i \, of \, Mpx_j} m_i^{[bb]} > m_j \end{cases} \tag{3.19}$$

In this way of calculating the multiplex coefficient and the service rate the utilization $\rho_i^{[bb]}$ is equal for every logical server handled by the same multiplexer. There are other possible ways to calculate the multiplexer coefficient if $\sum_{\forall \, SRq_i \, of \, Mpx_j} m_i^{[bb]} > m_j$, like depending only on the number of logical servers and not on the service time: $m_{i,mpx}^{[bb]} = \frac{m_j}{\sum_{\forall \, SRq_i \, of \, Mpx_j} m_i^{[bb]}}$. The used calculation was chosen because it was considered as fair, but for certain scenarios other multiplexer coefficients could be more suitable.

Coming back to the Calculus, the service rate $\mu_i^{[bb]}$ then is the reciprocal of the service time, multiplied by the multiplex coefficient:

$$\mu_i^{[bb]} = \frac{m_{i,mpx}^{[bb]}}{X_i^{[bb]}} \tag{3.20}$$

For the calculation of the utilization ($\rho_i^{[bb]}$), the mean number of queued requests ($n_{i,q}^{[bb]}$) and the mean number of requests in service ($n_{i,s}^{[bb]}$) there is a distinction between parallel logical servers and infinite logical servers.

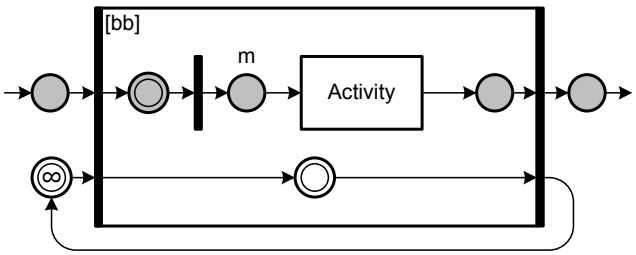


Figure 3.47: Parallel Server

For parallel logical servers, as shown in figure 3.47, the calculation of the values is defined as [67] ($M/M/m$):

$$\rho_i^{[bb]} = \frac{\lambda_i^{[bb]}}{m_i^{[bb]} \mu_i^{[bb]}} \tag{3.21}$$

$$n_{i,q}^{[bb]} = \left(\frac{\left(\frac{\lambda_i^{[bb]}}{\mu_i^{[bb]}}\right)^{m_i^{[bb]}} * \rho_i^{[bb]}}{m_i^{[bb]}! \left(1-\rho_i^{[bb]}\right)^2} \right) * \frac{1}{\sum_{k=0}^{m_i^{[bb]}-1} \frac{\left(m_i^{[bb]}\rho_i^{[bb]}\right)^k}{k!} + \left(\frac{\left(m_i^{[bb]}\rho_i^{[bb]}\right)^{m_i^{[bb]}}}{m_i^{[bb]}!} \right) \left(\frac{1}{1-\rho_i^{[bb]}} \right)} \tag{3.22}$$

$$n_{i,s}^{[bb]} = \frac{\lambda_i^{[bb]}}{\mu_i^{[bb]}} \tag{3.23}$$

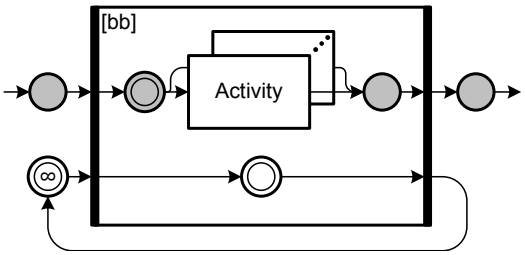


Figure 3.48: Infinite Server

For infinite logical servers, as shown in figure 3.48, the calculation of the values is defined as [88] ($M/M/\infty$):

$$\rho_i^{[bb]} = empty \tag{3.24}$$

$$n_{i,q}^{[bb]} = 0 \tag{3.25}$$

$$n_{i,s}^{[bb]} = \frac{\lambda_i^{[bb]}}{\mu_i^{[bb]}} \tag{3.26}$$

The mean number of service requests in the station ($n_i^{[bb]}$) and the times: waiting time $W_i^{[bb]}$, service duration $Y_i^{[bb]}$ and the response time $R_i^{[bb]}$ are calculated the same way for every server type. The mean number of service requests in the station ($n_i^{[bb]}$) is defined as the sum of queued service requests ($n_{i,q}^{[bb]}$) and service requests in service($n_{i,s}^{[bb]}$):

$$n_i^{[bb]} = n_{i,q}^{[bb]} + n_{i,s}^{[bb]} \tag{3.27}$$

and the waiting time ($W_i^{[bb]}$) as well as the service duration $Y_i^{[bb]}$ and the response time $R_i^{[bb]}$ are computed through Little's Law [98] as mean number of service requests in the queue ($n_{i,q}^{[bb]}$), in service ($n_{i,s}^{[bb]}$) or in the station ($n_i^{[bb]}$) divided by the arrival rate at the logical server ($\lambda_i^{[bb]}$) as:

$$W_i^{[bb]} = \frac{n_{i,q}^{[bb]}}{\lambda_i^{[bb]}}, \qquad Y_i^{[bb]} = \frac{n_{i,s}^{[bb]}}{\lambda_i^{[bb]}}, \qquad R_i^{[bb]} = \frac{n_{i,q}^{[bb]}}{\lambda_i^{[bb]}}. \tag{3.28}$$

In contrast to the service time X, the service duration Y is not a parameter, but a resulting value. If the corresponding basic server station is not multiplexed, the service duration equals the service time, but if the basic server station is multiplexed, the service duration is the sum of the service time and the time the multiplexer handles the other multiplexed basic server stations in parallel (elapsed time).

Control Service Request

For *Control Service Requests* (non leaf nodes) the FMC-QE Calculus provides the handling of:

- Hierarchical Activities,
- Serial Activities,
- Parallel Activities,
- Branches and
- Loops.

The service rate of the composed service request ($\mu_i^{[bb]}$) is the minimum of all service request rates of the serial children divided by the internal traffic flow coefficients and the probabilities:

$$\mu_i^{[bb]} = \min \left(\frac{\mu_{child(i)}^{[bb+1]} * m_{child(i),int}^{[bb+1]}}{v_{child(i),int}^{[bb+1]} * p_{i,child(i)}} \right) \forall \, children \, of \, i \tag{3.29}$$

The utilization ($\rho_i^{[bb]}$) is empty for all *Control Service Requests*:

$$\rho_i^{[bb]} = empty \tag{3.30}$$

The calculation of mean number of service requests in the station ($n_i^{[bb]}$) and the response time ($R_i^{[bb]}$) is the same for all *Control Service Requests*:

The mean number of service requests in the station ($n_i^{[bb]}$) is defined as the sum of the queued service requests ($n_{i,q}^{[bb]}$) and the service requests in service ($n_{i,s}^{[bb]}$):

$$n_i^{[bb]} = n_{i,q}^{[bb]} + n_{i,s}^{[bb]} \tag{3.31}$$

As in the basic server station the waiting time ($W_i^{[bb]}$), the service duration $Y_i^{[bb]}$ and the response time $R_i^{[bb]}$, according to Little's Law [98], are calculated as the mean number of service requests in the queue ($n_{i,q}^{[bb]}$), in service ($n_{i,s}^{[bb]}$) or in the station ($n_i^{[bb]}$) divided by the arrival rate at the logical server ($\lambda_i^{[bb]}$) as:

$$W_i^{[bb]} = \frac{n_{i,q}^{[bb]}}{\lambda_i^{[bb]}}, \qquad Y_i^{[bb]} = \frac{n_{i,s}^{[bb]}}{\lambda_i^{[bb]}}, \qquad R_i^{[bb]} = \frac{n_{i,q}^{[bb]}}{\lambda_i^{[bb]}}. \tag{3.32}$$

The other different formulas for the calculation of the mean number of queued service requests ($n_{i,q}^{[bb]}$) and the mean number of service requests in service($n_{i,s}^{[bb]}$) are provided in the next paragraphs.

Hierarchical Activity A hierarchical activity with one activity inside, as shown in figure 3.49, is just a special case of the other hierarchical decompositions like serial activities or parallel activities, but nevertheless the formulas are shown here.

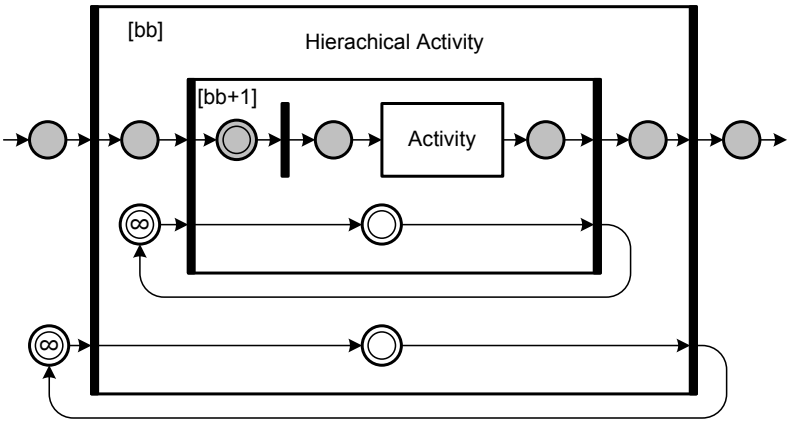


Figure 3.49: Hierarchical Activities

$$n_{i,q}^{[bb]} = n_{child(i),q}^{[bb+1]} \tag{3.33}$$

$$n_{i,s}^{[bb]} = n_{child(i),s}^{[bb+1]} \tag{3.34}$$

Serial Activity In a serial activity service request, as shown in figure 3.50, the service request is hierarchically decomposed into different serial activities.

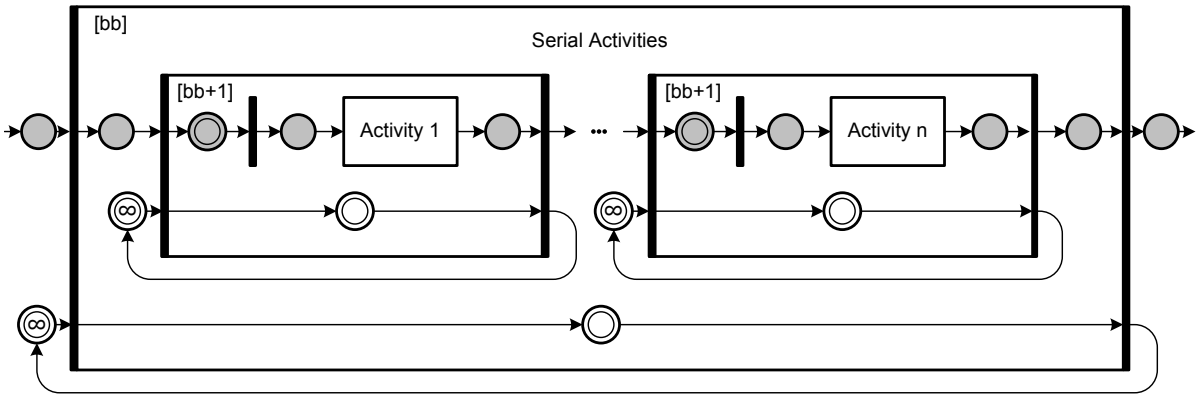


Figure 3.50: Serial Activities

The number of queued service requests ($n_{i,q}^{[bb]}$) and the number of service requests in service ($n_{i,s}^{[bb]}$) in a serial activity service request is the sum of the corresponding values of the single serial activities:

$$n_{i,q}^{[bb]} = \sum_{\forall children\ of\ i} n_{child(i),q}^{[bb+1]} \tag{3.35}$$

$$n_{i,s}^{[bb]} = \sum_{\forall children\ of\ i} n_{child(i),s}^{[bb+1]} \tag{3.36}$$

Parallel Activities In a parallel activity service request, as shown in figure 3.51, the service request is hierarchically decomposed into different activities which are processed in parallel.

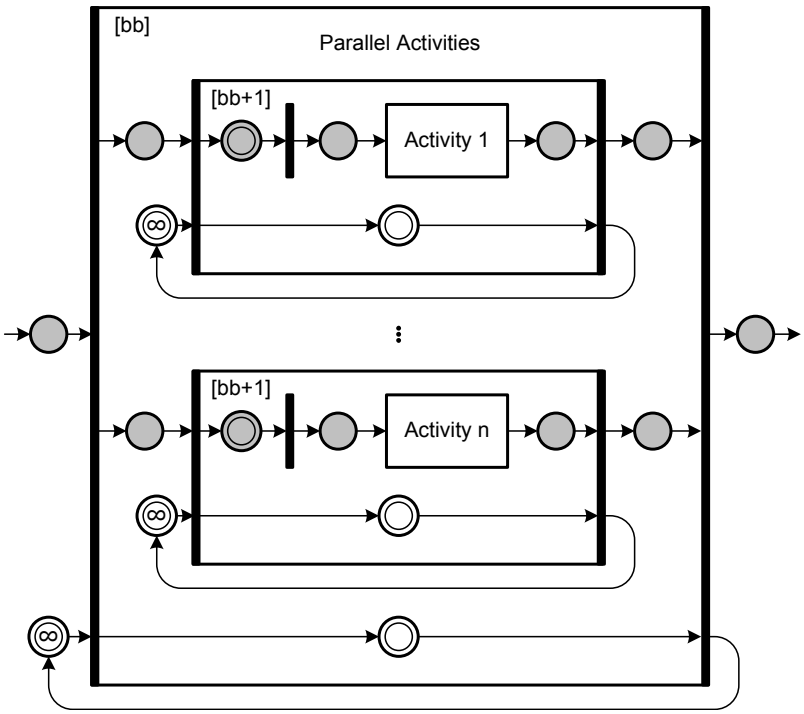


Figure 3.51: Parallel Activities

The number of queued service requests ($n_{i,q}^{[bb]}$) and the number of service requests in service ($n_{i,s}^{[bb]}$) in a parallel activity service request are:

$$n_{i,q}^{[bb]} = \max(n_{child(i),q}^{[bb+1]})\ \forall\ children\ of\ i \tag{3.37}$$

$$n_{i,s}^{[bb]} = \max(n_{child(i),s}^{[bb+1]})\ \forall\ children\ of\ i \tag{3.38}$$

Branch In a branch activity service request, as shown in figure 3.51, the service request is hierarchically decomposed into different activities, where each of it is processed with the probability p.

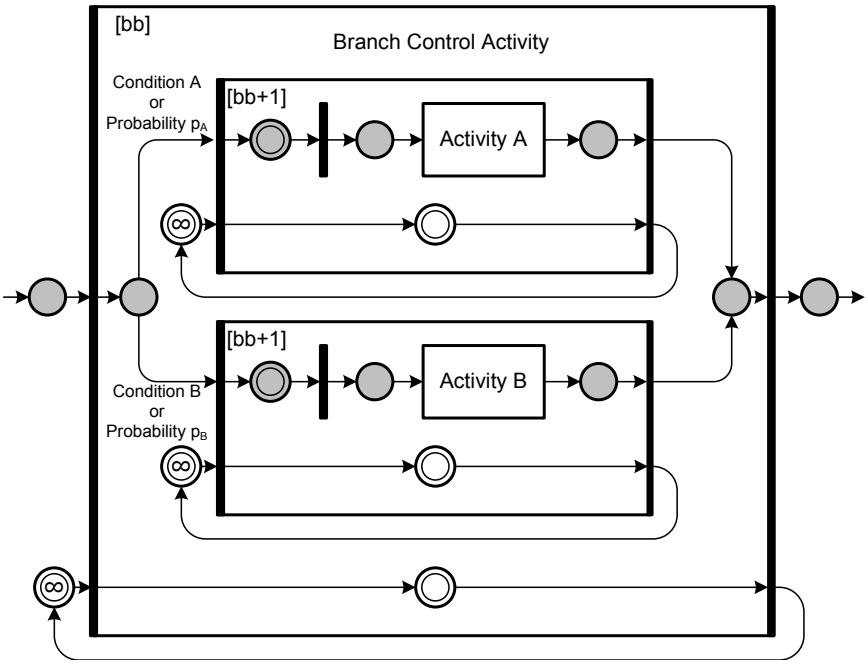


Figure 3.52: Branch

The number of queued service requests ($n_{i,q}^{[bb]}$) and the number of service requests in service ($n_{i,s}^{[bb]}$) in a branch activity service request is the sum of the corresponding values of the single serial activities:

$$n_{i,q}^{[bb]} = \sum_{\forall children\ of\ i} n_{child(i),q}^{[bb+1]} \tag{3.39}$$

$$n_{i,s}^{[bb]} = \sum_{\forall children\ of\ i} n_{child(i),s}^{[bb+1]} \tag{3.40}$$

Loops In FMC-QE the system is modeled from the perspective of the hierarchical service request structures processed by the system. In order to evaluate the model through the FMC-QE Calculus, the model must follow a tree shaped structure. If the modeled system does not follow this assumption, due to modeling with a different modeling technique or special system behavior, the model is transformed into this tree shape structure. Examples for this behavior are loops. Loops are transformed, and after the transformation the behavior of the transformed model is equivalent to the behavior of the original loop but analyzable with the FMC-QE Calculus.

While Loop: In FMC-QE a while loop is transformed in order to compute the performance values of the model. Figure 3.53 shows an FMC-QE model of a while loop. The loop body is represented by the execution of the service request SRq_i.

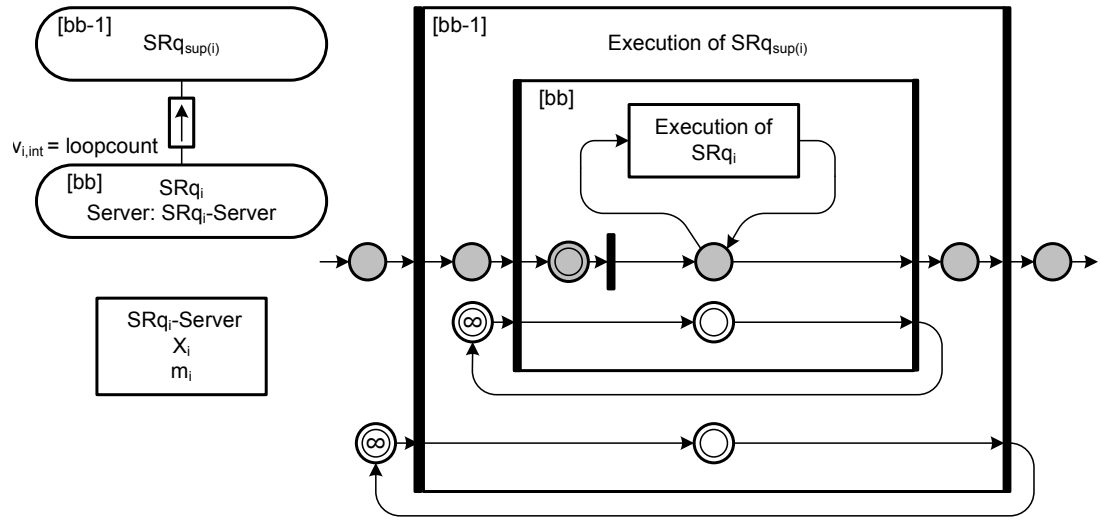


Figure 3.53: Original While Loop

In the service request structures the loop is represented by a hierarchy of service requests. The service request $SRq_{sup(i)}$ represents the whole execution of all iterations, while the service request SRq_i represents one single loop body execution. The traffic flow coefficient $v_{i,int}$ represents the mean number of loop iterations. In the Petri Net the inner service request loop represents the while loop and the corresponding multiplexer server has the multiplicity m_i and the Service time X_i.

In the first step of the transformation the loop is transformed to a sequence of executions of the loop body (SRq_i), as shown in figure 3.54.

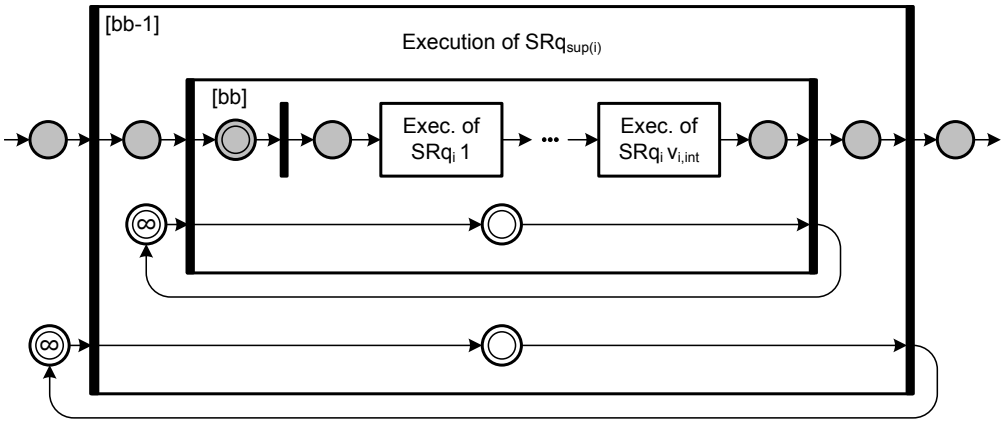


Figure 3.54: While Loop Transformation (Serialization)

Then the single executions of the loop body are combined to one execution, in which the service time is the sum of the single service times. This step is shown in figure 3.55.

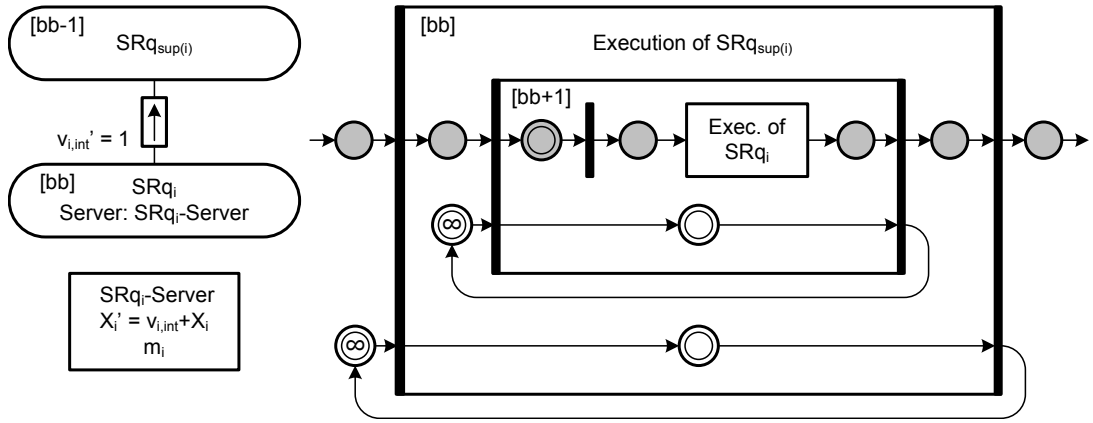


Figure 3.55: While Loop Transformation (Combination)

In this transformation the arrival rate of the service requests SRq_i is not changed:

$$\lambda_i^{'[bb]} = \lambda_i^{[bb]}, \tag{3.41}$$

but the service time is the sum of the single executions of the loop bodies, respectively the product of the mean loopcount $v_{i,int}$ and the original service time X_i:

$$X_i^{'[bb]} = v_{i,int} * X_i^{[bb]} \tag{3.42}$$

Feed Backward Loop: In FMC-QE feed backward loops, as shown in figure 3.56, are transformed into a feed forward execution, shown in figure 3.57.

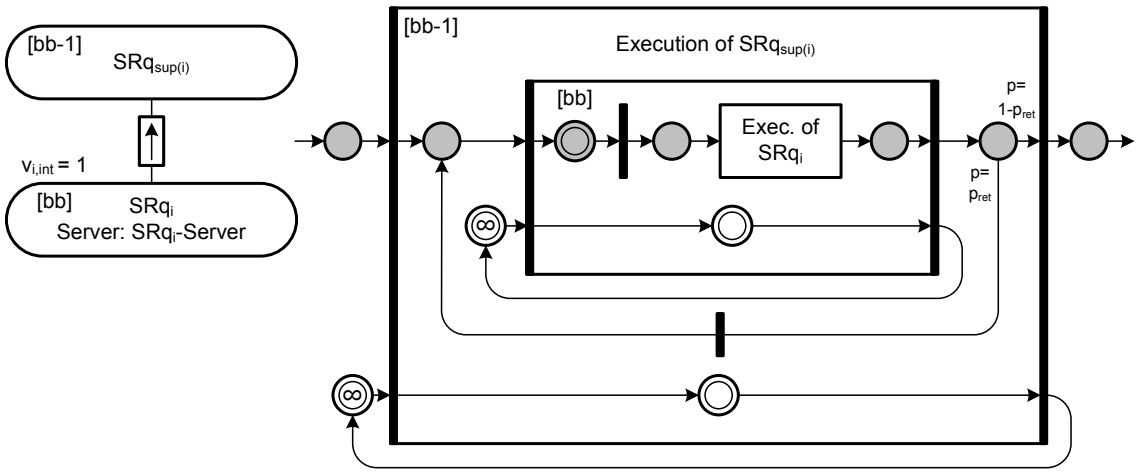


Figure 3.56: Feed Backward Loop

In the feed backward execution there is a probability p_{ret} that the execution service request SRq_i has to be repeated (e.g. failure).

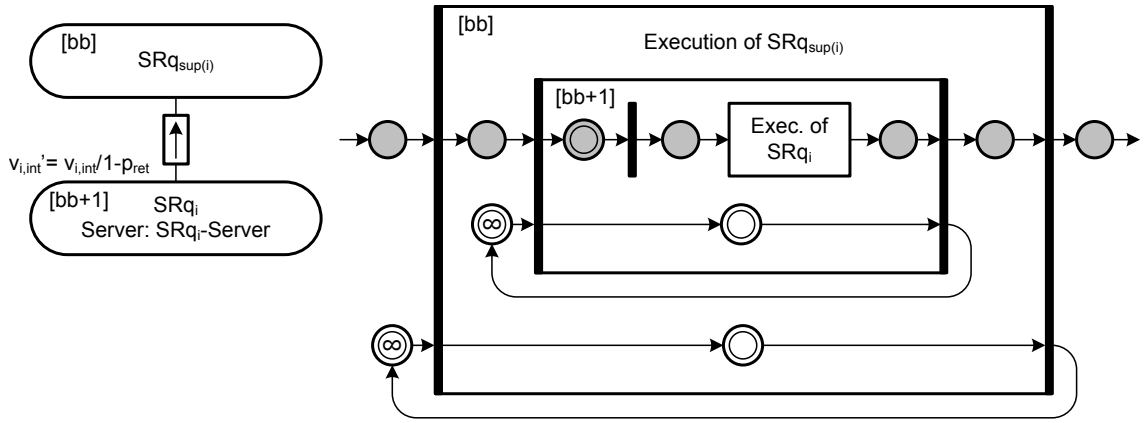


Figure 3.57: Feed Forward

In the feed backward - feed forward transformation not the service time X_i but the arrival rate λ_i is transformed. The increased arrival rate better corresponds to the real system than the extension of the service rate. The new arrival rate λ_i' is the product of the traffic flow coefficient $v_{i,int}$ and the old arrival rate λ_i:

$$\lambda_i^{'[bb]} = v_{i,int}^{'[bb]} * \lambda_i^{[bb]} \tag{3.43}$$

The traffic flow coefficient is calculated due to the formula of the geometric sum [16, 65]:

$$s = \sum_{n=1}^{\infty} a_1 q^{n-1} = \frac{a_1}{1-q} \tag{3.44}$$

as:

$$v_{i,int}^{'[bb]} = \frac{v_{i,int}^{[bb]}}{1 - p_{return}} \tag{3.45}$$

In comparison to the transformation of the while loop the service time X is not transformed:

$$X_i^{'[bb]} = X_i^{[bb]} \tag{3.46}$$

After the transformation the performance values are calculated like a normal hierarchical activity.

An exemplary *Dynamic Evaluation Section* of the Tableau is shown in table 3.6.

Table 3.6: Tableau Example - Dynamic Evaluation Section

Dynamic Evaluation Section								
$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
1,000	18,000	0,125	0,018	0,008	0,125	0,056	0,143	0,063
1,000	2,857		0,000	0,000	0,394	0,350	0,394	0,350
	∞		0,000	0,000	0,394	0,175	0,394	0,175
0,467	1,333	0,506	0,519	0,769	0,506	0,750	1,025	1,519
0,125	0,375	0,600	0,675	1,500	1,200	2,667	1,875	4,167
	3,750		1,194	0,531	1,706	0,758	2,900	1,289
0,750	2,250	0,600	0,900	0,667	0,600	0,444	1,500	1,111
0,533	1,778	0,506	0,519	0,577	0,506	0,563	1,025	1,139
	3,750		1,419	0,631	1,106	0,492	2,525	1,122
1,000	9,000	0,083	0,000	0,000	0,250	0,111	0,250	0,111
	3,750		2,631	1,169	3,581	1,592	6,212	2,761
	0,095		0,000	0,000	23,788	10,572	23,788	10,572

3.4.6 Multiplexer Section

In the *Multiplexer Section* the overall parameters of the multiplexer servers are defined to support the calculation of the individual performance values of the multiplexed logical servers, defined in the operational service request part of the dynamic evaluation section, explained in section 3.4.5.

The parameters in this part are:

- j – Index
- $Name_j$ – Multiplexer name
- m_j – Multiplicity
- $X_j^{[1]}$ – Aggregated normalized service time

In $Server_j$ a server name is defined. This parameter is imported from the corresponding Block Diagram of the model.

The parameter m_j is also imported from the Block Diagram and defines the multiplicity of every multiplexer server.

$X_j^{[1]}$ is the service time a server needs to serve its parts of the top level service request. It is defined as the normalized sum of all measured corresponding service times.

$$X_j^{[1]} = \sum_{\forall mpxed\ SRq_i} X_i^{[bb]} * v_i^{[bb]} \tag{3.47}$$

For infinite multiplexer servers ($m_j = \infty$) the aggregated normalized service time $X_j^{[1]}$ is empty and the multiplexer coefficient $m_{i,mpx}^{[bb]}$ will be 1 for each logical server i connected to the infinite server j.

In figure 3.7 an exemplary *Multiplexer Section* of a Tableau is shown.

Table 3.7: Tableau Example - Multiplexer Section

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Appentice	5	0,167
2	Barber	1	0,267
3	Barber Boss	1	0,225
4	Customer	∞	

3.4.7 Computation Algorithm / Complexity Analysis

In order to analyze the complexity of the FMC-QE performance predictions and the FMC-QE Calculus with the corresponding Tableau, the algorithm of the derivation of the values will be sketched.

In the first step the different parameters of the service request structure tree and the corresponding server structures and dynamic behavior model are imported. Therefore, the tree(s) - multiple trees in multiclass scenarios - are traversed in an adapted postorder traversal (right left root). These parameters are: $n_{ges}^{[1]}$ and f as experimental parameters, $[bb]$, $SRq_i^{[bb]}$ and $v_{i,int}^{[bb]}$ from the Entity Relationship Diagram, $Server_i^{[bb]}$, $m_{i,int}^{[bb]}$, $Mpx_i^{[bb]}$ and $X_i^{[bb]}$ from the Block Diagram and $p_{parent(i),i}^{[bb]}$ and the execution order from the Petri Net. The parameters $v_{parent(i)}^{[bb]}$, $v_i^{[bb]}$, $m_{parent(i)}^{[bb]}$ and $m_i^{[bb]}$ have only top-down interconnections and could therefore computed as a by-product of the tree traversal. Also the index i is defined in the traversal algorithm (order of traversal). Also as a part of the import procedure, the multiplexer parameters $Name_j$ and m_j are extracted from the Block Diagram and the index j is defined. Therefore, the list of multiplexers in the Block Diagram has to be traversed, which is by definition shorter than the number of nodes in the service request tree (there can not be more multiplexers than logical servers). While in the first step the model tree is created and traversed, the complexity of this step is O(n) where n is the number of service request nodes.

In the second step the service request table is traversed in order to aggregate the service times $X_i^{[bb]}$ for every multiplexer to compute X_j. Therefore, every row in the service request table has to be traversed (O(n) - n=number of service request nodes) and the multiplexer section has also to be traversed once (O(n)).

In the third step the multiplex coefficient $m_{i,mpx}^{[bb]}$ and the service rates $\mu_i^{[bb]}$ are computed. Therefore, the service request table is traversed from $i = 1$ to $i = n$. In order to compute the multiplex coefficient, the multiplex section has also be traversed for every basic logical server. In this service request table traversal the minimum of all normalized logical service rates (see 3.11) is saved in order to compute $\lambda_{bott}^{[1]}$. While in this step the service request table has to be traversed twice and the multiplexer table has to be traversed once, the complexity is again O(n).

In the forth and last step the service request table is traversed one time from $i = 1$ to $i = n$ in order to compute $\lambda_i^{[bb]}$, $\rho_i^{[bb]}$, $n_{i,q}^{[bb]}$, $n_{i,s}^{[bb]}$, $n_i^{[bb]}$ and $R_i^{[bb]}$. The values of the logical basic servers (leaf nodes in service request tree) can be computed directly and the logical control servers have to aggregate the values of the sub service requests. In this step the service request table has also to be traversed once and for every node some calculations (only parameters of the node) have to be performed. Therefore, the complexity is again O(n).

While all steps in the calculation of the Tableau are performed with a complexity of O(n), where n is the number of service request nodes, the calculation of the whole Tableau is also performed with a linear complexity of O(n).

3.5 FMC-QE Example - Open Queueing Network

In the following example an open Queueing Network (taken from [18] - Example 7.4, page 287f) is modeled and analyzed. This example has been chosen in order to provide a full example for FMC-QE and to compare the performance predictions of an exemplary open Queueing Network, computed through Queueing Theory methods with the performance predictions of FMC-QE. Another goal of this example is an exemplary description of the transformation of a Queueing Network into an FMC-QE model.

3.5.1 Original Model and Calculation

The original model is shown in figure 3.58.

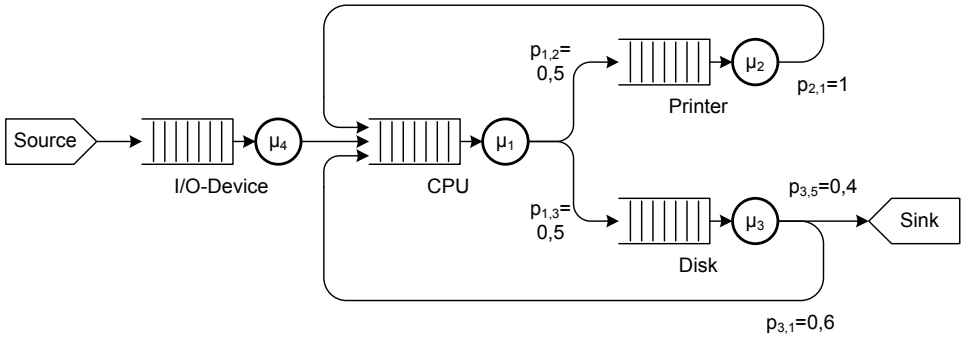


Figure 3.58: Open Queueing Example - Original Model [18]

In this example four single server FCFS nodes with exponentially distributed service times are connected. The mean service rates are [18]:

$$\mu_1 = 25\,\frac{[Jobs]}{[s]};\ \mu_2 = 33\frac{1}{3}\,\frac{[Jobs]}{[s]};\ \mu_3 = 16\frac{2}{3}\,\frac{[Jobs]}{[s]};\ \mu_4 = 20\,\frac{[Jobs]}{[s]}$$

The example is an open Queueing Network with an exponentially distributed arrival rate of [18]:

$$\lambda = \lambda_{0,4} = 4\,\frac{[Jobs]}{[s]}.$$

In [18] the performance values are calculated using Jacksons Theorem (see 2.1.4). In the first step the arrival rates are calculated [18]:

$$\lambda_1 = \lambda_2 p_{2,1} + \lambda_3 p_{3,1} + \lambda_4 p_{4,1} = 20\,\frac{[Jobs]}{[s]}; \quad \lambda_2 = \lambda_1 p_{1,2} = 10\,\frac{[Jobs]}{[s]}$$
$$\lambda_3 = \lambda_1 p_{1,3} = 10\,\frac{[Jobs]}{[s]}, \qquad \lambda_4 = \lambda_{0,4} = 4\,\frac{[Jobs]}{[s]}.$$

In the second step the different performance values are derived. The utilizations λ_i are calculated as $\rho_i = \frac{\lambda_i}{\mu_i}$ [18]:

$$\rho_1 = \frac{\lambda_1}{\mu_1} = 0,8;\ \rho_2 = \frac{\lambda_2}{\mu_2} = 0,3;\ \rho_3 = \frac{\lambda_3}{\mu_3} = 0,6;\ \rho_4 = \frac{\lambda_4}{\mu_4} = 0,2.$$

The mean number of jobs at a node for an M/M/1 node are calculated as $n_i = \frac{\rho_i}{1-\rho_i}$ (see table A.3) and therefore [18]:

$$n_1 = 4\,[Jobs];\ n_2 = 0,429\,[Jobs];\ n_3 = 1,5\,[Jobs];\ n_4 = 0,25\,[Jobs].$$

The mean response times of the servers are computed with the help of Little's Law [98] as $R_i = \frac{n_i}{\lambda_i}$ [18]:

$$R_1 = 0,2\,[s];\ R_2 = 0,043\,[s];\ R_3 = 0,15\,[s];\ R_4 = 0,0625\,[s].$$

The mean overall response time is calculated by aggregating the mean number of jobs at every node and Little's Law [18]:

$$R = \frac{n}{\lambda} = \frac{1}{\lambda}\sum_{i=1}^{4} n_i = 1,545\,[s]$$

The mean queue lengths are calculated as $n_{i,q} = \frac{\rho_i^2}{1-\rho_i}$ (see table A.3) [18]:

$$n_{1,q} = 3,2\,[Jobs];\ n_{2,q} = 0,129\,[Jobs];\ n_{3,q} = 0,9\,[Jobs];\ n_{4,q} = 0,05\,[Jobs].$$

Therefore, the mean waiting times (calculated as $W_i = \frac{\rho_i}{\mu_i - \lambda_i}$ - (see table A.3)) are [18]:

$$W_1 = 0,16\,[s];\ W_2 = 0,013\,[s]\ W_3 = 0,09\,[s];\ W_4 = 0,0125\,[s].$$

In the original example in [18] the marginal probabilities and the state probabilities are also computed, but in FMC-QE these values are not considered and so this is omitted here.

3.5.2 Transformation

In order to compare these results with the corresponding FMC-QE model, the Queueing Theory model will be transformed into the corresponding FMC-QE model. In the first step an initial Petri Net, shown in figure 3.59, was set up.

In this first Petri Net the behavior of the different servers is represented. The transitions are now labeled with the actions (*Retrieve Input*, *Compute Results*, *Print*, *Save to Disk*) of the different servers and not with the names (*I/O-Device*, *CPU*, *Printer*, *Disk*) as in the Queueing Theory server structures, because the Petri Net represents the behavioral view of the model, which is only implicitly modeled in the Queueing Theory. This first Petri Net has no hierarchies (flat) and is still an open network.

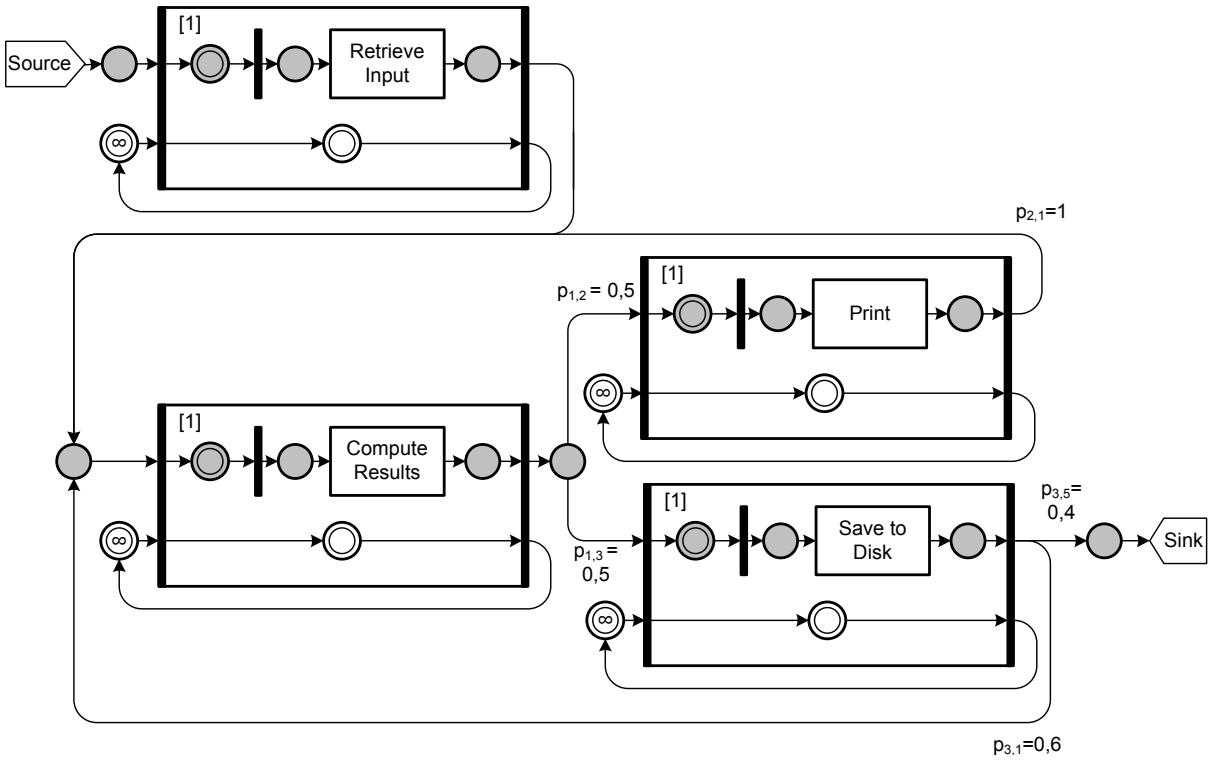


Figure 3.59: Open Queueing Example - Initial Petri Net

Due to the fact that in FMC-QE models there is no distinction between open and closed queueing networks and the outside world is always modeled (see 3.4.2), in the second transformation step the load generation, represented by the action *Generate Request*, is added as shown in figure 3.60.

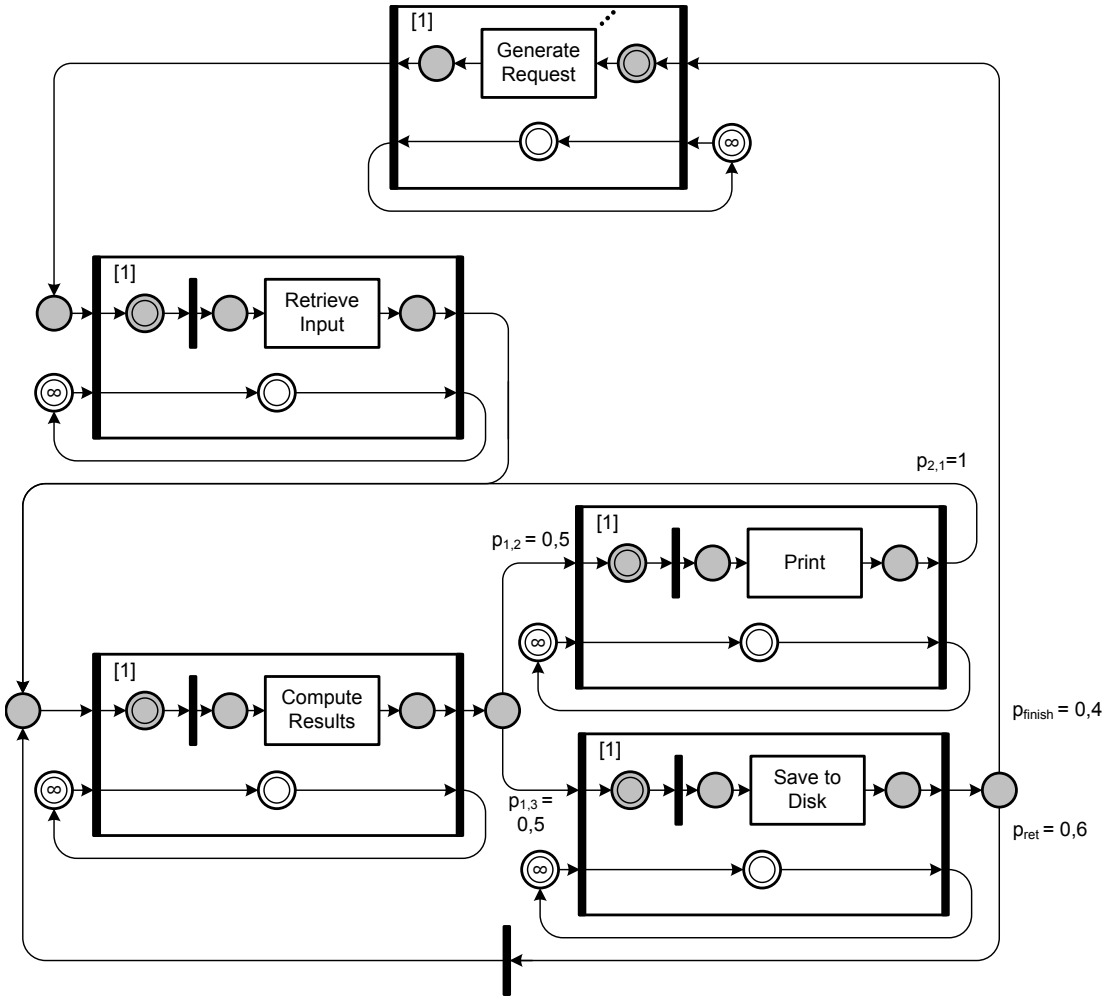


Figure 3.60: Open Queueing Example - Load Generation

An open network with a fixed arrival rate is then represented by a modified paternoster model (see 3.4.2) in which the external time is calculated as a result from the parameters mean overall number of service requests $n_{ges}^{[1]}$, arrival rate $\lambda^{[1]}$ and overall mean response time $R_{sys}^{[1]}$:

$$X_{ext}^{[1]} = (\frac{n_{ges}^{[1]}}{\lambda^{[1]}}) - R_{sys}^{[1]}. \tag{3.48}$$

The mean overall number of service requests $n_{ges}^{[1]}$ therefore have to be greater than one and thus have to be greater than the mean number of service requests in the system $n_{sys}^{[1]}$:

$$n_{ges}^{[1]} \geq n_{sys}^{[1]} = R_{sys}^{[1]} \lambda^{[1]}. \tag{3.49}$$

In the third step a first hierarchy, the *Persist Data Branch*, is introduced. This hierarchy abstracts from the decision if the results are *Printed* or *Saved to the Disk*. Though this abstraction the return probability p_{ret} and the finish probability $p_{finish} = 1 - p_{ret}$ where adjusted in order to add the 50% of the printed and then returned service requests:

$$p_{ret} = p_{1,2} * p_{2,1} + p_{1,3} * p_{ret} = p_{1,2} * p_{2,1} + p_{1,3} * p_{3,1} = 0,8.$$

The new Petri Net is shown in figure 3.61.

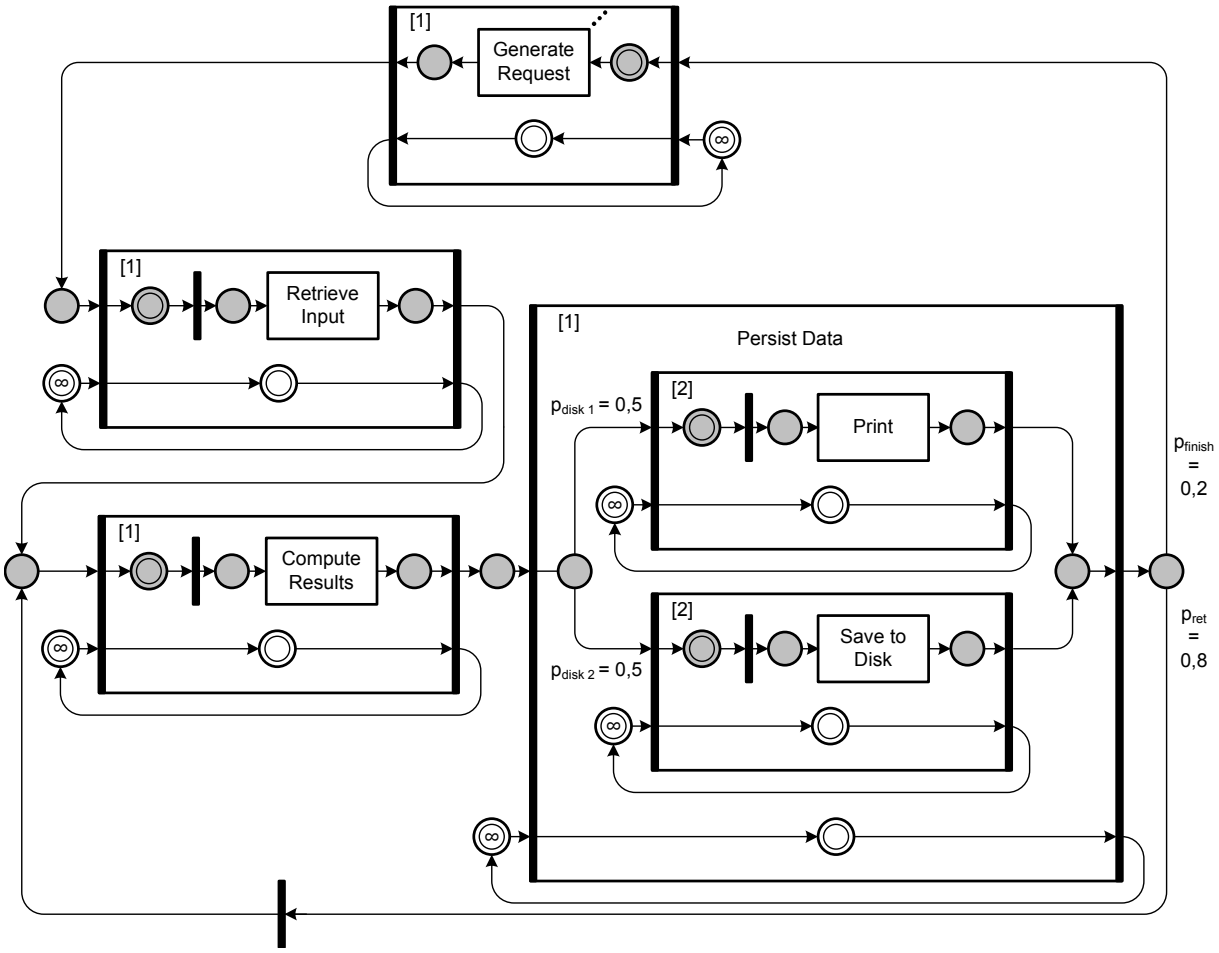


Figure 3.61: Open Queueing Example - Persist Data Branch

In the next step, shown in figure 3.62, the serial execution of the *Result Computation* and the *Persist Data Request* are abstracted to the *Unreliable Execution*. It is an unreliable execution because a return loop envelopes this execution.

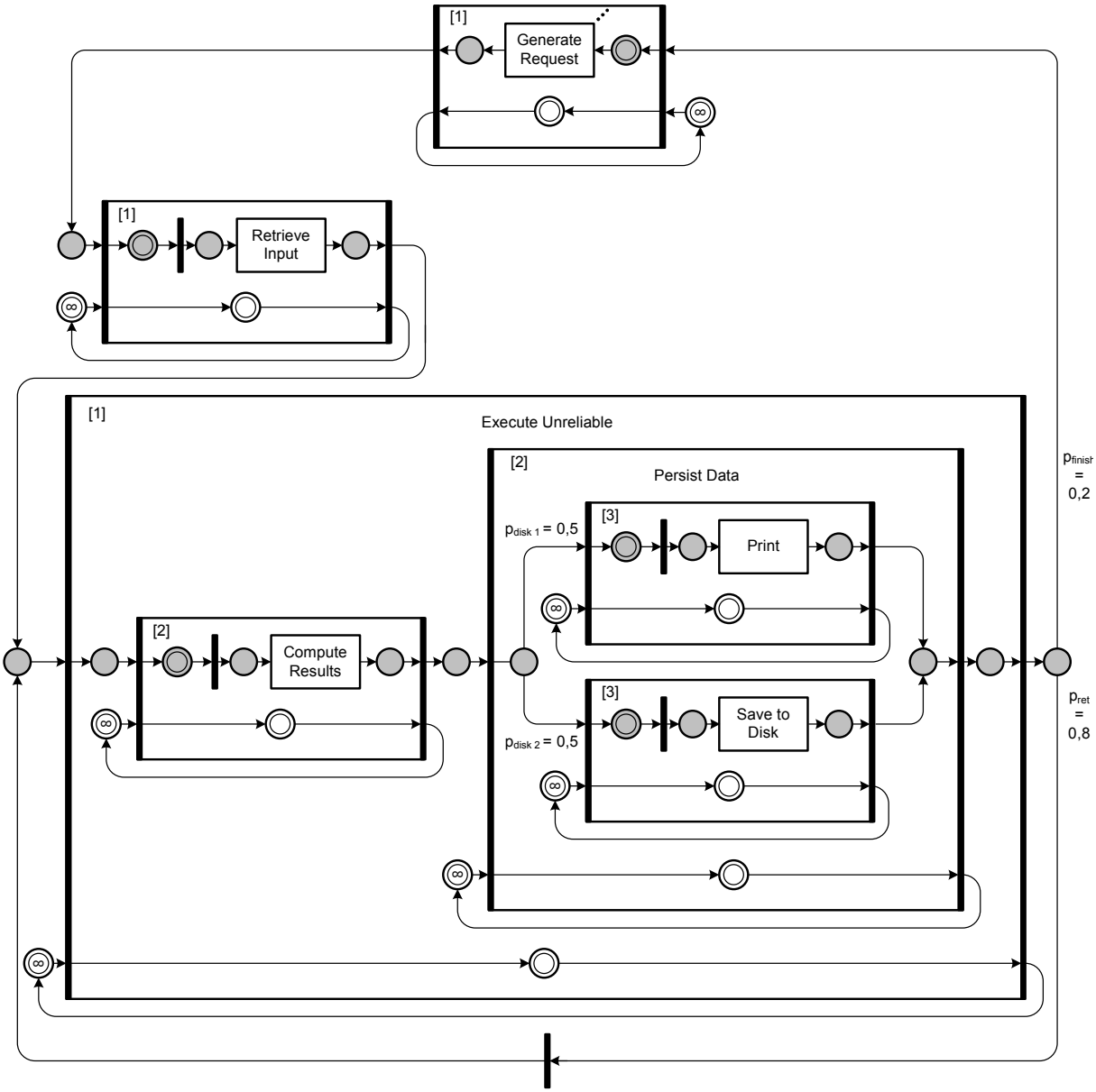


Figure 3.62: Open Queueing Example - Unreliable Execution

The return feed backward loop around the *Unreliable Execution* is resolved through transforming the feed backward loop into an feed forward execution, as shown in figure 3.63. The new *Reliable Execution* has an internal traffic flow coefficient $v_{i,int}^{'[bb]}$ calculated as the geometric sum of the returns:

$$v_{i,int}^{'[bb]} = \frac{v_{i,int}^{[bb]}}{1 - p_{return}}. \tag{3.50}$$

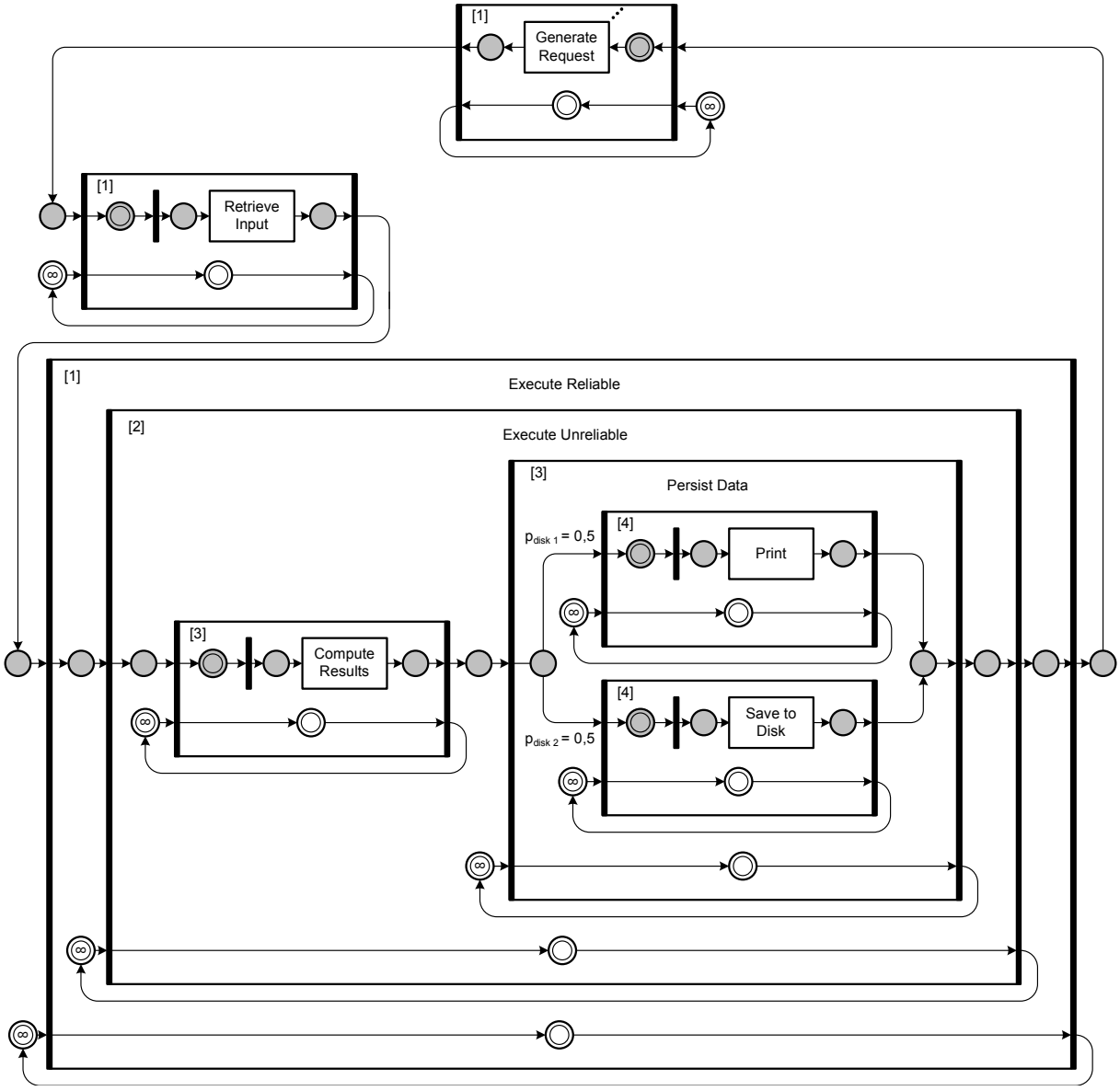


Figure 3.63: Open Queueing Example - Feed Forward - Feed Backward

Finally the serial execution of *Input* and *Reliable Execution* is abstracted to *Request*, which is the top-level ([1]) service request generated by the clients. The final transformed Petri Net is shown in figure 3.64.

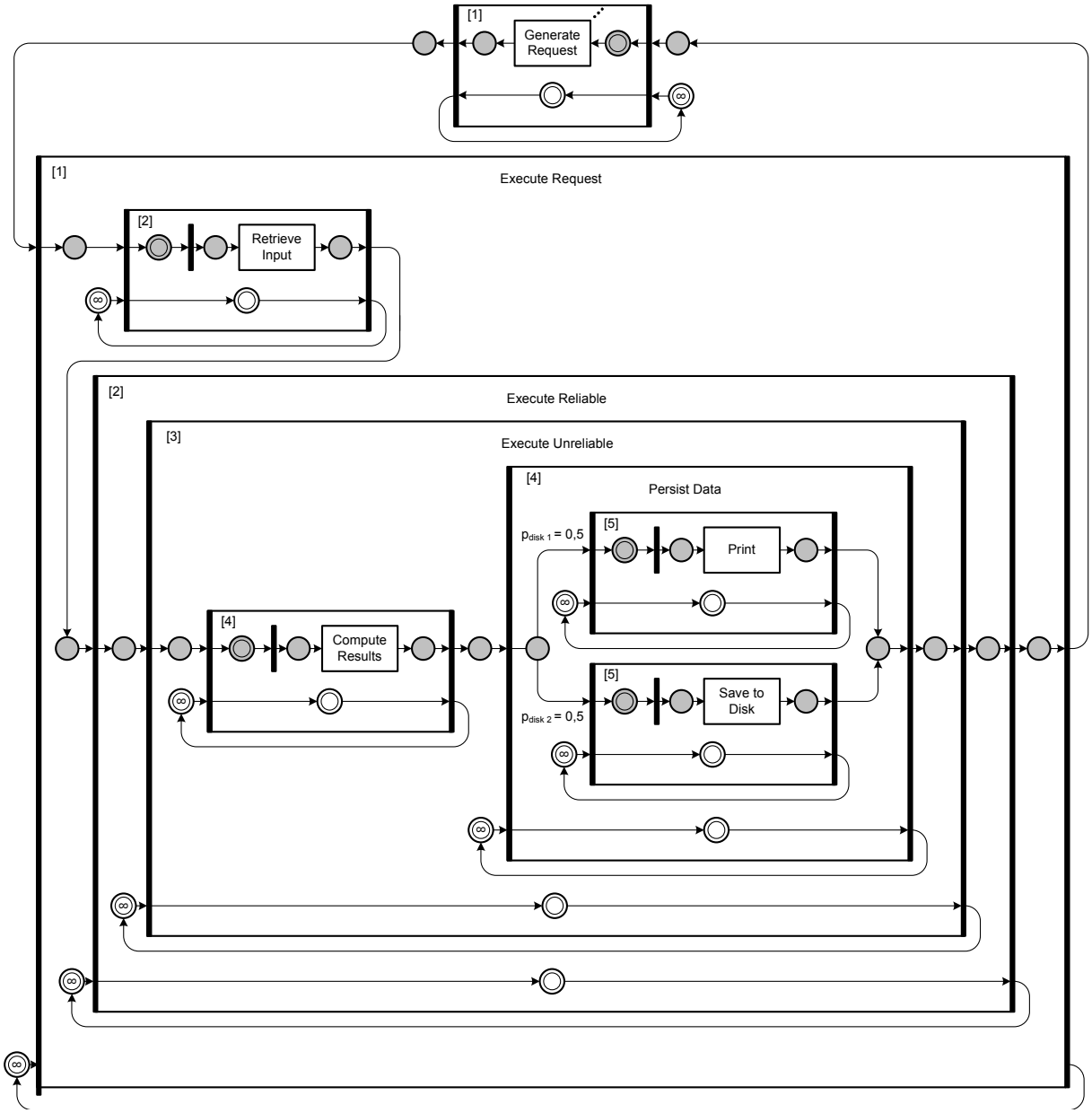


Figure 3.64: Open Queueing Example - Transformed Petri net

3.5.3 Service Request Structure and Static Structure

The corresponding service request structures of the example are shown in figure 3.65. The service request is partitioned into 5 hierarchical levels. In this diagram the increased traffic flow between the *Reliable Execution* and the *Unreliable Execution* ($v_{int} = 5\ \frac{[UnreliableExecutionRequests]}{[ReliableExecutionRequests]}$) is visualized.

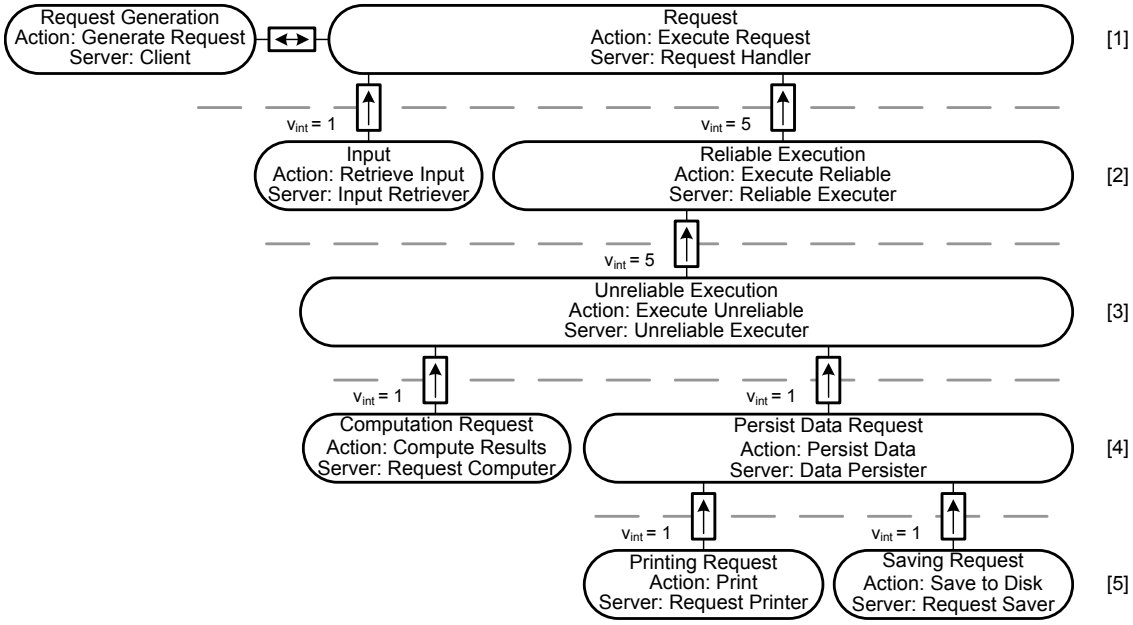


Figure 3.65: Open Queueing Example - Service Request Structures

The third diagram of the model, the server structures are represented in the Block Diagram, as shown in figure 3.66. In this model the five hierarchies of the logical server structures and the mappings to the four multiplexer servers (*CPU*, *Printer*, *Disk* and *I/O-Device*) are defined. The different service times for the basic servers are also defined in this diagram.

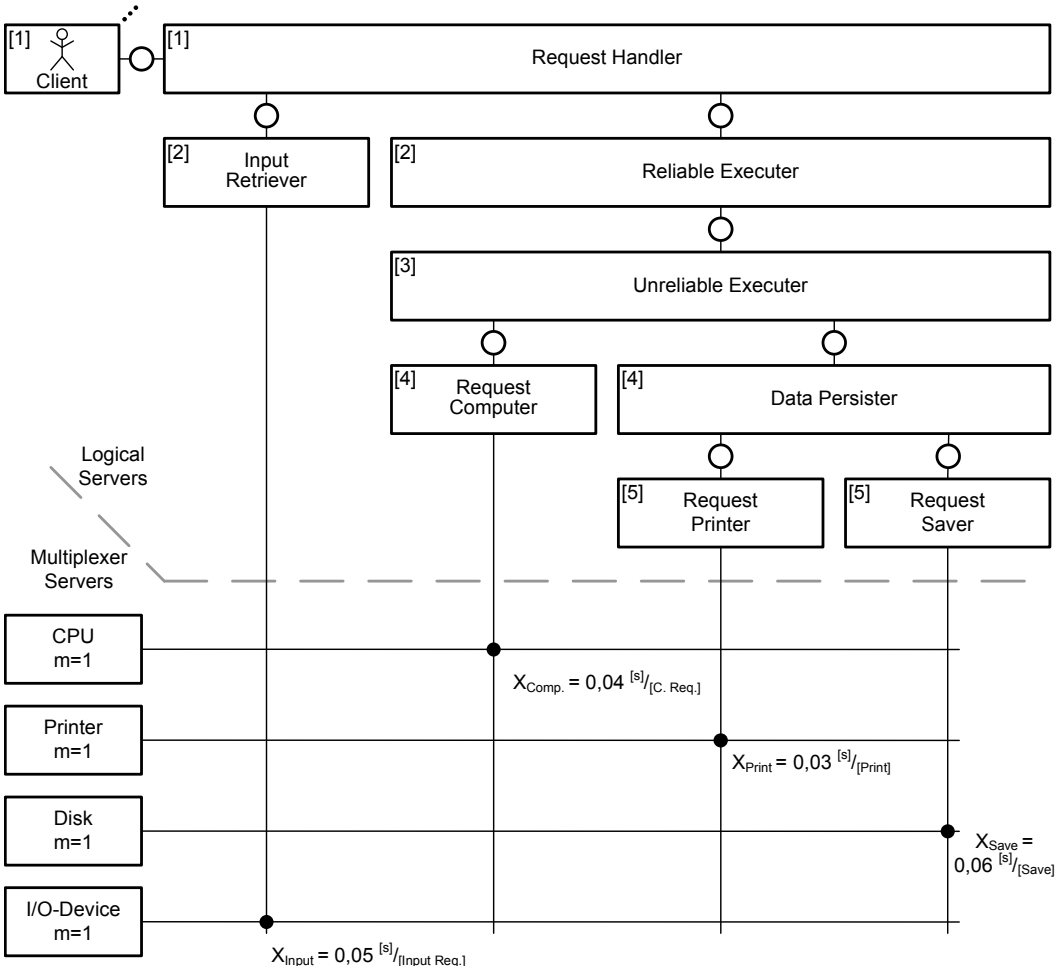


Figure 3.66: Open Queueing Example - Server Structures

3.5.4 Summary

FMC-QE delivers exact solutions for this open Product From Queueing Networks. Through transformations the flat example could be transformed to a hierarchical model. After the transformation a broad range of performance values could be calculated in the FMC-QE Tableau, as shown in table 3.8.

Table 3.8: Open Queueing Example - Tableau (see Appendix - Table B.2)

Experimental Parameters	
$n_{ges}^{[1]}$	80
$\lambda_{bott}^{[1]}$	5,0000
f	0,9000
$\lambda^{[1]}$	4,5000

Service Request Section								Server Section						Dynamic Evaluation Section									
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
5	1	Saving Request	0,50	5,00	1,00	2,50	11,250	Request Saver	1	1	1	3	0,060	1,000	16,667	0,675	1,402	0,125	0,675	0,060	2,077	0,185
5	2	Printing Request	0,50	5,00	1,00	2,50	11,250	Request Printer	1	1	1	2	0,030	1,000	33,333	0,338	0,172	0,015	0,338	0,030	0,509	0,045
4	3	Persist Data Request	1,00	5,00	1,00	5,00	22,500	Data Persister	1	1	1				33,333		1,574	0,070	1,013	0,045	2,586	0,115
4	4	Computation Request	1,00	5,00	1,00	5,00	22,500	Request Computer	1	1	1	1	0,040	1,000	25,000	0,900	8,100	0,360	0,900	0,040	9,000	0,400
3	5	Unreliable Execution	1,00	1,00	5,00	5,00	22,500	Unreliable Executer	1	1	1				25,000		9,674	0,430	1,913	0,085	11,586	0,515
2	5	Reliable Execution	1,00	1,00	1,00	1,00	4,500	Reliable Executer	1	1	1				5,000		9,674	2,150	1,913	0,425	11,586	2,575
2	6	Input	1,00	1,00	1,00	1,00	4,500	Input Retriever	1	1	1	4	0,050	1,000	20,000	0,225	0,065	0,015	0,225	0,050	0,290	0,065
1	7	Request	1,00	1,00	1,00	1,00	4,500	Request Handler	1	1	1				5,000		9,739	2,164	2,138	0,475	11,877	2,639
1	8	Request Generation	1,00	1,00	1,00	1,00	4,500	Client	1	1	1		15,139		0,066		0,000	0,000	68,123	15,139	68,123	15,139

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	CPU	1	0,200
2	Printer	1	0,075
3	Disk	1	0,150
4	I/O-Device	1	0,050

One example could be the dependency of overall response time R_{ges} from the overall arrival rate λ, as shown in figure 3.67. In this chart it can be seen that for an arrival rate from approximately $\lambda > 4,3\frac{[SRq]}{[s]}$ the overall response time R_{ges} begins to grow rapidly and goes to infinity for $\lambda = \lambda_{bott} = 5\frac{[SRq]}{[s]}$.

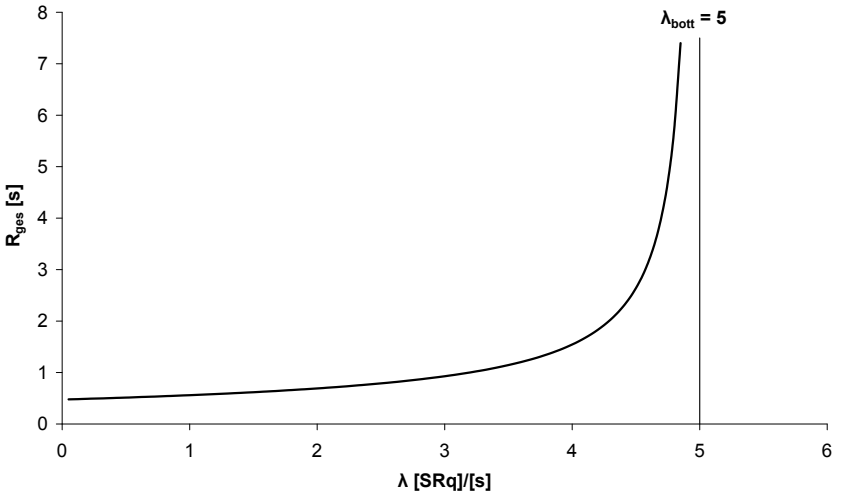


Figure 3.67: Open Queueing Example - Chart: Response Time - Arrival Rate

Another example is the dependency of the external service time X_{ext} on the mean overall number of service requests in the model (mean population n_{ges}), as depicted in figure 3.68. This type of chart could help in drawing conclusions to the behavior of the clients. In an example of

a mean population of $n_{ges} = 80[SRq]$ every single client could request the system every $15,1s$ in a configuration of a desired mean overall arrival rate of $\lambda = 4,5\frac{[SRq]}{[s]}$ (resp. $f = 0,9$).

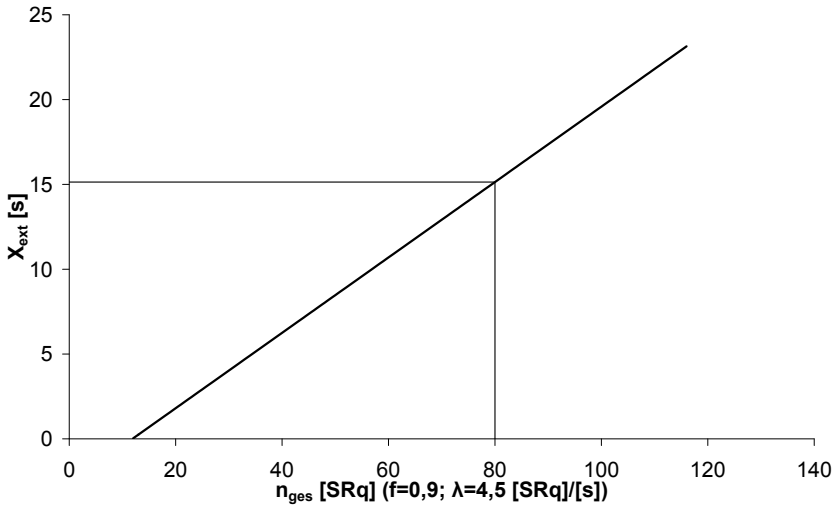


Figure 3.68: Open Queueing Example - Chart: External Service Time - Population

Figure 3.69 shows the changes of the utilization of the different servers ρ_j depending on the arrival rate λ. While the *CPU* is the bottleneck, the utilization of the CPU grows until $\rho_{CPU} = 1$ for $\lambda = \lambda_{bott} = 5\frac{[SRq]}{[s]}$. The *Disk*, the *Printer* and the *I/O Device* are the respectively lower utilized servers with a utilization of $\rho_{Disk} = 0,8$, $\rho_{Printer} = 0,375$ and $\rho_{I/O\ Device} = 0,25$ for $\lambda = \lambda_{bott}$. In addition to the utilization of the different servers the corresponding overall response time R_{ges} is also plotted.

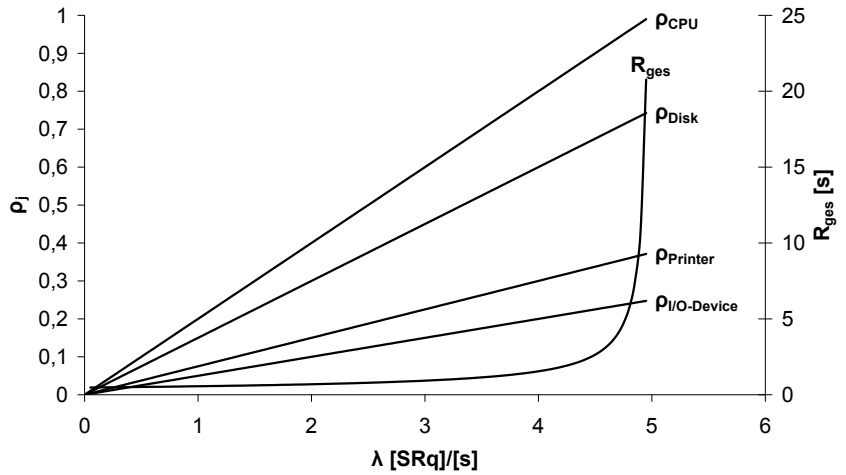


Figure 3.69: Open Queueing Example - Chart: Utilization, Response Time - Arrival Rate

3.6 FMC-QE Tool

The development of an FMC-QE Tool is in the main focus of Tomasz Porzucek, who is also a member of the Research Group of Prof. Dr.-Ing. Werner Zorn. While this thesis is focused on the theoretical background and the further development of FMC-QE, his thesis will be focused on the development of the FMC-QE framework and research questions in the development of the Tool, like model transformations. Through this topic there are close cooperations between the author and Mr. Porzucek, as in [115]. Figure 3.70 shows a screenshot of a prototype of the tool.

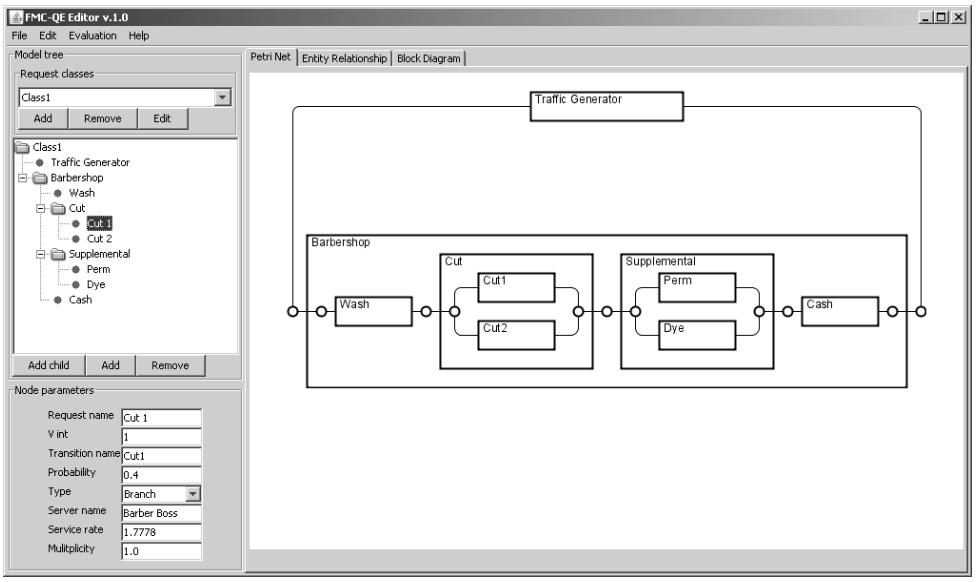


Figure 3.70: FMC-QE Tool - Screenshot [115]

With the help of the Tool the performance analyst is able to model the quantitative behavior of the analyzed system. From this model a Tableau is generated, which could be evaluated with a broad range of different system and load parameters in order to predict many different system scenarios.

Additionally, existing models, which are not modeled using FMC-QE and are possibly not hierarchical, could also be imported in this tool, as described in [116].

Chapter 4

FMC-QE Extensions

This chapter provides extensions of FMC-QE, explained with the help of representative basic examples as well as comparisons of FMC-QE to related work.

In section 4.1 FMC-QE is extended to handle Closed Queueing Networks. In this section, the integration of the summation method [17] into FMC-QE is explained in order to support the approximative evaluation for the class of Closed Queueing Systems. Section 4.2 illustrates the handling of multiclass scenarios in FMC-QE. In section 4.3 a classical Time Augmented Petri Net problem, the semaphore synchronization, is modeled and the methodology is extended to handle such scenarios. In section 4.4 related work is compared to FMC-QE. This includes the Queueing Theory, Time Augmented Petri Nets, Layered Queueing Networks (LQN) and performance simulations.

4.1 Closed Queueing Networks

In the following section the extension of FMC-QE for the handling of closed queueing networks is described. Therefore, first the general model of closed queueing networks is discussed. Then the performance predictions for exemplary closed queueing networks, modeled and computed through Queueing Theory approaches, are compared to the predictions of the corresponding FMC-QE model and Tableau and the extensions of FMC-QE are explained with the help of these models.

4.1.1 General Discussion

In closed queueing networks there is a short circuit from the system output to the system input. In the moment the service request is fulfilled and the service response is delivered from the system, this response is immediately inserted in the system, again. In section 3.4.2 this model was referenced as the *Roller Coaster* (the children instantly queue up again). From the viewpoint of FMC-QE the common model of closed queuing networks raises some questions. The first question comes from the distinction of service requests and service responses. In the steady state of closed queueing networks the numerical value of the throughput (service response rate) equals the arrival rate (service request rate), but the service request is not the same as the service response. The handling of the service response was done for some reason and therefore it makes no sense to immediately queue up the response again. Even in the roller coaster model the response (child did the ride and is happy) has to be changed into the service request (child wants to ride again). Therefore, the consideration of the external (outside) world is important, because even if the external service time would be zero, the external world is to be considered for the generation of the service request and the receiving of the service response.

Another question concerning the classical closed model also results from the neglection of the outside world and the service request generation: 'Who inserted the service requests in the system?' or 'Who initialized the system?'. In the standard model there is no service request generation and therefore, the single servers are not of type M/M/m (or similarly) but -/M/m. The output (throughput) of the system is transformed (*or not* - as discussed) into the arrival rate. Therefore, there is no service request generation M/M/m. The arrivals are a result of the network throughput -/M/m. As the arrival rate λ is not a free parameter in this model, also the basic laws, like Little's Law [98] ($n = \lambda * R$), have to be interpreted differently. While in standard FMC-QE models the number of service requests in the system (n_{sys}) is a result, here n_{sys} is the parameter and the arrival rate (or throughput) is the result.

From the viewpoint of FMC-QE closed queueing networks and the not modeling of the request generation and outside world is questionable, but as closed queueing networks are used as a standard model in a broad range of literature, this class of models will also be handled in FMC-QE. The extensions of FMC-QE for closed networks are described in this section. The extensions will be discussed on examples in order to exemplary compare the different approaches. In this examples first the networks are modeled and analyzed with Queueing Theory techniques, and later on an FMC-QE model and Tableau are set up. It will be seen that the standard FMC-QE performance evaluation techniques are not a good approximation for this class of models. Therefore, the methodology will be extended in order to solve this class of problems.

4.1.2 Closed Tandem Network

As a simple but significant example a tandem network consisting of two servers connected to each other in a closed network will be examined. First the example will be calculated using standard Queueing Theory approaches, calculating the global balance equations and solving a linear equation system. Then the example is calculated using the standard FMC-QE load model and setting the external time to zero. It will be seen that there are large approximation errors in this model. In a second Tableau the performance values are calculated using an M/M/1/K model. This model leads to correct solutions for this special case of a closed network, but in order to compute the performance values, some results of the calculations had to be known in advance. In the third model the summation method [17] is adapted to FMC-QE. The adaption will be explained and it will be seen that this is a good approximation method.

Original Example -/M/1

The original model, shown in figure 4.1, is defined in [18].

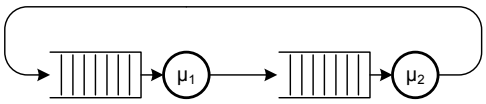


Figure 4.1: Closed Tandem Network - Original Model [18]

In this example the two servers have exponentially distributed service times with mean values of 5s ($\mu_1 = \frac{1}{5}\left[\frac{SRq_1}{s}\right]$) and 2,5s ($\mu_1 = \frac{1}{2,5}\left[\frac{SRq_1}{s}\right]$) and a FCFS service discipline. There are 3 service requests in the network ($n_{ges} = 3$), which lead to the state transition diagram shown in figure 4.2 [18]:

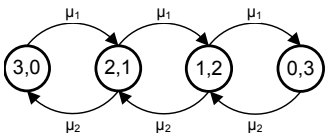


Figure 4.2: Closed Tandem Network - State Transition Diagram [18]

In [18] the global balance equations are set up as:

$$\begin{aligned} p(3,0)\mu_1 &= p(2,1)\mu_2; \\ p(2,1)(\mu_1+\mu_2) &= p(3,0)\mu_1 + p(1,2)\mu_2; \\ p(1,2)(\mu_1+\mu_2) &= p(2,1)\mu_1 + p(0,3)\mu_2; \\ p(0,3)\mu_2 &= p(1,2)\mu_1. \end{aligned}$$

This leads to the steady state probabilities [18]:

$$p(3,0) = 0,5333;\ p(2,1) = 0,2667;\ p(1,2) = 0,1333;\ p(0,3) = 0,0667.$$

Using this steady state probabilities, the marginal probabilities are computed as [18]:

$$p_1(0) = p_2(3) = p(0,3) = 0,0667;$$
$$p_1(1) = p_2(2) = p(1,2) = 0,1333;$$
$$p_1(2) = p_2(1) = p(2,1) = 0,2667;$$
$$p_1(3) = p_2(0) = p(3,0) = 0,5333.$$

After computing these probabilities, the performance values, starting with the utilization ρ_i, could be derived as [18]:

$$\rho_1 = 1 - p_1(0) = 0,9333;\ \rho_2 = 1 - p_2(0) = 0,4667.$$

The arrival rates (throughput in the closed network) are then derived as [18]:

$$\lambda = \lambda_1 = \lambda_2 = \frac{rho_1}{\mu_1} = \frac{rho_2}{\mu_2} = 0,1867 \left[\frac{SRq}{s}\right].$$

The mean number of service requests n_i are [18]:

$$n_1 = \sum_{k=1}^{3} k * p_1(k) = 2,2667 \left[SRq_1\right];\ n_2 = \sum_{k=1}^{3} k * p_2(k) = 0,7333 \left[SRq_2\right].$$

The mean response times R_i are [18]:

$$R_1 = \frac{n_1}{\lambda_1} = 12,1429[s];\ R_2 = \frac{n_2}{\lambda_2} = 3,9286[s].$$

FMC-QE Model

The corresponding FMC-QE model is shown in figure 4.3 - 4.5. The service request structure is defined in figure 4.3. The actions of the two connected servers are now defined as *Sub-Request 1* and *Sub-Request 2*, which are parts of an overall *Request*. Furthermore, in this model there is also a request generation associated to the overall request.

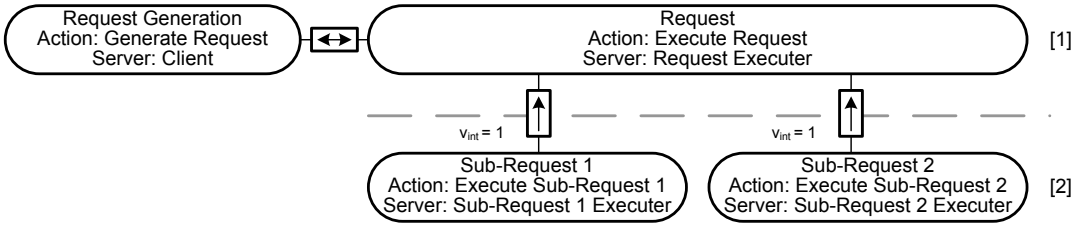


Figure 4.3: Closed Tandem Network - Service Request Structure

The server structure of this model is shown in figure 4.4. The two servers *Server 1* and *Server 2* are connected to the two actions, executed by the logical servers *Sub-Request 1 Executer* and *Sub-Request 2 Executer*.

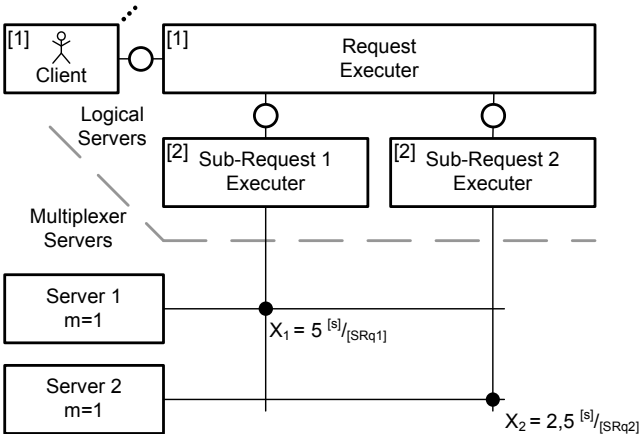


Figure 4.4: Closed Tandem Network - Server Structure

Finally, the dynamic behavior and the control flow are described in figure 4.5. Beside the definition of two hierarchical layers (one for *Execute request* and one for the two *Sub-Requests*) the main difference between this model and the original model in figure 4.1 is the introduction of a request generation, as usual in FMC-QE. In order to extend FMC-QE to closed networks, the think time of this external server (*Client/Request Generator*) has to be reduced to zero ("short-circuit") and furthermore, the overall number of service requests in the system has to be adjusted to a constant natural number.

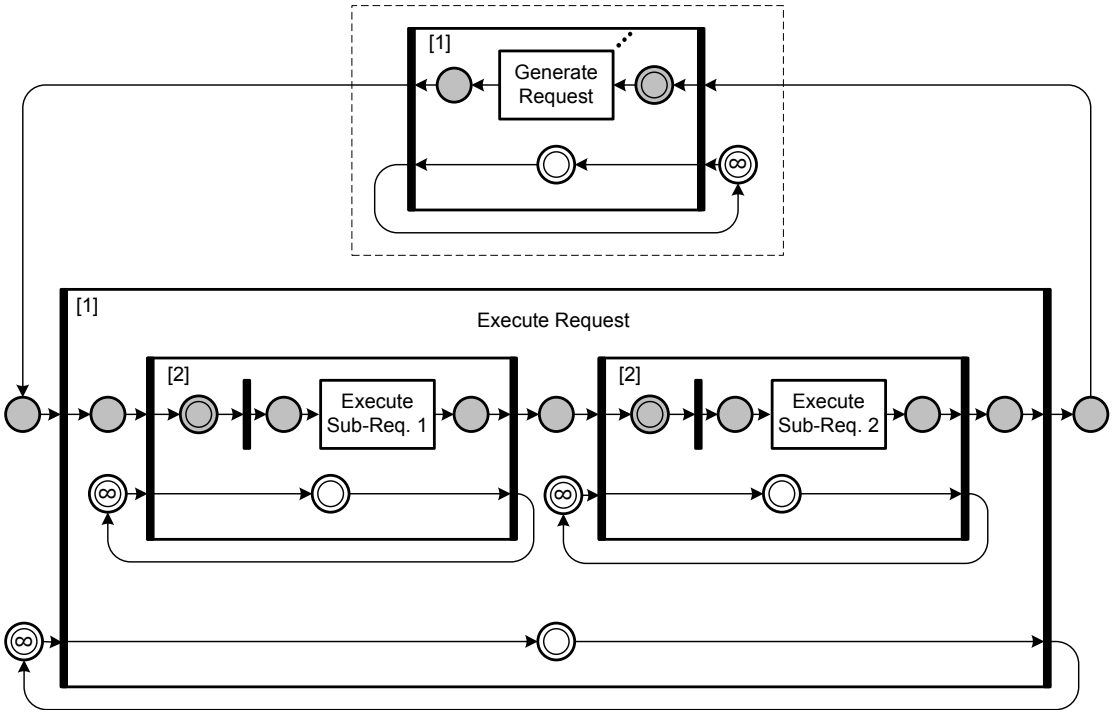


Figure 4.5: Closed Tandem Network - Dynamic Behavior

In the following some approaches for the extension of FMC-QE to closed networks are described with the help of this example.

M/M/1 Approximation Method

In a first approximation approach for a closed network the performance values are derived in a standard FMC-QE Tableau with M/M/1 servers (formulas are also in table A.3):

$$n_i, q = \frac{\rho^2}{1-\rho} \qquad n_i, s \quad = \rho \qquad n_i = \frac{\rho}{1-\rho} \tag{4.1}$$

and an external time $X_{ext} = 0$. In this model the network is handled like an open network, including an external service request generation (M/M/1 servers), whereas the service request generation and the handling of the service response (outside world) is handled with a service time of zero ($X_{ext} = 0$).

In order to achieve an external service time of zero, the arrival rate λ is adjusted in an iterative approximation approach until $X_{ext} = 0$ for $n_{ges} = 3$. In figure 4.6 the functional dependency of the external service time X_{ext} from the desired bottleneck utilization f, where $\lambda = f * \lambda_{bott}$ and $0 < f < 1$, is shown, which is then later adjusted though the iterative approach (values of first iterative steps are plotted). For a small f the external service time is large, because the arrival rate is small and therefore only a few service requests are in the system and the external service time X_{ext} for the rest of the service requests ($n_{ext} = n_{ges} - n_{sys}$) has to be large. If the desired bottleneck utilization f is too large and therefore the arrival rate λ is also too large, there are too many service requests in the system ($n_{sys} > n_{ges}$) and therefore the external service time X_{ext} converges against $-\infty$ (the external world is a "time machine").

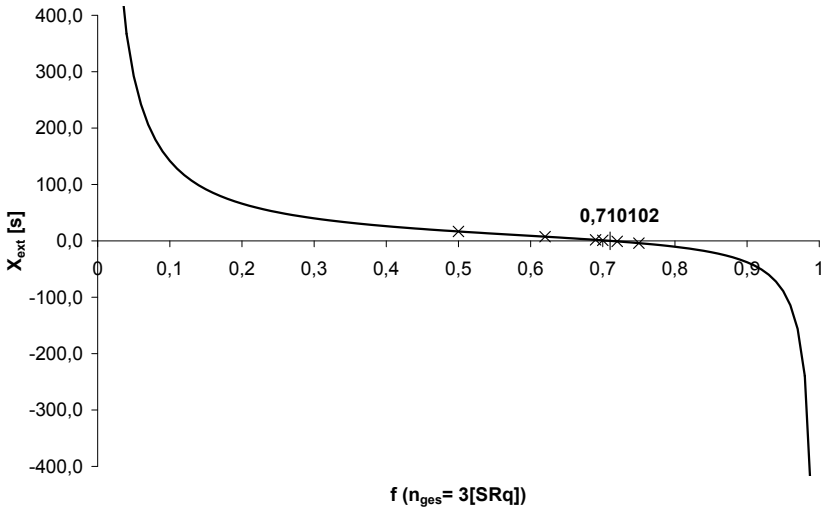


Figure 4.6: Closed Tandem Network - M/M/1 - Chart: Adjustment External Service Time - f

The corresponding FMC-QE Tableau is shown in table 4.1. The desired bottleneck utilization f with the value $0,710102$ was calculated through an iterative approximation approach in which this value was adjusted until $n_2^{[1]} = 3$ (overall number of requests in the system) as predefined in the parameter n_{ges} of the original model.

Table 4.1: Closed Tandem Network - M/M/1 Tableau (see Appendix - Table B.3)

Experimental Parameters	
$n_{ges}^{[1]}$	3
$\lambda_{bott}^{[1]}$	0,2000
f	0,710102
$\lambda^{[1]}$	0,1420

Service Request Section								Server Section						Dynamic Evaluation Section									
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
3	1	Sub-Request 2	1,00	1,00	1,00	1,00	0,142	Executer 2	1	1	1	2	2,500	1,000	0,400	0,355	0,195	1,376	0,355	2,500	0,551	3,876
3	2	Sub-Request 1	1,00	1,00	1,00	1,00	0,142	Executer 1	1	1	1	1	5,000	1,000	0,200	0,710	1,739	12,247	0,710	5,000	2,449	17,247
1	3	Request	1,00	1,00	1,00	1,00	0,142	Executer	1	1	1				0,200		1,935	13,624	1,065	7,500	3,000	21,124
1	4	Req. Generation	1,00	1,00	1,00	1,00	0,142	Client	1	1	1		0,000		####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Server 1	1	2,500
2	Server 2	1	5,000

The approximation errors (relative error $\delta x_i = \frac{\Delta x_i}{x} * 100$) [22] in the prediction of the overall arrival rate:

$$\lambda_{GlobalBalanceCalculation} = 0,1867;\ \lambda_{M/M/1\ Approx.} = 0,1420$$
$$\delta_\lambda = \frac{|\lambda_{GlobalBalanceCalculation} - \lambda_{M/M/1\ Approx.}|}{|\lambda_{GlobalBalanceCalculation}|} = 23,9\%$$

and the response times of the servers (especially R_1):

$$R_{1,GlobalBalanceCalculation} = 12,1429;\ R_{1,M/M/1\ Approx.} = 17,2474;$$
$$\delta_{R_1} = \frac{|R_{1,GlobalBalanceCalculation} - R_{1,M/M/1\ Approx.}|}{|R_{1,GlobalBalanceCalculation}|} = 29,6\%;$$
$$R_{2,GlobalBalanceCalculation} = 3,9286;\ R_{2,M/M/1\ Approx.} = 3,8763;$$
$$\delta_{R_2} = \frac{|R_{2,GlobalBalanceCalculation} - R_{2,M/M/1\ Approx.}|}{|R_{2,GlobalBalanceCalculation}|} = 1,3\%$$

are very high, because in this solution the number of service requests in the system is only a mean number derived by calculations for open networks and not a constant natural number as usually for closed networks.

M/M/1/K Method

In a second calculation approach the two servers are represented by M/M/1/K servers with a capacity of $K = 3$ and so the server formulas are (formulas are also in table A.6):

$$\begin{aligned}
\rho_i &= \frac{\lambda i}{\mu_i} \\
n_{i,q} &= \begin{cases} \frac{\rho_i}{1-\rho_i} - \frac{\rho_i\left(K\rho_i^K+1\right)}{1-\rho_i^{K+1}} & \rho_i \neq 1 \\ \frac{K(K-1)}{2(K+1)} & \rho_i = 1 \end{cases} \\
n_{i,s} &= \begin{cases} 1 - \frac{1-\rho_i}{1-\rho_i^{K+1}} & \rho_i \neq 1 \\ 1 - \frac{1}{K+1} & \rho_i = 1 \end{cases} \\
n_i &= \begin{cases} \frac{\rho_i}{1-\rho_i} - \frac{K+1}{1-\rho_i^{K+1}}\rho_i^{K+1} & \rho_i \neq 1 \\ \frac{K}{2} & \rho_i = 1 \end{cases}
\end{aligned} \tag{4.2}$$

The problem is that in this method the arrival rates are an input parameter of the model and not a result, as usual for closed models. Furthermore, in M/M/1/K models there is a distinction between the arrival rates $\lambda_i^{[bb]}$ and the effective arrival rates $\lambda_{i,eff}^{[bb]}$ which is dependent on the utilization of the different servers. So in order to adjust the right effective arrival rates, not only the overall arrival rate but also the different traffic flow coefficients had to be adjusted. For this proof-of-concept the known effective arrival rates from the original example, calculated in the beginning of this section, are used to solve this problem. So in order to receive the correct results, in the Tableau shown in table 4.2, the different effective arrival rates $\lambda_{i,eff}^{[bb]}$ are adjusted by the overall arrival rate $\lambda^{[1]}$ and the traffic flow coefficients $v_i^{[bb]}$ in order to fit with the value $0,1867$ of the original example [18], calculated via the global balance equation.

Table 4.2: Closed Tandem Network - M/M/1/K Tableau (see Appendix - Table B.4)

Experimental Parameters	
$n_{ges}^{[1]}$	3
$\lambda^{[1]}$	0,4000

Service Request Section									Server Section						Dynamic Evaluation Section									
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$\lambda_{i,eff}^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
3	1	Sub-Request 2	1,00	1,00	0,50	0,5	0,200	0,187	Executer 2	1	1	1	2	2,500	1,000	0,400	0,500	0,267	1,429	0,467	2,500	0,733	3,929
3	2	Sub-Request 1	1,00	1,00	1,00	1	0,400	0,187	Executer 1	1	1	1	1	5,000	1,000	0,200	2,000	1,333	7,143	0,933	5,000	2,267	12,143
2	3	Request	1,00	1,00	1,00	1	0,400	0,187	Executer	1	1	1				0,200		1,600	8,571	1,400	7,500	3,000	16,071
1	4	Req. Generation	1,00	1,00	1,00	1	0,400		Client	1	1	1		0,000		####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Server 1	1	5,000
2	Server 2	1	1,250

In the special case of the tandem network the performance values are exactly the same for the M/M/1/K model and the calculation via the global balance equations, but this in not true for every closed network (in the second example of this section the values are not the same). Furthermore, for the calculation of this model the effective arrival rates had to be known in advance in order to adjust the traffic flow coefficients for this model or a more complex equation system or multi dimensional iteration approach had to be solved in order to retrieve the results. Because in this closed tandem model the effective arrival rates:

$$\lambda_{i,eff} = \begin{cases} \lambda \left(1 - \frac{1-\rho_i}{\frac{1}{\rho_i^K} - \rho_i}\right) & \rho_i \neq 1 \\ \lambda \left(1 - \frac{1}{K+1}\right) & \rho_i = 1 \end{cases} \tag{4.3}$$

had to be the same for every server in the tandem network, so the arrival rates λ_i had to be adjusted through the traffic flow coefficient $\lambda_i = v_i * \lambda$ and also the overall number of service requests in the system n_{ges}, with:

$$n_i = \begin{cases} \frac{\rho_i}{1-\rho_i} - \frac{K+1}{1-\rho_i^{K+1}}\rho_i^{K+1} & \rho_i \neq 1 \\ \frac{K}{2} & \rho_i = 1 \end{cases} \tag{4.4}$$

and

$$n_{ges} = \sum_{i=1}^{2} n_i \tag{4.5}$$

had to be 3[SRq] ($n_{ges} = 3[SRq]$). For this proof-of-concept tandem network the M/M/1/K model was calculable, but for larger networks this model and calculation is not feasible.

Summation Method

While in FMC-QE the arrival rate is an input parameter and the number of service requests in the system is a result, in closed systems this normally is the opposite, as the number of service requests in the system is an input parameter and the arrival rate or throughput is a result. The summation method [17] is an exception and fits to this model of the arrival rate as input and the number of service requests in the system as a result and so it solves problems of the M/M/1/K model.

In the summation method the mean number of service requests in each node is a function of the throughput of the node [18]:

$$n_i = f_i(\lambda_i). \tag{4.6}$$

[17] propose the following formulas for $f_i(\lambda_i)$ [18]:

$$f_i(\lambda_i) = \begin{cases} \frac{\rho_i}{1-\frac{K-1}{K}\rho_i}, & Type-1,2,4\ (m_i = 1), \\ m_i\rho_i + \frac{\rho_i}{1-\frac{K-m_i-1}{K-m_i}\rho_i} p_i(m_i), & Type-1\ (m_i > 1), \\ \frac{\lambda_i}{\mu_i}, & Type-3. \end{cases} \tag{4.7}$$

with the utilization [18]:

$$\rho_i = \frac{\lambda_i}{m_i\mu_i} \tag{4.8}$$

and the waiting probabilities (for Type-3 Server, $m_i > 1$) [18]:

$$p_i(m_i) = \left(\frac{m_i!\,(1-\rho_i)\sum_{k=0}^{m_i-1} \frac{(m_i\rho_i)^k}{k!}}{(m_i\rho_i)^{m_i}} + 1 \right)^{-1} \tag{4.9}$$

The function $f_i(\lambda_i)$ is correct for Type-3 servers (infinite servers) and an approximation for Type-1,2 and 4 [18].

If f_i is given for every basic server station (number of basic server stations = I) in the network, the overall number of service requests in the system is given by [18]:

$$\sum_{i=1}^{I} n_i = \sum_{i=1}^{I} f_i(\lambda_i) = K \tag{4.10}$$

and including the traffic flow coefficients v_i, the overall number of service requests in the system is a function of the arrival rate [18]:

$$\sum_{i=1}^{I} f_i(v_i\lambda) = g(\lambda) = K. \tag{4.11}$$

For the usage of the summation method in the FMC-QE calculations the basic server formulas are substituted by the summation formulas (4.7), and then the solution is derived in an iterative calculation, modified from [18]:

While the desired bottleneck utilization f ($\lambda = f * \lambda_{bott}$) is 0 for the lower bound of the arrival rate and 1 for the upper bound (arrival rate λ = bottleneck throughput λ_{bott}), the bounds are $f_l = 0$ and $f_u = 1$ in the first step (desired bottleneck utilization $f \neq f_i$).

Then in the second step $f = \frac{f_l + f_u}{2}$ and the Tableau is solved. If the overall number of service requests in the system is $K \pm \epsilon$, then the solution is found, else the bounds are set to

- $f'_u = \frac{f_l + f_u}{2}$ if the overall number of service requests in the system > K and
- $f'_l = \frac{f_l + f_u}{2}$ if the overall number of service requests in the system < K,

and then the next iteration is started with the second step.

The corresponding chart, which shows the dependency of the desired bottleneck utilization (f) and the number of service requests in the system (n_{system}) for the example, is depicted in figure 4.7 (including the first iterative steps). In this configuration the desired bottleneck utilization is adjusted to $f = 0,9144$ to achieve an overall number of service requests in the system of 3 service requests ($n_{system} = 3[SRq]$).

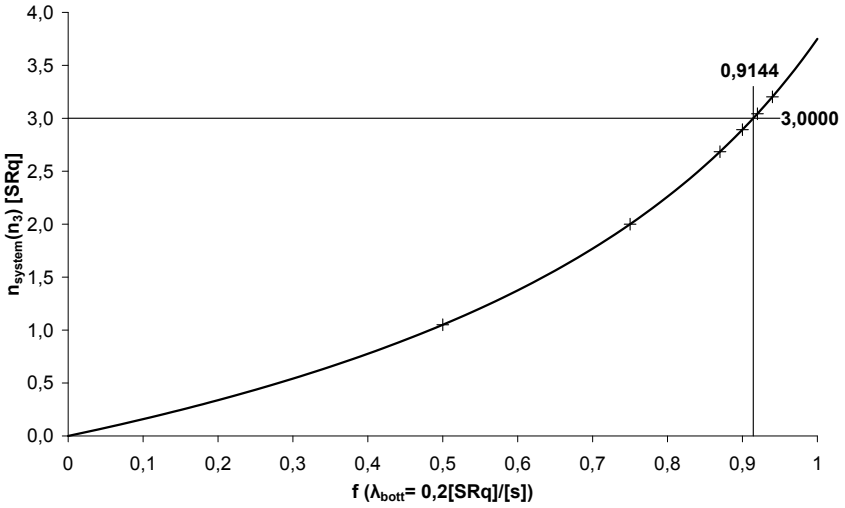


Figure 4.7: Closed Tandem Network - Summation Method Chart: n_{System} - f

Table 4.3 shows the corresponding Tableau of the closed tandem network example.

Table 4.3: Closed Tandem Network - Summation Method Tableau (see Appendix - Table B.5)

Experimental Parameters	
$n_{ges}^{[1]}$	3
$\lambda_{bott}^{[1]}$	0,2000
f	0,9144
$\lambda^{[1]}$	0,1829

Service Request Section								Server Section						Dynamic Evaluation Section									
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,Int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,Int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
3	1	Sub-Request 2	1,00	1,00	1,00	1,00	0,183	Executer 2	1	1	1	2	2,500	1,000	0,400	0,457	0,200	1,096	0,457	2,500	0,658	3,596
3	2	Sub-Request 1	1,00	1,00	1,00	1,00	0,183	Executer 1	1	1	1	1	5,000	1,000	0,200	0,914	1,428	7,808	0,914	5,000	2,342	12,808
1	3	Request	1,00	1,00	1,00	1,00	0,183	Executer	1	1	1				0,200		1,628	8,904	1,372	7,500	3,000	16,404
1	4	Req. Generation	1,00	1,00	1,00	1,00	0,183	Client	1	1	1		0,000		####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Server 1	1	2,500
2	Server 2	1	5,000

The approximation error (relative error $\delta x_i = \frac{\Delta x_i}{x} * 100$) [22] in the prediction of the overall arrival rate and the response times of the servers are:

$$\lambda_{GlobalBalanceCalculation} = 0,1867;\ \lambda_{SUM\ Approx.} = 0,1829$$
$$\delta_\lambda = \frac{|\lambda_{GlobalBalanceCalculation} - \lambda_{SUM\ Approx.}|}{|\lambda_{GlobalBalanceCalculation}|} = 2,04\%$$
$$R_{1,GlobalBalanceCalculation} = 12,1429;\ R_{1,SUM\ Approx.} = 12,8078;$$
$$\delta_{R_1} = \frac{|R_{1,GlobalBalanceCalculation} - R_{1,SUM\ Approx.}|}{|R_{1,GlobalBalanceCalculation}|} = 5,48\%;$$
$$R_{2,GlobalBalanceCalculation} = 3,9286;\ R_{2,SUM\ Approx.} = 3,5961;$$
$$\delta_{R_2} = \frac{|R_{2,GlobalBalanceCalculation} - R_{2,SUM\ Approx.}|}{|R_{2,GlobalBalanceCalculation}|} = 8,46\%$$

Although there are two approximations in the new FMC-QE algorithm extension - one through the iteration and the second through the summation method itself [18], the approximation errors for this example are quite small. Furthermore, as discussed later in the summary of this section, the computational complexity of the summation method in FMC-QE is small for a large overall number of service requests (population) in comparison to the Mean Value Analysis.

4.1.3 Central Server Network

The second closed network example is a classical CPU - Disk(s) closed model (Central Server). In this part the solution following the Theorem of Gordon and Newell [64], already discussed in section 2.1.5, is compared to the FMC-QE summation method.

Original Model

The original example is the example 4.3-2 in [68]. It was described using a graph, shown in figure 4.8, and a table, shown in table 4.4.

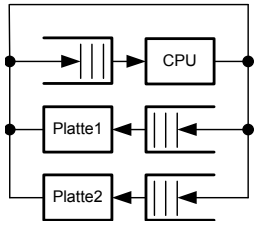


Figure 4.8: Central Server Model - Original Model [68]

Table 4.4: Central Server Model - Original Parameters [68]

Name	Index i	B_i	$p_{i,1}$	$p_{i,2}$	$p_{i,3}$	N = 3
CPU	1	0,50	0,10	0,40	0,50	
Platte1	2	0,40	1,00	0,00	0,00	
Platte2	3	0,25	1,00	0,00	0,00	

In order to have consistent Queueing Network models in this thesis, the model was reengineered to a more convenient model, shown in figure 4.9.

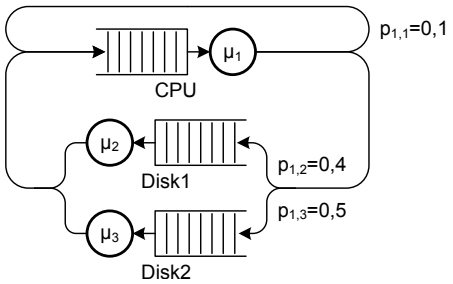


Figure 4.9: Central Server - Original Model Reengineered

This model has the visit ratios:

$$e_1 = 1,000;\ e_2 = 0,400;\ e_3 = 0,500.$$

In [68] the model was calculated using the Theorem of Gordon and Newell [64], further explained in section 2.1.5. For a system with three Service Requests inside ($n_{ges} = 3[SRq]$) the calculations lead to the following results [68] (recalculated manually and using WinPEPSY[1] [86]):

$$\begin{array}{llllll} \rho_1 & = 0,6939; & \rho_2 & = 0,3469; & \rho_3 & = 0,6939; \\ D_1 & = 0,3469; & D_2 & = 0,1388; & D_3 & = 0,1735; \\ n_{1,q} & = 0,5714; & n_{2,q} & = 0,1224; & n_{3,q} & = 0,5714; \\ n_{1,s} & = 0,6939; & n_{2,s} & = 0,3469; & n_{3,s} & = 0,6939; \\ n_1 & = 1,2653; & n_2 & = 0,4693; & n_3 & = 1,2653; \\ R_1 & = 3,6471; & R_2 & = 3,3824; & R_3 & = 7,2941; \\ R & = 8,6471 & D & = 0,3469 & & \end{array} \tag{4.12}$$

[1] WinPESPY, Website: http://www7.informatik.uni-erlangen.de/~prbazan/pepsy/, January 2010

FMC-QE Model

In the corresponding FMC-QE model, shown in figure 4.10, figure 4.11 and figure 4.12, there have also been done some transformations, as in the example in section 3.5. The different steps will not be shown here in detail, but these were a transformation to a Petri Net, the integration of the external load generation, a feed-forward - feed-backward transformation for the loop around the CPU, a hierarchy for the disk branch and another hierarchy for the whole service request. For the sake of simplicity and a smaller model the branch of if the data has to be written ($p_{write} = 0,9$) or not ($p_{out} = 0,1$ - request finished) has no additional hierarchy. This probability is integrated into the model and corresponding tableau without an additional hierarchical level.

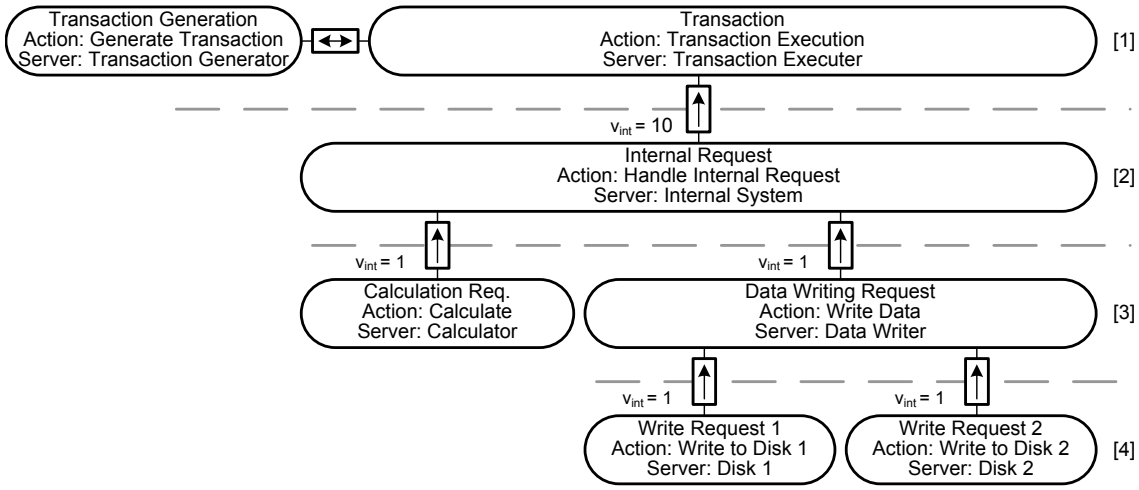


Figure 4.10: Central Server - FMC-QE Model (Service Request Structure)

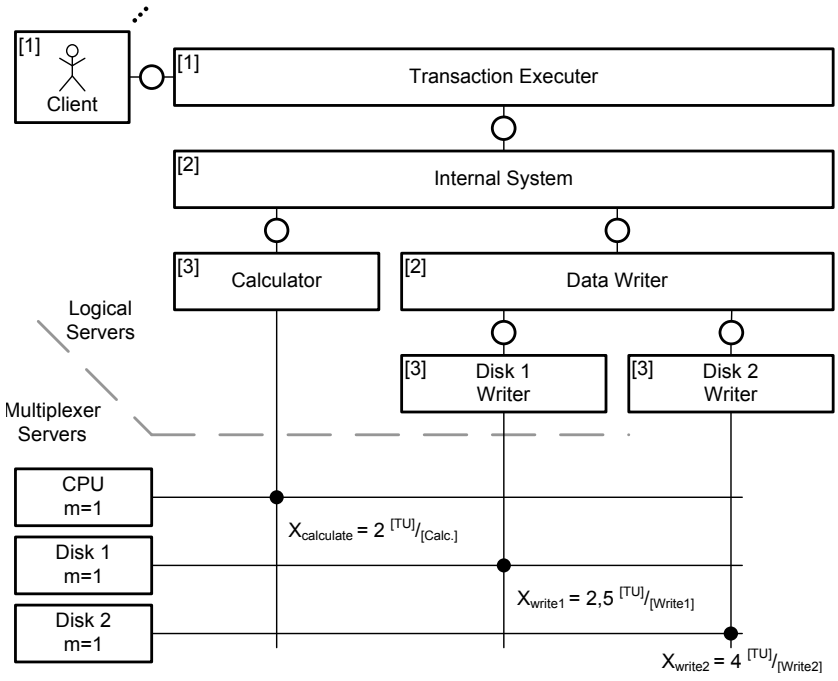


Figure 4.11: Central Server - FMC-QE Model (Server Structure)

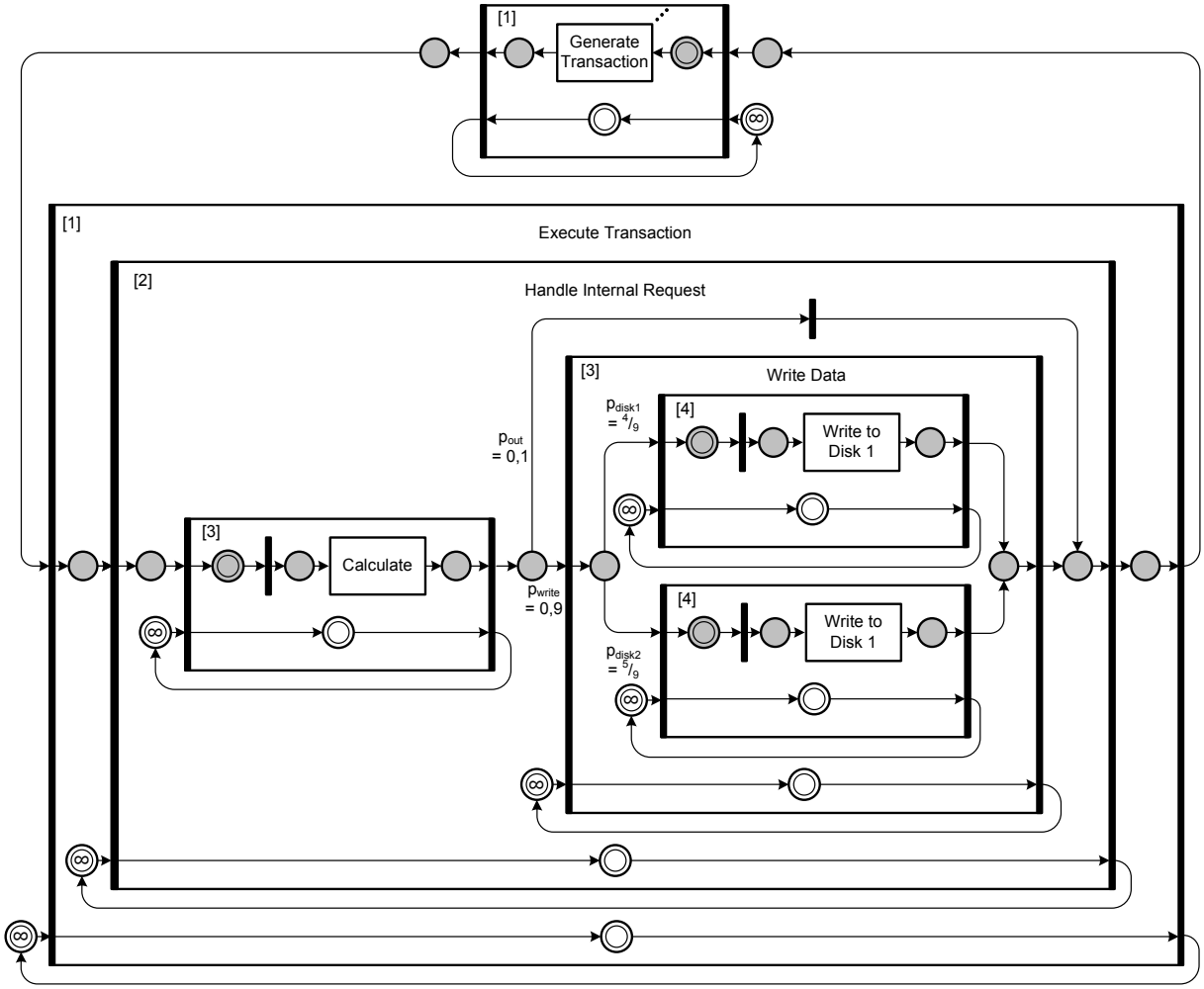


Figure 4.12: Central Server - FMC-QE Model (Dynamic Behavior)

The corresponding Tableau in table 4.5 shows the performance predictions of the FMC-QE model.

Table 4.5: Central Server - Tableau (see Appendix - Table B.7)

Experimental Parameters	
$n_{ges}^{[1]}$	3
$\lambda_{bott}^{[1]}$	0,0500
f	0,6895
$\lambda^{[1]}$	0,0345

Service Request Section								Server Section						Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
4	1	Write Request 2	0,56	9,00	1,00	5,00	0,172	Disk2 Writer	1	1	1	3	4,000	1,000	0,250	0,690	0,587	3,403	0,690	4,000	1,276	7,403
4	2	Write Request 1	0,44	9,00	1,00	4,00	0,138	Disk1 Writer	1	1	1	2	2,500	1,000	0,400	0,345	0,103	0,746	0,345	2,500	0,448	3,246
3	3	Data Writing Request	0,90	10,00	1,00	9,00	0,310	Data Writer	1	1	1				0,450		0,690	2,222	1,034	3,333	1,724	5,556
3	4	Calculation Request	1,00	10,00	1,00	10,00	0,345	Calculator	1	1	1	1	2,000	1,000	0,500	0,690	0,587	1,702	0,690	2,000	1,276	3,702
2	5	Internal Request	1,00	1,00	10,00	10,00	0,345	Internal Exec.	1	1	1				0,500		1,276	3,702	1,724	5,000	3,000	8,702
1	6	Transaction	1,00	1,00	1,00	1,00	0,034	Trans. Exec.	1	1	1				0,050		1,966	57,016	1,724	50,000	3,000	87,016
1	7	Transaction Generation	1,00	1,00	1,00	1,00	0,034	Client	1	1	1		0,000		####		0,000	0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	CPU	1	20,000
2	Disk1	1	20,000
3	Disk2	1	10,000

When comparing the results of FMC-QE in table 4.5 to the results of the Queueing Theory method in equation 4.12, the approximation errors (relative error $\delta x_i = \frac{\Delta x_i}{x} * 100$) [22] at the server level are very small (except the outlier ("Ausreißer") $n_{2,q}$ for which the error is still small):

$$\begin{array}{lllllll}
\delta_{\rho_1} & = 0,63\%; & \delta_{\rho_2} & = 0,61\%; & \delta_{\rho_3} & = 0,63\%; \\
\delta_{D_1} & = 0,61\%; & \delta_{D_2} & = 0,65\%; & \delta_{D_3} & = 0,63\%; \\
\delta_{n_{1,q}} & = 2,66\%; & \delta_{n_{2,q}} & = 15,93\%; & \delta_{n_{3,q}} & = 2,66\%; \\
\delta_{n_{1,s}} & = 0,63\%; & \delta_{n_{2,s}} & = 0,61\%; & \delta_{n_{3,s}} & = 0,63\%; \\
\delta_{n_1} & = 0,86\%; & \delta_{n_2} & = 4,60\%; & \delta_{n_3} & = 0,86\%; \\
\delta_{R_1} & = 1,49\%; & \delta_{R_2} & = 4,03\%; & \delta_{R_3} & = 1,49\%.
\end{array}$$

On the net level (D and R) the approximation error seems to be very high at the first moment ($D_{Queueing\ Theory} = 0,3469\frac{SRq}{s}$ vs. $D_{FMC-QE} = 0,0345\frac{SRq}{s}$ and $R_{Queueing\ Theory} = 8,6471s$ vs. $R_{FMC-QE} = 87,016s$), but in the FMC-QE model the repetition of the internal requests is transformed into a higher traffic flow within the *Internal Request*. The throughput at the highest level is 10 times lower than in the original model, which is just another interpretation of the model and not an error. - In the original model the CPU (including the repetitions of the internal requests) is the reference for the network throughput and the the new model the real net level throughput is the reference (loop above CPU). - Analogical, the overall response time has to be changed in order to consider these repetitions. With this "normalization" the approximation errors are:

$$\delta_D = 0,63\%, \delta_R = 0,63\%.$$

The problems with the normalization and the different understanding reveal another already discussed problem of the classical models: no distinction between the different service request types. Through the flat original model there is no distinction between the internal requests and the transactions (*real system requests*) and therefore these interpretation problems arise. In [68] in example 4.3-6 they also define this different net-throughput, which then leads to the normalized values of the original calculation of $D'_{Queueing\ Theory} = 0,0347\frac{SRq}{s}$ and $R'_{Queueing\ Theory} = 86,47s$ (resp. $85,71s$ - rounded values in the calculation of [68]).

After setting up the model and the corresponding Tableau, system and performance parameters could be changed in order to predict a broad range of scenarios. An exemplary parameter change is depicted in figure 4.13, where the population (n_{ges}) is changed and the derived throughput of the system ($\lambda^{[1]}$) is plotted. For comparative values the network was also modeled and evaluated using WinPEPSY[2] [86]. In the comparison of the predicted throughput of FMC-QE and the MVA [118] calculations of WinPEPSY the prediction is very precise, with a maximal error of $\delta = 0,75\%$ for $n_{ges} = 1..100$.

[2]WinPESPY, Website: http://www7.informatik.uni-erlangen.de/~prbazan/pepsy/, January 2010

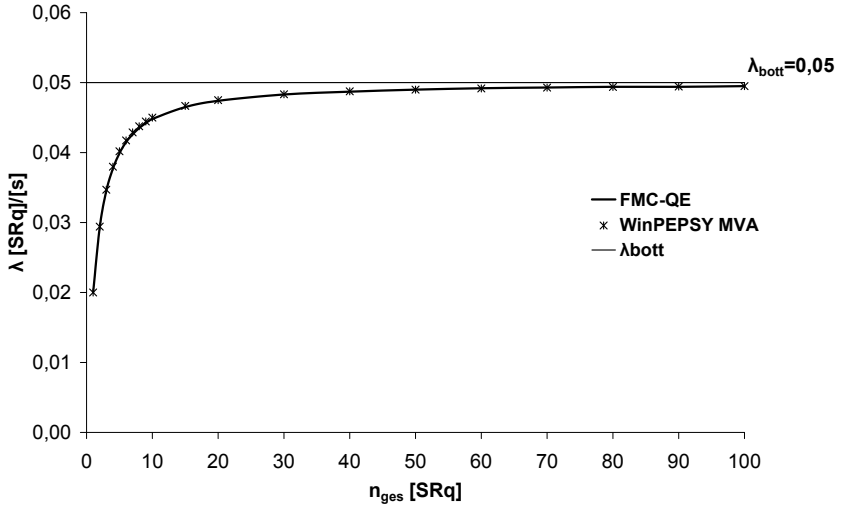


Figure 4.13: Central Server - Chart: Throughput - Population

4.1.4 Summary

Three methods have been presented: The first two approaches of the integration of closed Queueing Networks into FMC-QE, more precisely the approximation using M/M/m servers and an external service time of zero time units as well as the second model of M/M/m/K servers, were either to imprecise or inapplicable. But the third approach, the integration of the summation method [17, 18] into FMC-QE, explained and exemplified in this section, extends FMC-QE to the class of closed Queueing Networks, providing an iterative algorithm and good approximation for closed systems[3]. The approximative errors in the examples, evaluated using the summation method, are, except for the outlier $n_{2,q}$ in the second example, within the maximal error of 15% descibed in [17] and often even better than the average error of 5% for the number of service requests and the response time, as also described in [17]. The values for the throughput also deliver precise values, as descibed in [17].

Referring to [118], the operations count of the fast, simple and widely distributed closed Queueing Network analysis algorithm, the algorithm of the Mean Value Analysis (MVA) is $5MR$ per service center and recursive step, where M is the number of parallel servers at the service center and R is the number of chains which visit the service center (number of classes). Therefore, the overall complexity is $O(n * m)$, where n is the number of service centers and m is the overall number of service requests (n_{ges}). Comparing to this, the complexity of the FMC-QE summation method is independent from the overall number of service requests, while every value is calculated independently in contrast to the iteration over the number of service requests in the MVA. In the summation method in FMC-QE there is an iteration in order to adjust the desired bottleneck utilization respectively the throughput, here the number of iterations could be seen as a constant[4] and therefore the complexity is still $O(n)$, where n is the number of service centers (stations). Therefore, for a small overall number of service requests n_{ges} MVA is faster than FMC-QE, but in this region both algorithms are very efficient. For a large number of service requests FMC-QE outperforms MVA, while it is independent from the overall number of service requests in the system.

[3]The iterative algorithm converges to the overall number of service requests in the system, but there is still an approximative error for servers of Type-1,2 and 4 in the system [18].

[4]In the approximation of f, where $(0 \leq f < 1)$, the search space is bisected in every iteration, so f has a accuracy of $\epsilon \approx 10^{-4,5}$ after 15 steps.

4.2 Handling of Multiclass Scenarios

If there are different kinds of service requests and it is not reasonable to combine the different arrivals to a single class of service requests, multiclass models are the way of modeling such systems. In FMC-QE the multiclass scenario is an extension of the multiplex case. In multiclass models the logical structures differ from each other, so there is a different model for every class. The different logical structures meet at the server level, were the the different logical servers of the different classes are handled by overlapping multiplexers. The model is then partitioned by the multiplex coefficients and calculated separately. The handling of multiclass scenarios is a pre-step for the handling of semaphores, explained in the next section, where there are more inter relations between the different sub-nets.

The handling of multiclass scenarios is explained on an example of two classes *A* and *B* as follows: In figure 4.14 the service request structures and in figure 4.15 dynamic behavior of service request class *A* are shown. In class *A* a service request *A* is decomposed into a serial execution for two service requests *A.1* and *A.2*.

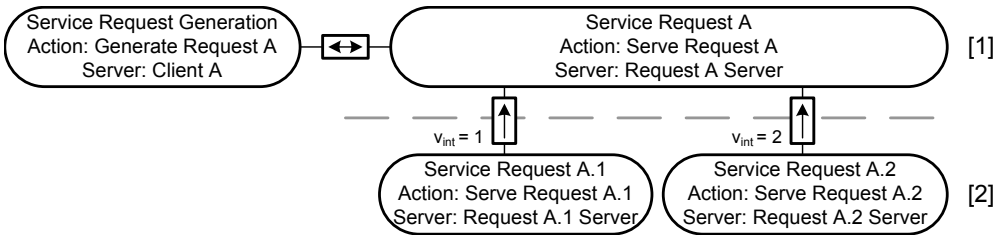


Figure 4.14: Multiclass Example - Class A - Service Request Structures

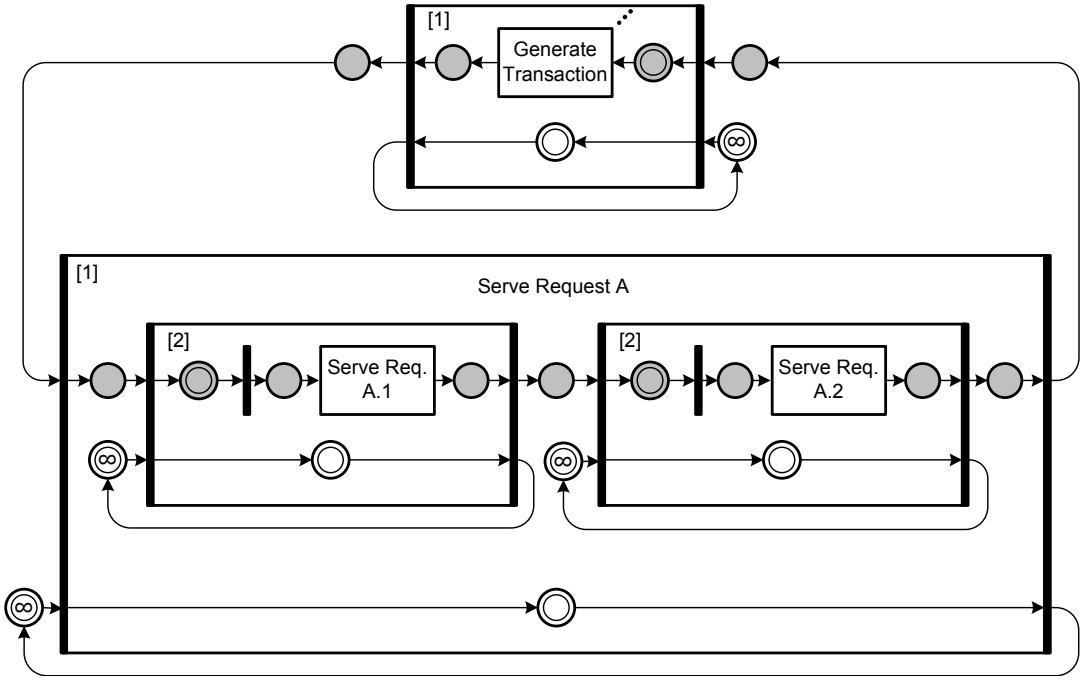


Figure 4.15: Multiclass Example - Class A - Dynamic Behavior

In the other class *B* (described in figure 4.16 and 4.17), there is also a decomposition into two service requests *B.1* and *B.2*. The service request *B.2* is furthermore decomposed into two parallel executed service requests *B.2.1* and *B.2.2*.

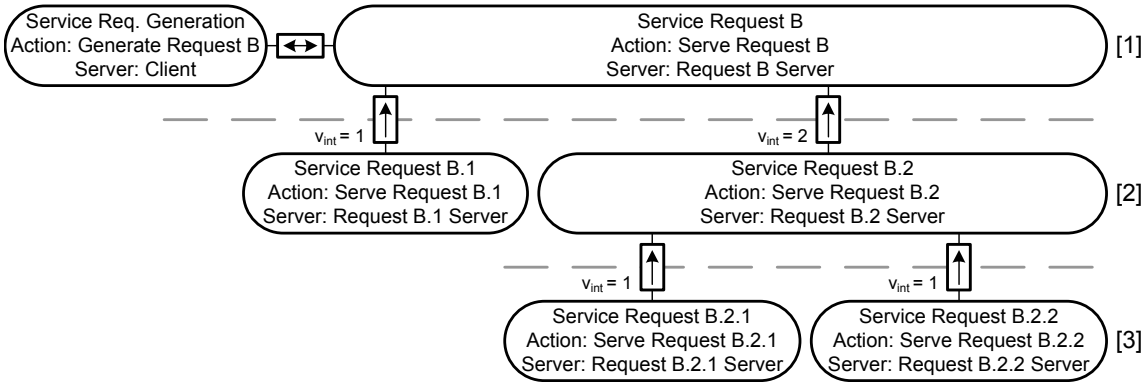


Figure 4.16: Multiclass Example - Class B - Service Request Structures

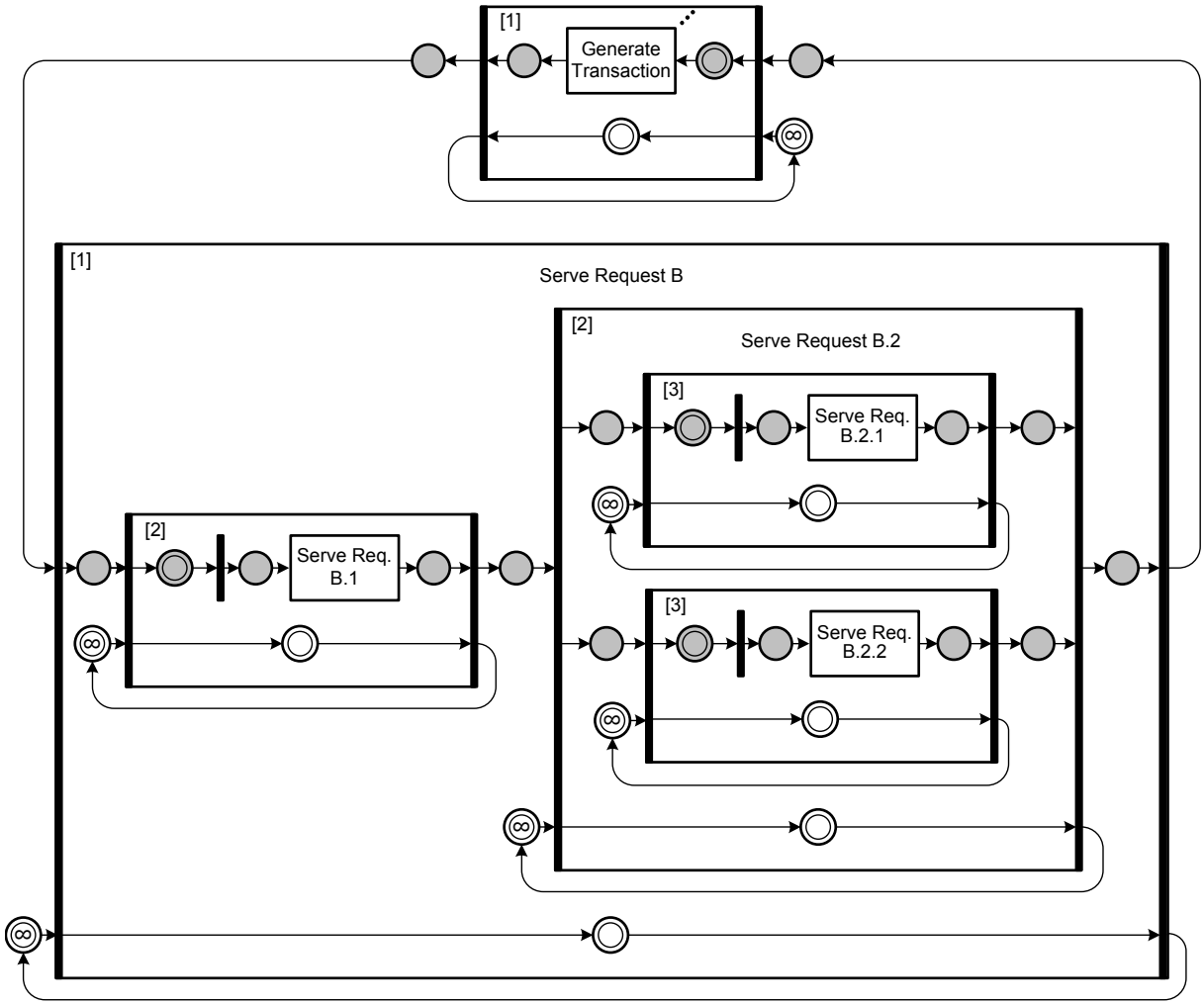


Figure 4.17: Multiclass Example - Class B - Dynamic Behavior

In addition to the service request structures and the dynamic behavior of the two classes *A* and *B*, the static structures are defined in figure 4.18. There is a multiplex of server *X* for the logical servers *Request A.1 Server* and *Request B.1 Server* and a multiplex of server *Y* for the logical servers *Request A.1 Server* and *Request B.2.1 Server*. The server *Z* is a dedicated server for the service request *B.2.2* in class *B*.

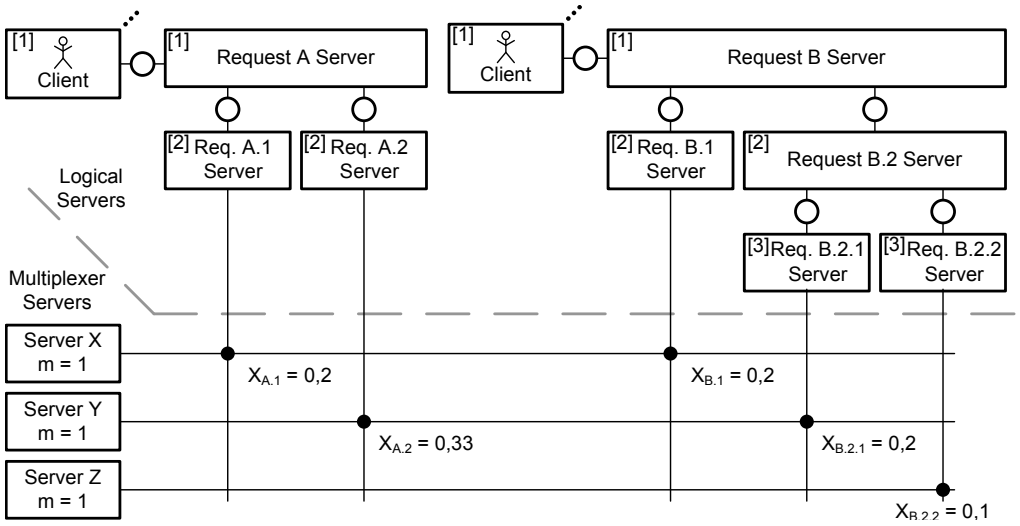


Figure 4.18: Multiclass Example - Static Structures

The corresponding Tableau of of the multiclass example is shown in table 4.6. In comparison to a non multiclass model this table consists of more than one experimental parameter and service request sections.

Table 4.6: Multiclass Example - Tableau (see Appendix - Table B.9)

Experimental Parameters	
n_{ges}	30
λ_{bott}	0,9434
f	0,8000
λ_A	0,7547

Service Request Section									Server Section						Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
2	1	Service Request A.2	1,00	1,00	2,00	2,00	1,509	Req. A.2 Srv.	1	1	1	2	0,330	0,623	1,887	0,800	3,200	2,120	0,800	0,530	4,000	2,650
2	2	Service Request A.1	1,00	1,00	1,00	1,00	0,755	Req. A.1 Srv.	1	1	1	1	0,200	0,500	2,500	0,302	0,131	0,173	0,302	0,400	0,432	0,573
1	3	Service Request A	1,00	1,00	1,00	1,00	0,755	Req. A.2.2 Srv.	1	1	1				1,887		3,331	4,413	1,102	1,460	4,432	5,873
1	4	Generate Request A	1,00	1,00	1,00	1,00	0,755	Client A	1	1	1		33,877		0,030			0,000	25,568	33,877	25,568	33,877

Experimental Parameters	
n_{ges}	30
λ_{bott}	0,9434
f	0,5000
λ_B	0,4717

Service Request Section									Server Section						Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
3	5	Service Request B.2.2	1,00	2,00	2,00	4,00	1,887	Req. B.2.2 Srv.	1	1	1	3	0,100	1,000	10,000	0,189	0,044	0,023	0,189	0,100	0,233	0,123
3	6	Service Request B.2.1	1,00	2,00	1,00	2,00	0,943	Req. B.2.1 Srv.	1	1	1	2	0,200	0,377	1,887	0,500	0,500	0,530	0,500	0,530	1,000	1,060
2	7	Service Request B.2	1,00	1,00	2,00	2,00	0,943	Req. B.2 Srv.	1	1	1				1,887		0,544	0,577	0,689	0,730	1,233	1,307
2	8	Service Request B.1	1,00	1,00	1,00	1,00	0,472	Req. B.1 Srv.	1	1	1	1	0,200	0,500	2,500	0,189	0,044	0,093	0,189	0,400	0,233	0,493
1	9	Service Request B	1,00	1,00	1,00	1,00	0,472	Req. B Srv.	1	1	1				1,887		0,588	1,246	0,877	1,860	1,465	3,106
1	10	Generate Request B	1,00	1,00	1,00	1,00	0,472	Client B	1	1	1		60,494		0,017			0,000	28,535	60,494	28,535	60,494

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Server X	1	0,400
2	Server Y	1	1,060
3	Server Z	1	0,400

Through the change of parameters in the Tableau and the corresponding plotting of the results interesting observations could be illustrated. Figure 4.19 shows the dependency of the response times R_A and R_B on the arrival rate of one class (λ_A).

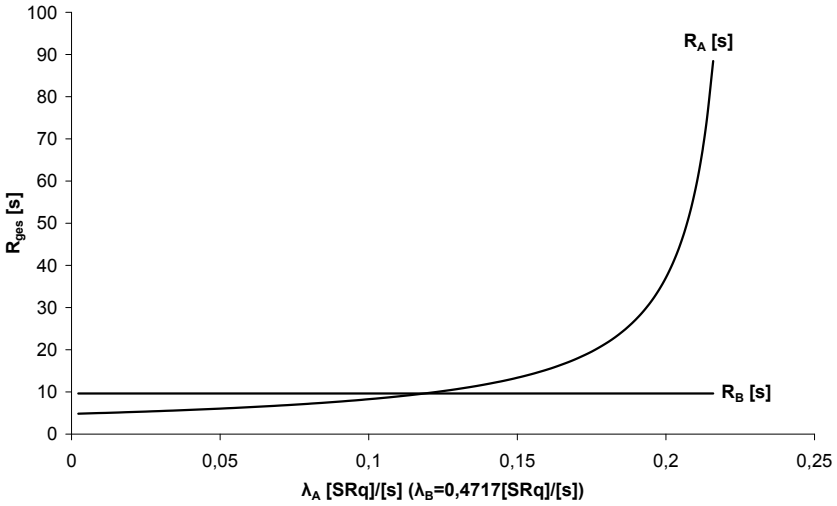


Figure 4.19: Multiclass Example - Chart: Response Times A, B - Arrival Rate A

In this figure it can be seen that the change of the arrival rate in one class has no influence on the response time of the other class. This is the case because the multiplex coefficient is precomputed independent from the actual arrival rate, as described in section 3.4.5:

$$m_{i,mpx}^{[bb]} = \begin{cases} 1 & if \ \sum_{\forall \, SRq_i \, of \, Mpx_j} m_i^{[bb]} \leq m_j \\ \frac{X_i^{[bb]} * v_i^{[bb]}}{X_j} * \frac{m_j}{m_i^{[bb]}} & if \ \sum_{\forall \, SRq_i \, of \, Mpx_j} m_i^{[bb]} > m_j \end{cases} \tag{4.13}$$

The dependency of the different classes is defined through this multiplex coefficient, and therefore, a change of the service time of one logical server in one class changes the response times of both classes, as shown in figure 4.20, where the service time of server *Request A.2 Server* ($X_{A.2}$) is variated and the corresponding overall response times of both classes (R_A and R_B) are recorded. If the service time $X_{A.2}$ grows, the overall response times of the corresponding class A also grows as well as the response time of the other classes, in which a logical server is handled by the same multiplexer as *Request A.2 Server* - here *Server Y* also multiplexes the *Request B.2.1 Server* of class B, and therefore, the response time of class B also grows for a larger service time $X_{A.2}$ for *Request A.2 Server* with constant arrival rates of $\lambda_A = 0,7547 \frac{[SRq]}{[s]}$ and $\lambda_B = 0,4717 \frac{[SRq]}{[s]}$.

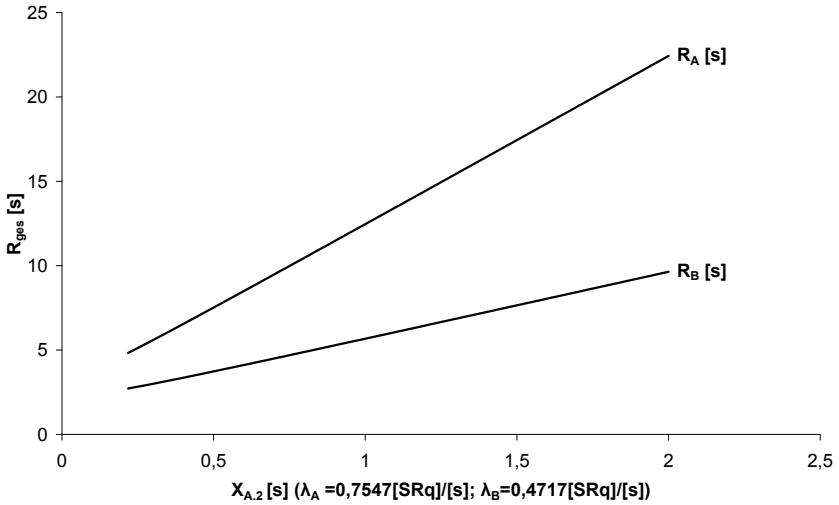


Figure 4.20: Multiclass Example - Chart: Response Times A, B - Service Time Req. A.2 Srv.

As a further extension of the methodology the different arrival rates of the classes could influence the multiplex coefficient and therefore the division of the subnets. The next section on the integration of semaphore synchronization scenarios into FMC-QE addresses this issue.

4.3 Semaphore Synchronization

The performance modeling of processes, which have to synchronize, or the modeling of semaphores is a classical Time Augmented Petri Net task. In this section an exemplary semaphore synchronization scenario is modeled and evaluated using FMC-QE. While this problem is a Non Product Form problem, an approximation for the calculation has been developed and presented in this section. For comparative reasons this system is also modeled using Generalized Stochastic Petri Nets (GSPN [102]).

In Time Augmented Petri Nets the mutual exclusive usage of resources could be synchronized via semaphores and drills down to the modeling and simulation or calculation of the discrete steady state distribution including the state space explosion problem of this kind of calculations. In FMC-QE the mutual exclusive usage of the resources and therefore the inter-server control flows are approximated by shared resources and a stochastic model without the synchronization and therefore without inter-server control flows. The example will show that although the real synchronization through the semaphore is neglected, this simplification leads to quite accurate results without the state space explosion problem.

While early ideas of modeling semaphore synchronization scenarios through shared resources and multiplexers were already published by Zorn in 2007 [143], the method in section gives a clearer definition of the interdependencies of the different sub-nets through a pre-calculation step extending the ideas of the last section about multiclass scenarios. Furthermore, the usage of the summation method, discussed in section 4.1, including the iterative calculation of closed networks, brings the integration of semaphore synchronization scenarios into FMC-QE to a round figure.

4.3.1 GSPN Model

In figure 4.21 a semaphore synchronization scenario is modeled. This model was already described in section 2.2.3 where this network was used as an exemplary Generalized Stochastic Petri Net.

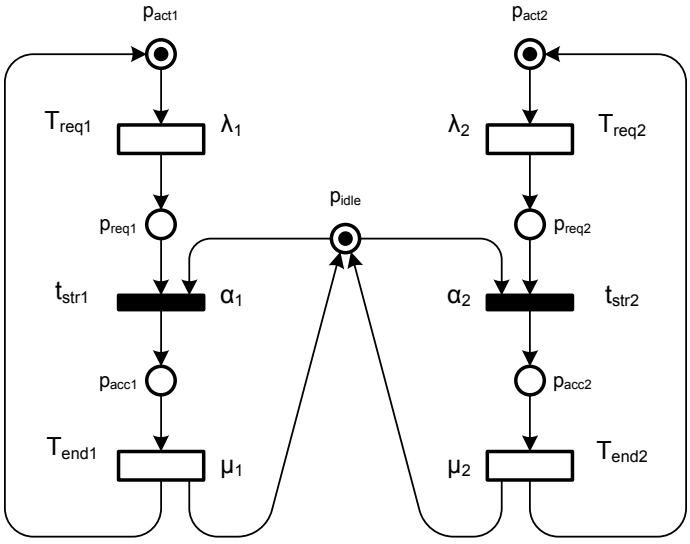


Figure 4.21: Semaphore Synchronization - GSPN Model [102]

The performance parameters of this semaphore synchronization model are defined in table 4.7.

Transition	Rate/Weight	Semantics
T_{req1}	$\lambda_1 = 1$	single server
T_{req2}	$\lambda_2 = 2$	single server
T_{str1}	$\alpha_1 = 1$	immediate
T_{str2}	$\alpha_1 = 1$	immediate
T_{end1}	$\mu_1 = 10$	single server
T_{end2}	$\mu_2 = 5$	single server

Table 4.7: Semaphore Synchronization - Parameters

The reachability graph of this Petri Net is illustrated in figure 4.22.

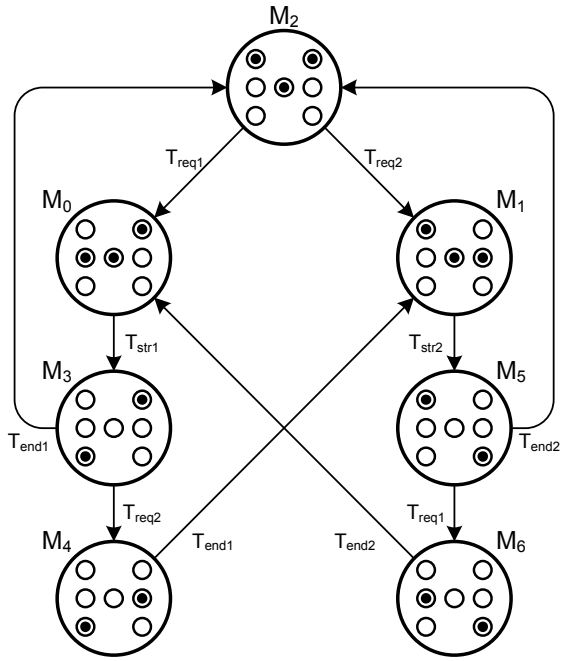


Figure 4.22: Semaphore Synchronization - Reachability Graph

The transition state probability matrix P is defined as:

$$P = \left[\begin{array}{cc|ccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \frac{\lambda_1}{\lambda_1+\lambda_2} & \frac{\lambda_2}{\lambda_1+\lambda_2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\mu_1}{\lambda_2+\mu_1} & 0 & \frac{\lambda_2}{\lambda_2+\mu_1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\mu_2}{\lambda_1+\mu_2} & 0 & 0 & 0 & \frac{\lambda_1}{\lambda_1+\mu_2} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

This results in the steady state distribution $\tilde{\pi}$ of:

$$\tilde{\pi}_0 = \frac{8}{63};\ \tilde{\pi}_1 = \frac{13}{63};\ \tilde{\pi}_2 = \frac{5}{18};\ \tilde{\pi}_3 = \frac{8}{63};\ \tilde{\pi}_4 = \frac{4}{189};\ \tilde{\pi}_5 = \frac{13}{63};\ \tilde{\pi}_6 = \frac{13}{378}.$$

Finally, the steady state distribution π is calculated as:

$$\pi_0 = 0;\ \pi_1 = 0;\ \pi_2 = \frac{175}{277};\ \pi_3 = \frac{20}{277};\ \pi_4 = \frac{4}{277};\ \pi_5 = \frac{65}{277};\ \pi_6 = \frac{13}{277}.$$

The throughput of the different timed transition T_k is calculated as $D_k = \omega_k \sum_{T_k \in E(M_i)} \pi_i$ and therefore:

$$D_{req1} = \lambda_1 (\pi_2 + \pi_5) = \frac{240}{277};\ D_{req2} = \lambda_2 (\pi_2 + \pi_3) = \frac{390}{277};$$
$$D_{end1} = \mu_1 (\pi_3 + \pi_4) = \frac{240}{277};\ D_{end2} = \mu_2 (\pi_5 + \pi_6) = \frac{390}{277}.$$

The throughput of the left subnet equals:

$$D_{req1} = D_{end1} = \frac{240}{277} \approx 0,8664$$

and the throughput right subnet is:

$$D_{req2} = D_{end2} = \frac{390}{277} \approx 1,4079.$$

4.3.2 FMC-QE Model

In the corresponding FMC-QE model, illustrated in figure 4.23 and 4.24, the transitions T_{req1} and T_{req2} were interpreted as the request generation denoted as *Generate Request of Type 1* and *Generate Request of Type 2*. The transitions T_{end1} and T_{end2}, which are guarded by the semaphore, are denoted as *Execute Critical Action 1* and *Execute Critical Action 2*. This model includes inter-server control flows in order to implement the semaphore synchronization.

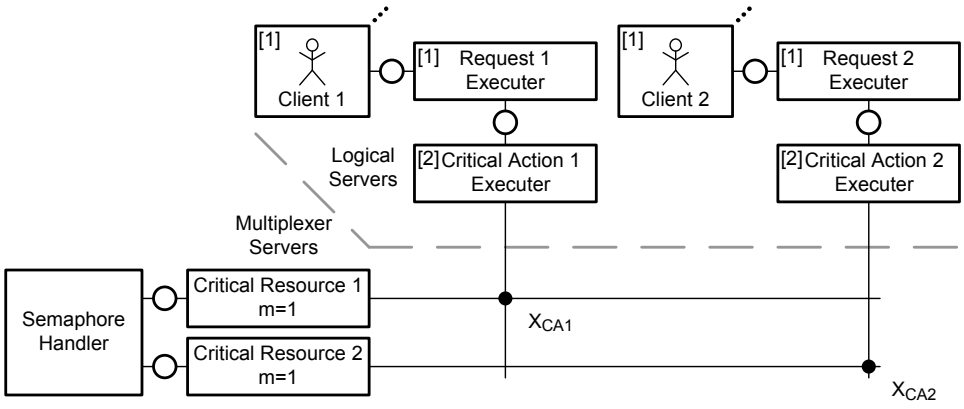


Figure 4.23: Semaphore Sync. with Inter-Server Control Flows (Static Structures) [147]

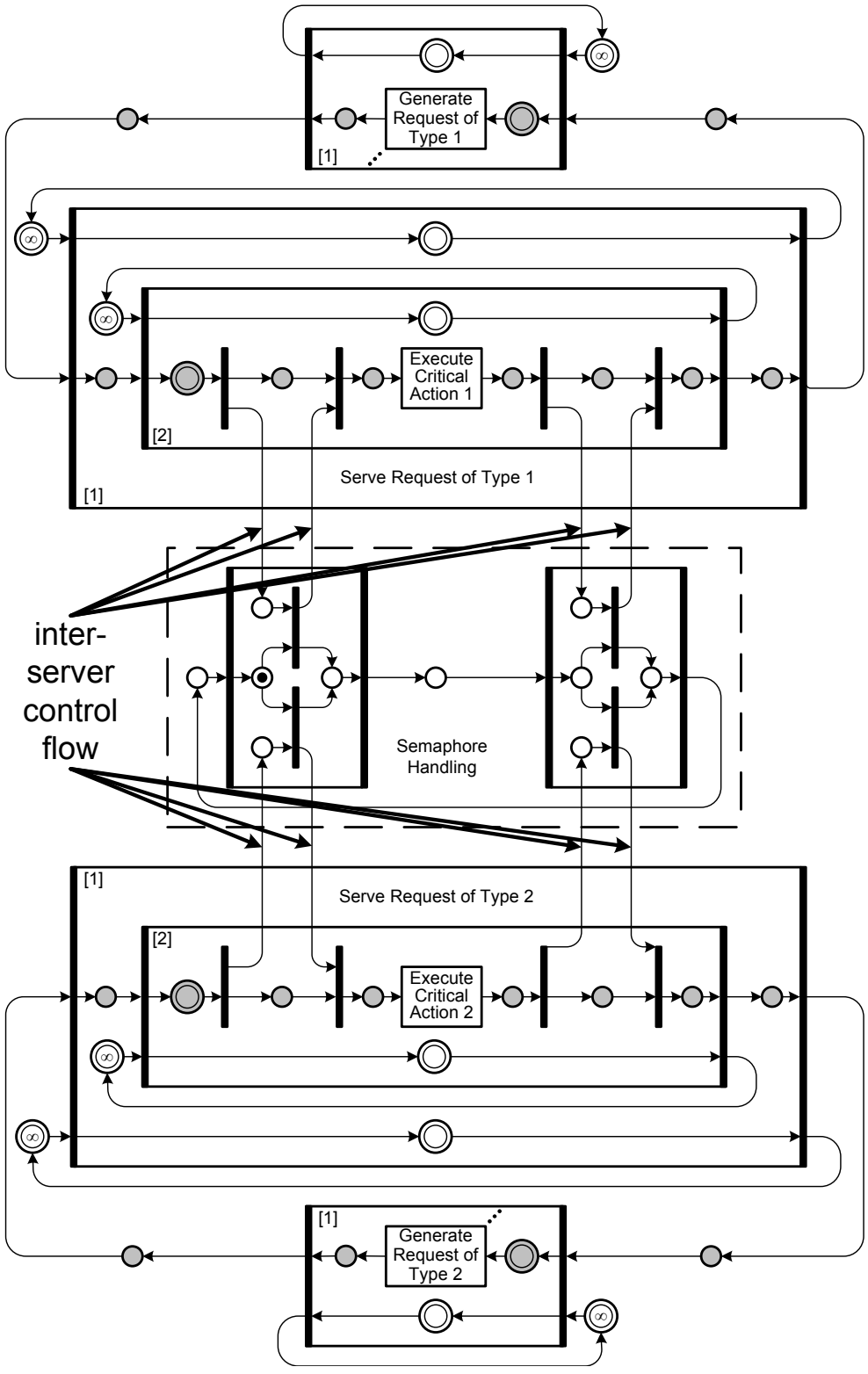


Figure 4.24: Semaphore Sync. with Inter-Server Control Flows (Dynamic Structures) [147]

In order to eliminate the inter-server control flows, in figure 4.25 and 4.26, the model has been transformed. Instead of synchronizing two independent critical resources, in this model the critical actions are handled by one critical resource modeled as a multiplexer. This model is an approximative model because the different service requests are now handled in parallel by the

multiplexer and are not really synchronized, like in the semaphore synchronization model in figure 4.24.

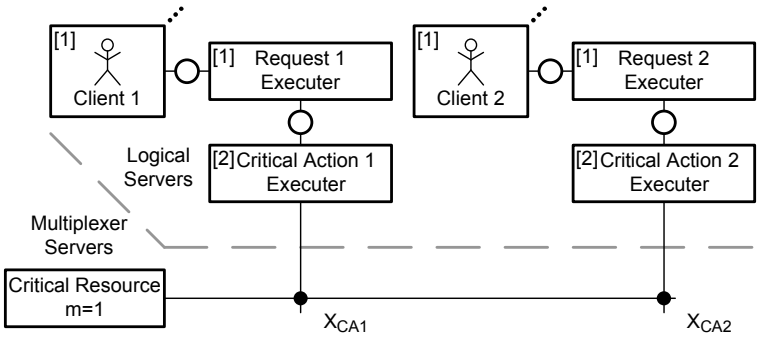


Figure 4.25: Semaphore Sync. without Inter-Server Control Flows (Static Structures)

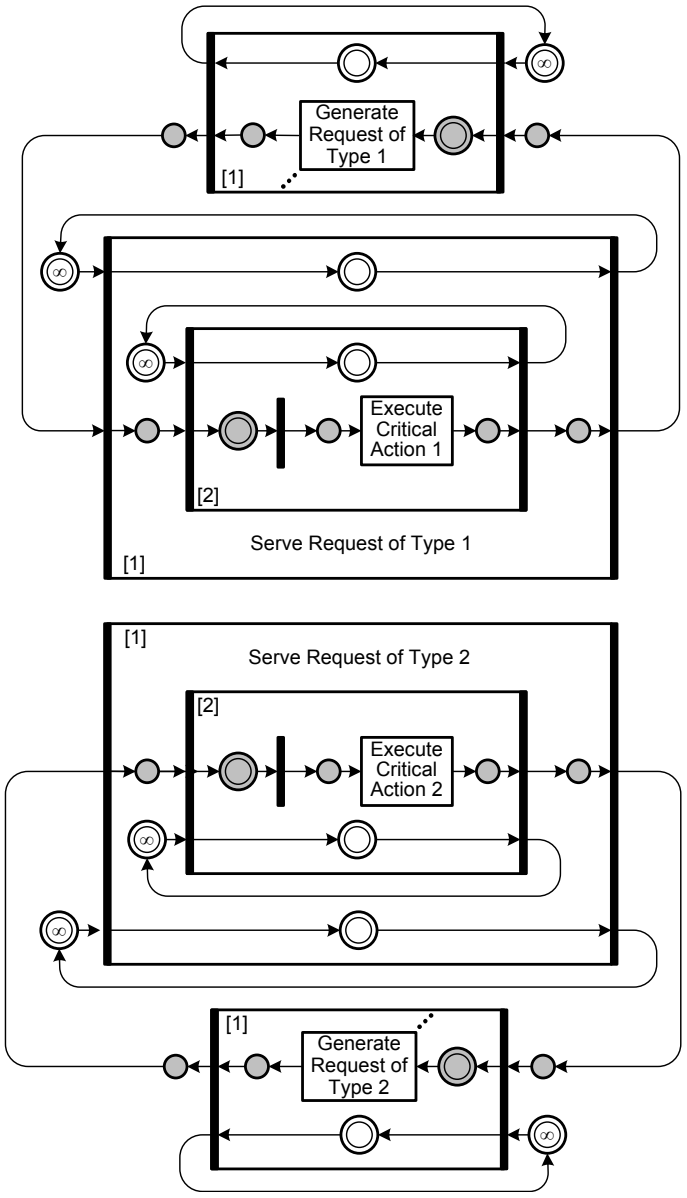


Figure 4.26: Semaphore Sync. without Inter-Server Control Flows (Dynamic Structures)

Table 4.8: Semaphore Synchronization - Tableau (see Appendix - Table B.8)

Experimental Parameters	
$n_{ges}^{[1]}$	1
$\lambda_{bott}^{[1]}$	1,0000
f	0,8800
$\lambda^{[1]}$	0,8800

Service Request Section								Server Section						Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
2	1	Critical Action 1	1,00	1,00	1,00	1,00	0,880	CA 1 Executer	1	1	1	1	0,100	0,733	7,333	0,120	0,000	0,000	0,120	0,136	0,120	0,136
1	2	Request 1	1,00	1,00	1,00	1,00	0,880	Executer 1	1	1	1				7,333		0,000	0,000	0,120	0,136	0,120	0,136
1	3	Req. Generation 1	1,00	1,00	1,00	1,00	0,880	Client 1	1	1	1		1,000	1,000	1,000	0,880	0,000	0,000	0,880	1,000	0,880	1,000

Experimental Parameters	
$n_{ges}^{[1]}$	1
$\lambda_{bott}^{[1]}$	2,0000
f	0,7073
$\lambda^{[1]}$	1,4146

Service Request Section								Server Section						Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
2	1	Critical Action 2	1,00	1,00	1,00	1,00	1,415	CA 2 Executer	1	1	1	1	0,200	0,967	4,833	0,293	0,000	0,000	0,293	0,207	0,293	0,207
1	2	Request 2	1,00	1,00	1,00	1,00	1,415	Executer 2	1	1	1				4,833		0,000	0,000	0,293	0,207	0,293	0,207
1	3	Req. Generation 2	1,00	1,00	1,00	1,00	1,415	Client 2	1	1	1		0,500	1,000	2,000	0,707	0,000	0,000	0,707	0,500	0,707	0,500

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Critical Resource	1	0,300

In the Tableau in table 4.8 the performance values of the multiplexer semaphore synchronization model are calculated. While the original model is an closed Petri Net, in this Tableau the summation method formulas, explained in section 4.1, have been used. As depicted in the model, the critical actions *Critical Action 1* and *Critical Action 2* are executed by one server *Critical Resource* as a multiplexer, like proposed in [147].

Here the multiplexer coefficient formula:

$$m_{i,mpx} = \begin{cases} 1 & if \ \sum_{\forall \, SRq_i \, of \, Mpx_j} m_i \leq m_j \\ \frac{X_i^{[bb]} * v_i^{[bb]}}{X_j} * \frac{m_j}{m_i} & if \ \sum_{\forall \, SRq_i \, of \, Mpx_j} m_i > m_j \end{cases} \tag{4.14}$$

with

$$X_j = \sum_{\forall \, SRq_i \, of \, Critical \, Resource_j} X_i^{[bb]} * v_i^{[bb]}, \tag{4.15}$$

from section 3.4 has been adapted for the semaphore case. When the two critical actions are handled by one critical resource, the critical resource has to be multiplexed if the respectively opposite critical action k is executed. The respectively opposite critical action k is executed with the probability ρ_k and not executed with the probability $(1 - \rho_k)$. So with a probability of ρ_k the multiplex coefficient of formula 4.14 is used and with a probability of $(1 - \rho_k)$ the semaphore multiplex coefficient is 1 (the critical action i is the only action on the critical resource). So the semaphore multiplex coefficient $m_{i,sem-mpx}$ is defined as:

$$m_{i,sem-mpx} = \begin{cases} 1 & if \sum\limits_{\forall \, SRq_i \, of \, Critical \, Resource_j} m_i \leq m_j \\ (1 - \rho_k) + \rho_k \left(\frac{X_i^{[bb]} * v_i^{[bb]}}{X_j} * \frac{m_j}{m_i} \right) & if \sum\limits_{\forall \, SRq_i \, of \, Critical \, Resource_j} m_i > m_j, \end{cases} \tag{4.16}$$

where ρ_k is the probability that the respectively opposite critical action k is active (utilization). For more than two critical actions competing for one server this probability could be adapted.

These values could then be calculated iteratively. In the Tableau in table 4.8 this iterative calculation has been omitted because this does not results in a significant improvement of the approximation value while resulting in a more complex calculation. Here the utilizations $\rho_i = \frac{\lambda_i}{\mu i}$ are used, where $\lambda_1 = 1$, $\lambda_2 = 2$, $\mu_1 = 10$ and $\mu_2 = 5$, as defined in the model.

The approximation errors (relative error $\delta x_i = \frac{\Delta x_i}{x} * 100$) [22] for the throughputs D_i are:

$$\delta D_{req1} = \delta D_{end1} = \frac{|0,8800 - 0,8664|}{0,8664} = 1,55\%;$$
$$\delta D_{req2} = \delta D_{end2} = \frac{|1,4146 - 1,4079|}{1,4079} = 0,48\%.$$

For the comparison of the prediction of the mean number of service requests n_i, the mean number of tokens in the places p_{act1}, p_{act2} in the GSPN model is interpreted as $n_{Client\,1}$ resp. $n_{Client\,2}$ and calculated as:

$$p_{act1} = \pi_2 + \pi_5 = \frac{240}{277} \approx 0,8664;$$
$$p_{act2} = \pi_2 + \pi_3 = \frac{195}{277} \approx 0,7040.$$

The mean numbers of service requests executed in the critical actions $n_{Critical\ Action\,1}$ resp. $n_{Critical\ Action\,2}$ are the sums of the mean number of tokens in the places p_{req1} (waiting) and p_{acc1} (executed) as well as p_{req2} (waiting) and p_{acc2} (executed) because in the multiplexer model the requests are handled in parallel:

$$p_{req1} + p_{acc1} = \pi_3 + \pi_4 + \pi_6 = \frac{37}{277} \approx 0,1336;$$
$$p_{req2} + p_{acc2} = \pi_5 + \pi_6 + \pi_4 = \frac{82}{277} \approx 0,2960.$$

This results in the following approximation errors:

$$\delta_{n_{Client\,1}} = \frac{|0,8664 - 0,8800|}{0,8800} = 1,55\%;$$
$$\delta_{n_{Client\,2}} = \frac{|0,7040 - 0,7073|}{0,7040} = 0,47\%;$$
$$\delta_{n_{Critical\ Action\,1}} = \frac{|0,1336 - 0,1200|}{0,1336} = 10,18\%;$$
$$\delta_{n_{Critical\ Action\,2}} = \frac{|0,2960 - 0,2927|}{0,2960} = 1,11\%.$$

After setting up the FMC-QE model and the corresponding Tableau, several different system and load scenarios could be predicted, as shown in figure 4.27. In this chart the service time of the second critical action X_{CA2} is varied from $0,001s$ to $0,8s$, with a constant service time of $0,1s$ for the first critical action ($X_{CA1} = 0,1$). Then the corresponding throughputs of subnet 1 (λ_1) and subnet 2 (λ_2) are observed. For comparative reasons the network was also modeled and evaluated using the Time Augmented Petri Net Tool TimeNET[5] [133, 134]. In the chart it can be seen that for the upper throughput, the throughput λ_2 of the network with the critical action

[5]TimeNET, Website: http://www.tu-ilmenau.de/TimeNET, January 2010

2, the performance predictions of FMC-QE and the Time Augmented Petri Net predictions of TimeNET are nearly identical. For the lower curve λ_1, the predictions are fine for $X_{CA2} < 0,5$. For $X_{CA2} \geq 0,5$ the precomputed utilization ρ_{CA2}, needed for the calculation of $m_{sem-mpx_{CA1}}$, which is defined as $\rho_{CA2} = \frac{\lambda_2}{\mu_2}$, would be ≥ 1 ($\lambda_2 = 0,5$, $\mu_2 = \frac{1}{X_{CA2}} \leq 0,5$) and is therefore set to 1. This is the case if the critical action becomes the bottleneck of the subnet because this pre-defined utilization calculates the utilizations of the servers if the bottleneck server is fully utilized ($\rho_{bott} = 1$). However, initializing $\rho_{CA2} \leq 0,9$, as also shown in figure 4.27, leads to more precise results. In future work the pre-calculation of this value could be further optimized.

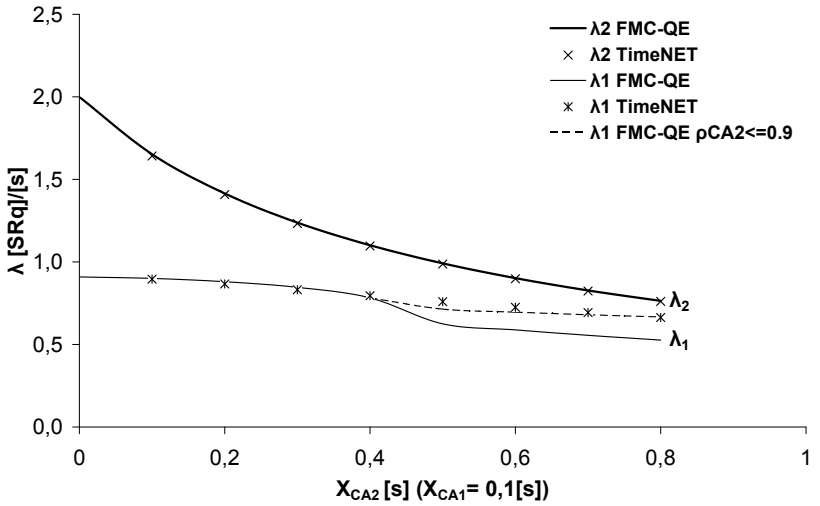


Figure 4.27: Semaphore Synchronization - Chart: Throughput - Critical Action 2 Service Time

4.3.3 Summary

For this exemplary semaphore synchronization problem the integration of the handling of the semaphore by a multiplexed resource and therefore elimination of the control flow [147], the summation method [17, 18] and ideas from the method of complementary delays [71] result in a very precise approximation of the performance values, as seen in figure 4.27, without the need of the calculation of the steady state distribution. Therefore, the complexity reduces from the exponential calculation of the $\binom{n}{k}$ states, where n = number of Petri Net - places ($\approx$ number of queues $\approx$ number of service stations) and k = number of tokens (service requests) to the linear complexity calculation ($O(n)$) of FMC-QE, where n is the number of basic and hierarchical service stations.

4.4 Comparisons

In this section FMC-QE is compared to the foundations described in chapter 2. As an introduction to this section figure 4.28 relates the compared methods to the range of the considered steady state variables.

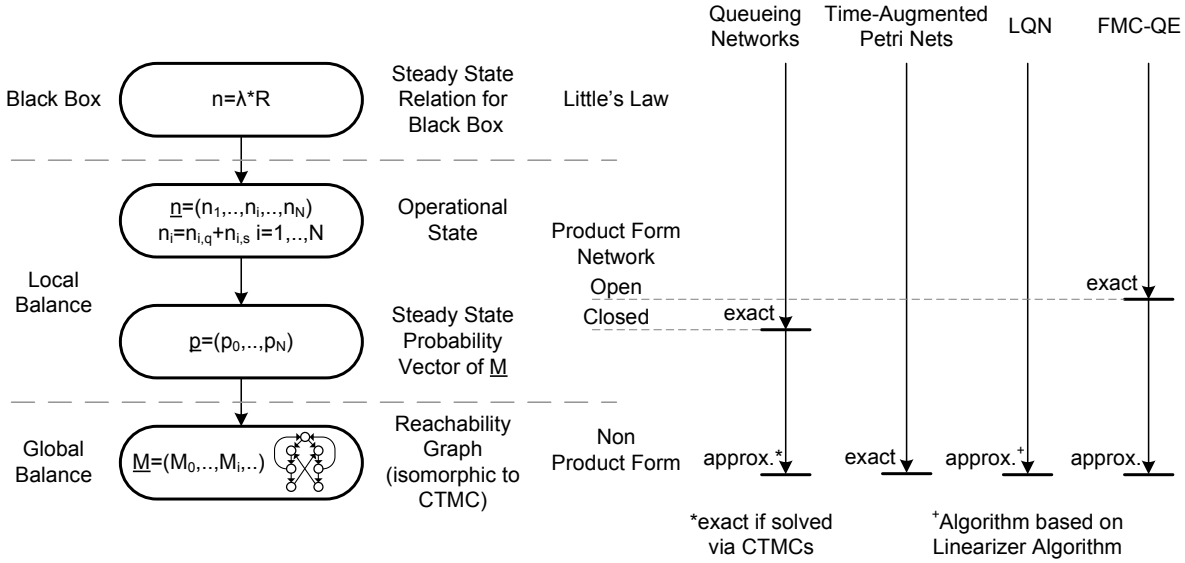


Figure 4.28: Range of Steady State Variables [140] and Methods

The range of steady state variables in figure 4.28 can be mainly divided into three sections: black box, local balance and global balance. On the black box level the steady state relation of the black box is computed on the basis of Little's Law [98]. If the considered system is of type product form, where the overall state of the system could be computed as a product of the single states of the different stations and the states of the different systems are independent from each other, the different stations are in local balance. This leads to an efficient computation of the steady state probability vector and the corresponding operational states through product form solutions. The operational state and the steady state probabilities are in a close relation as for example the number of service requests in a station maps onto the steady state probability of that station. If the steady state probabilities of the different stations are not independent from each other, the underlying problem is of type non product form. Then the performance values of the system have to be calculated on the basis of global balance through reachability graphs or continuous time Markov chains (CTMCs).

Time Augmented Petri Nets, further compared in subsection 4.4.2, analyze the systems on the basis of reachability graphs respectively the corresponding continuous time Markov chain (CTMC). Therefore, Non-Product Form problems can be addressed with the drawback of high computational complexity due to state-space explosion. There are also product form solutions defined for Time Augmented Petri Nets which are further discussed in section 2.2.4 and in subsection 4.4.2.

Methods of the Queueing Theory, further compared in subsection 4.4.1, and especially methods for Product Form Queueing Networks could abstract from this level and solve the networks on the basis of independent steady state probabilities of the different servers. Product form solutions of the Queueing Theory are therefore powerful in terms of computational complexity in solving this kind of questions. If the underlying problem is of type Non-Product Form,

they also analyze the network on the level of CTMCs (as Time Augmented Petri Nets) or use approximations [18].

Layered Queueing Networks (LQN), further compared in subsection 4.4.3, evaluate the models on the basis of an approximative Linearizer algorithm of the Mean Value Analysis [29, 59]. They consider open as well as closed workloads where an open workload can also generate a closed sub-behavior [58].

In FMC-QE the performance values of every basic server is computed and then the performance values of the network are computed in a bottom up approach. There are exact solutions for open product form networks as well as approximative solutions for closed networks (see section 4.1) and selected non product form problems as the semaphore synchronization (see section 4.3).

Beside the calculation of the performance values there are also many differences in the modeling of the different methods where fundamental differences are summarized in table 4.9.

	Queueing Networks		**Time Augmented Petri Nets**		**Layered Queueing Networks**		**FMC-QE**	
Modeled Type of Structures	System Modeling	Math. Model	System Modeling	Math. Model	System Modeling	Math. Model	System Modeling	Math. Model
Server Structures (static)								
flat	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
hierarchical	No[1]	Yes[2]	No	Yes	Layered[3]	Yes	Yes[4]	Yes
Control Structures (dynamic)								
flat	No[5]	Yes	Yes	Yes	Yes	Yes	Yes	Yes
hierarchical	No[1,5]	Yes[2]	Yes[6]	Yes	Layered[3]	Yes	Yes[4]	Yes
Service Req. Structure (content)								
flat	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
hierarchical	No[1]	Yes[2]	Yes[6]	Yes	Layered[3]	Yes	Yes[4]	Yes

1: Only early ideas like in Denning, Buzen [43] (section 4.4.1 and 2.3.5)
2: Hierarchies are distinguished through names of stations = indices
3: Layered flat Network - no mandatory usage of the Forced Traffic Flow Law and units of service requests, no hierarchical transformation (section 4.4.3)
4: Hierarchical transformations through units of service requests and the Forced Traffic Flow Law (section 3.2.2)
5: Data Flow - only exceptions like Fork/Join or Synchronization Nodes (section 4.4.1)
6: Through Colored Petri Nets [80]

Table 4.9: Comparison of Modeling Aspects

Table 4.9 differentiates the compared methodologies by their type of modeled structures (server, control and service request) and if the structures are defined flat or hierarchical in the system modeling and the underlying mathematical analytical model. Amongst others, this is further described in subsection 4.4.1 to 4.4.3 where FMC-QE is further compared to the other methodologies. An important definition in this distinction is:

Hierarchy [143]: A hierarchy implies a service request decomposition on one layer into different service requests on a lower layer.

In FMC-QE the decomposition is done by control service requests, the final service by operational service requests.

Beside mathematical analytical approaches, performance simulation is another method to predict the quantitative behavior of systems. In subsection 4.4.4 FMC-QE and simulative approaches are shortly compared to each other.

4.4.1 Queueing Theory

The Queueing Theory, described in section 2.1, delivers the main mathematical basis for the FMC-QE Calculus. The hierarchical calculations in the Calculus are based on the Forced Traffic Flow Law [68, 79, 95] for relations between hierarchical layers (vertical) and Little's Law [98] for relation within a hierarchical layer (horizontal). For the calculations of the performance values of the logical and multiplexer servers FMC-QE also uses the mathematical background of the Queueing Theory [67, 79, 83, 88, 89, 125]. Despite the Forced Traffic Flow Law and Little's Law, in the calculations of whole nets the investigations of Jackson [77, 78] and Baskett, Chandy, Muntz and Palacios [8] on Product Form Queueing Networks and the independence of the different servers were used. In the calculations of closed networks the summation method [17] was integrated into the calculus and compared to results examples calculated through the results of Gordon and Newell [64] and the Mean Value Analysis (MVA) [118]. The summation method in combination with the idea of modeling the synchronization through a multiplexer [147] and complementary delays [71] was also used for the performance estimation of semaphore synchronization problems.

While the Queueing Theory has strengths in the calculations of performance values, especially of Product Form Queueing Networks, the weaknesses are in the lack of expressiveness modeling power. While Queueing Networks focus on the modeling of server structures and service request flows, the modeling of control flows is not possible. There are extensions, discussed later, in which Fork-Joins or Synchronizations are modeled but in the classical case control flows could not be modeled. Furthermore, in this extensions the control flow is integrated into the static structures. While in FMC-QE the modeling of server structures, which is related to Queueing Theory, is only one view in the three-dimensional modeling space, in Queueing Theory this leads to this control flow modeling problem and therefore less expressiveness in the modeling of complex systems.

Furthermore, in FMC-QE the modeling of service requests as a tuple of value and unit, which enables the transformation of service requests at hierarchical borders and therefore the hierarchical modeling, has roots in the Queueing Theory - especially through the investigations of Denning and Buzen [43]. But there this approach was not systematically used in order to define hierarchies of service requests on arbitrary hierarchical levels, in [43] there were only two levels of *job* and *requests*. As already described in section 2.3.5, there was no definition of service request transformations in [43] and [95] and therefore real hierarchies could not be defined.

Queueing Networks with Fork-Join and Synchronization Nodes

There are extensions of classical Queueing Networks to include Fork-Join Queues [3, 84, 96] or Synchronization nodes [117]. Fork-Join queues [3] are queueing systems with parallel servers and a synchronized arrival and departure stream. Service requests could arrive in a batch and are then immediately dispatched to one server each (fork). The different service requests are then queued into infinite buffers and individually handled by the corresponding server. If all service requests of the batch are finished, the batch is recomposed and departed (join), as shown in figure 4.29.

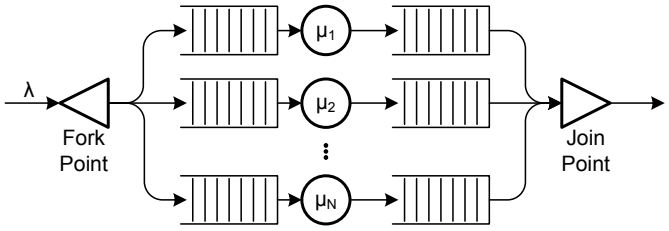


Figure 4.29: Fork-Join Queueing Model [96]

Synchronization nodes consist of M infinite capacity buffers where service requests arriving on M distinct random input flows are stored (one buffer per flow), as depicted in figure 4.30.

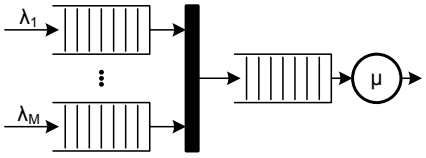


Figure 4.30: SM/M/1 Queue [117]

The service requests are stored until each buffer contains at least one service request. Then the service requests are grouped to one service request and instantaneously released in a synchronization departure [117]. In [117] it is shown that the synchronization of independent Poisson flows at a synchronization node converge to a Poisson process with a rate which is equal to the minimum of all input flows. They also integrate the synchronization nodes into Jackson networks [77, 78] and state that [117]: If in a generalized Jackson network with an acyclic synchronization process (every synchronization node is disconnected from itself - routing probability $p_{ii} = 0$) and finite synchronization buffers, the joint distribution of the equilibrium queue length process at all exponential server nodes is asymptotically product form and the equilibrium departure process converges weakly to independent Poisson processes, as the size of the synchronization buffers goes to infinity.

The results of the investigations of the Fork-Join Queues and especially the Synchronization nodes support the performance predictions of the hierarchical (parallel) activities of FMC-QE, described in section 3.4.5, while in these predictions the overall service rate is also equal to the minimum of all rates.

4.4.2 Time Augmented Petri Nets

For the modeling of the dynamic behavior and the control flow the time augmented adaptations of the FMC Petri Nets in FMC-QE also consist of timed and immediate transitions like Generalized Stochastic Petri Nets (GSPN) [101, 102], described in section 2.2.3. The distinction of operational and control flow in FMC-QE Petri Nets has similarities to Colored Petri Nets [80] where attributes are associated to tokens like in Colored GSPNs [33] (referring to [12]). Following the classification of section 2.2.1, FMC-QE Petri Nets are in the class of Timed Transitions Petri Nets (TTPN) without reservation while the reservation is often irrelevant because conflicts are mostly modeled using immediate transitions.

In networks with several inter-server dependencies, the performance estimations of Time Augmented Petri Nets are more precise than the FMC-QE approximations because in Time Augmented Petri Nets the global balance state of a Non-Product Form Network is computed, while

in FMC-QE these Non-Product Form problems are approximated though transformations into product form solutions. But, in contrast to FMC-QE with it's linear complexity in the calculation of the performance values, the algorithms for the calculation of the performance values of Time Augmented Petri Nets suffer from the state space explosion problem through the calculations of the global balance equations and the associated linear equation systems.

In FMC-QE the steady state behavior of the systems is of interest. In contrast to Time Augmented Petri Nets and Petri Nets in general, the transient behavior as well as an analysis for deadlocks is not in the focus in FMC-QE.

In the following some extensions of the Time Augmented Petri Nets are briefly compared to FMC-QE.

Product Form Petri Nets

In comparison to Non Product Form SPNs, stochastic Petri Nets with product form solutions, described in section 2.2.4, can calculate larger Petri Nets by avoiding the state-space explosion problem through product form solutions. But still, in comparison to the linear complexity in FMC-QE, the different approaches need complex calculations. In the approach of Lazar and Robertazzi [94] there is a need for the investigation of the reachability state which is the main drawback in this approach. In the approach of Henderson et al., even in the highly optimized version of Haddad et al. [70], only the recognition if an SPN could have a product form solution has a complexity of $O(|T|^2)$ steps [70] where T is the set of transitions. The approach of Florin and Natkin [52] is limited to closed synchronized queueing networks and there are efficiency problems in the solving of the linear system for the determination of the constant vector of their matrix product form approach [52]. Furthermore, Robertazzi [119] states that the general limitations of product form solutions for SPNs in concurrent resource sharing and synchronization limit the modeling space of SPNs. For resource sharing models Robertazzi [119] states that blocking excluded an SPN from product form solutions and only if resources are immediately available or requests for resources which are not immediately available are immediately cleared from the system, for which he states that this kind of resource sharing is not so widely distributed in real systems. From the viewpoint of isolated circulations [94] Robertazzi [119] also conjectures that synchronization could prevent higher dimensional models of having a product form solution. Furthermore, the product form SPNs still only model the behavior of the systems and not the server structures, like FMC-QE, and therefore this clear distinction for a complexity reduction and larger modeling capability is not included.

Queueing Petri Nets (QPN)

While the foundations of FMC-QE are mainly FMC and the Queueing Theory and mathematical foundation relies on the Queueing Theory, the Queueing Petri Nets [9, 10, 13, 14], further described in section 2.2.5, are an extension of a colored GSPN, through the integration of scheduling strategies into the places. This is the basis for the main differences between QPNs and FMC-QE. While in FMC-QE the quantitative system properties are modeled in three views, in a QPN both quantitative and qualitative (logical) properties are mixed in one network. The analysis of qualitative properties, like timeless traps or boundness, is not in the scope of FMC-QE. For the analysis of quantitative system properties QPNs must often be analyzed by inspecting the reachability set of the corresponding colored GSPN where the queue is modeled with CGSPN elements [14]. Though this limitation QPNs could describe queues in a

more convenient way without having to specify queues by pure Petri Net elements [14] but in the calculations they still suffer from the state space explosion problem of the Time Augmented Petri Nets.

Product form QPNs [11, 12] could address some problems of the state-space explosion problem in QPNs but since the networks are transformed to an equivalent SPN, the problems of product form SPNs are still the same and the complexity in the calculation of the performance values is still high while the modeling power of the original QPNs is reduced.

The HQPNs [15] deliver exact solutions for large networks and reduce the state pace explosion problem by one order of magnitude but still in contrast to FMC-QE with it's linear complexity in the calculation of the performance values, the HQPN suffers form the state space explosion problem. Also in HQPNs the different subnets have to be isolated through the input and output places while in the different views in FMC-QE and a clearer distinction of logical and multiplexer servers a logical server could be handled by an arbitrary multiplexer. But of course, if there are a lot of control flows and synchronization issues in the modeled system, an HQPN could be more attractive than an FMC-QE model if the HQPN is still computable while FMC-QE uses approximations for synchronizations.

4.4.3 Layered Queueing Networks (LQN)

In [59] the LQN approach, further described in section 2.3.6, especially the LQN Solver (LQNS), was compared to other layered queueing system approaches where as a result LQN covers more system features than any other attempt. In table 4.10 this comparison is imported and extended by a comparison to FMC-QE.

Feature	**LQNS**	**FMC-QE**	MOL	SRVN	TDA	Ramesh	MOD	Fontenot	WSQN-HQN	Kurasugi	APERA
FULL-ACCESS	**yes**	**yes**	no	yes	yes	no	no	no	no	no	yes
Device Scheduling	**FHPSh**	**Mpx**	FPHS	FPH	F	FP	FP	F	FP	FP	FP
Task Scheduling	**FH**	**FPSh**	F	F	F	F	F	F	F	F	F
Open arrivals (OPEN)	**yes**	**yes**	no	yes	no	?	?	yes	yes	yes	?
Infinite-servers	**yes**	**yes**	yes	yes	yes	yes	yes	yes	yes	yes	yes
SERV-PATTERN	**SD**	**SD**	S	SD	S	SD	D	S	SD	SD	SD
VAR	**yes**	**yes**	yes	yes	no	yes	no	?	yes	no	no
PAR	**yes**	**yes**	no	no	no	no	no	no	no	no	no
REPL	**yes**	**yes**	yes	no	no	no	no	no	no	no	?
ASYNC	**yes**	**yes**	no	yes	no	yes	?	no	no	yes	?
Forwarding	**yes**	**yes**	no	no	no	no	no	no	no	no	no
FAST, INTERLOCK	**yes**	**-**	no	yes	no	no	no	no	no	no	no

Where F: FIFO, P: Preemptive Priority, H: Head-of-Line Priority, R: Random, Sh: Processor Sharing, Mpx: Multiplex
S: Stochastic Phase, D: Deterministic Phase, ?: Unclear from the reference

Table 4.10: Comparison of LQNS to FMC-QE and other Layered Queueing Systems [59]

FULL-ACCESS means that a server can issue a request to any server at a lower layer, rather than just to the layer below it [59]. In FMC-QE, through the distinction of logical and multiplexer servers, this is also possible but hierarchically embedded. While the logical servers are defined through the service request structures, the relations through logical server layers are by definition only to the lower layer but this is no restriction while the servers are the logical layers following the service request structure. From the logical to multiplexer servers a connection from every layer to every layer is possible.

For the modeling of device scheduling in LQN the strategies FIFO, Preemptive Priority, Head-of-Line Priority and Processor Sharing and for task scheduling FIFO and Head-of-Line Priority are realized. In FMC-QE the device (multiplexer server) is modeled as a multiplexer and therefore the scheduling strategies are molded at Task (logical server) level. There, in FMC-QE, all possible scheduling strategies of the Kendall Notation, for which steady-state formulas have been set up, could be used. By now, FIFO, Preemptive Priority and Processor Sharing as well as deterministic server and infinite server have been realized but the calculus could be easily extended to other strategies by importing the formulas at the basic server station level.

The LQN Solver supports open Poisson arrival processes as well as closed models [59]. The standard hybrid load model in FMC-QE, where the arrival rate as well as the overall number of service requests could be defined, as described in section 3.4.2, follows the calculations of open queueing networks where the integration of the summation method with an iterative solution, described in section 4.1, enhances the modeling power to closed models. So in FMC-QE open and closed models are also handled.

The handling of infinite-servers for the modeling of for example client behavior is integrated into the LQN Solver as well as in FMC-QE.

In the LQN Solver both stochastic and deterministic patterns of request for lower-layer service (SERV-PATTERN [59]) are integrated. While in the FMC-QE Tableau the formulas for the different basic servers could be exchanged in order to fit to the real behavior of the system, also both stochastic and deterministic patterns are integrated.

An arbitrary variance of CPU demands (VAR), possible in LQN, is also possible to model and evaluate in FMC-QE.

Parallelism (PAR) and replication (REPL) used in LQN to model parallel scenarios as well as increasing the scalability of solutions by replicating servers or structures [59, 108] is also supported in FMC-QE. In FMC-QE parallelism in the handling of service requests, like the handling of sub-request 1 by server X and handling of sub-request 2 by server Y, is modeled in the Petri Net and then the different sub-requests are represented by one line each in the Tableau. Replication is supported by the ability of defining multiplicity coefficients for both logical and multiplexer servers on all hierarchical levels.

Asynchronous messages (ASYNC) and the forwarding of service requests (Forwarding) have not been explicitly modeled in this thesis but could also be modeled and evaluated in FMC-QE.

The solver features FAST (a fast-coupling correction for multiclass FIFO servers with different service times [59]) and INTERLOCK (a correction for correlated requests due to shared resources in generating arrivals [59]) are integrated into the LQN Solver for the improvement of the approximations. While the calculation of the performance values in FMC-QE is different to LQNS, these improvements have not been considered.

Whereas LQN and FMC-QE address similar problems, as described in this section, the calculation of the performance values differ fundamentally. The FMC-QE Tableau is mainly based on Little's Law and the Forced Traffic Flow Law. Although the visit ratios are mentioned, the Forced Traffic Flow Law is not a central concern. The LQN Solver is mainly based on Mean Value Analysis [59]. Furthermore, when hierarchies are defined through service request transformations and service request unit transformations [143], the Layered Queueing Networks are considered as flat networks with layers and no hierarchies because the Forced Traffic Flow Law is used in the detection of bottlenecks in LQNs [58] but it is not mandatory in the system modeling and the transformation of service requests and units of service requests like in FMC-QE. Also in FMC-QE there are many other fundamental differences to LQN like that in FMC-QE

is a stricter modeling from the view of the service request structures. But overall, the whole approach is interesting and in later collaborations to the LQN Research Group, where a first contact has already been initiated, further differences and similarities could be found.

4.4.4 Performance Simulations

The results of the cooperation with the Enterprise Platform and Integration Concepts Group of Prof. Dr. h.c. Hasso Plattner in the comparison of the Perfact Simulation environment [42] to FMC-QE and a cooperation [116] with Mathias Fritzsche, SAP Research CEC Belfast, for a coupling of FMC-QE and the Model-Driven Performance Engineering (MDPE) [61, 62] show that performance simulations and analytical methods like FMC-QE are more supplementations to each other than rivals.

If in a system there are a lot of control flows or side effects, an analytical model, like in FMC-QE is sometimes hard to derive. Therefore, a simulation and especially a simulation which is embedded in the real environment, like *Perfact*, is easier to establish and sometimes more precise. So analytical models and simulations could complete each other through comparisons for a higher precision and also through the integration of the respectively other methodology. The comparison of Perfact and FMC-QE through a case study accelerated the development of both approaches. In addition, the hierarchical modeling and calculations in FMC-QE would allow an integration of simulated performance values into the Tableau and thereby extending the modeling and evaluation power of FMC-QE. Through the integration of FMC-QE into simulation environments, like in the cooperation with SAP Research CEC Belfast, FMC-QE could provide other performance values like the simulation environment. Additionally, through the linear complexity of the performance value calculation a broad range of possible parameters could be covered in order to support sensitivity analyses which are very time consuming if every simulation takes some time. If interesting parameters are found, the simulation of the system with this parameters could support and refine the computed performance values.

In the next chapter some case studies with FMC-QE are summarized. The results of the comparison of the Perfact Simulation environment and FMC-QE are a part of this and are summarized in section 5.2.

Chapter 5

FMC-QE Case Studies

This chapter provides case studies modeled and evaluated with FMC-QE. Each case study focuses on different FMC-QE related questions.

In the first case study in section 5.1 a service-based search portal was modeled and evaluated using FMC-QE. The focus is on modeling a larger system with, in this case, 62 logical servers - 44 basic server stations and 18 hierarchical server stations on 7 hierarchical levels with a main interest in the computational complexity of the corresponding Tableau.

The second example in section 5.2, the Emergency Risk Management Framework (ERMF), focuses on the comparison of the performance predictions of an FMC-QE performance model, a simulation and the corresponding measured values in the real system. The Emergency Risk Management Framework (ERMF) is a system in the SAP NetWeaver and SAP WebAS Environment[1], originally developed by SAP Research France. This case study was also a case study in the development of the performance simulation environment *Perfact* [42]. For the validation of Perfact and the FMC-QE predictions the performance values of the real system were compared to the performance predictions of the Perfact simulation and the mathematical-analytical predictions of FMC-QE. Therefore, in this case study, also published in [120], the focus is on the differences respectively compliances between the applied methods.

In the third example in section 5.3 the Axis2 web service framework[2] is investigated. This case study focuses on the modeling of an hierarchical protocol stack and the performance prediction of multiplexers and synchronizations in the context of threads and processes. In [37] this case study was also published.

[1]SAP AG, SAP NetWeaver, Website: http://www.sap.com/germany/plattform/netweaver/index.epx, August 2009

[2]Apache Software Foundation, Apache Axis2 Architecture Guide, Website: http://ws.apache.org/axis2/1_3/Axis2ArchitectureGuide.html, August 2007

5.1 HPI Search Portal - a Service based Case Study

HPI Search Portal was a joint project of the "Service-Oriented Systems Engineering" Research School at the Hasso-Plattner-Institute [3]. It has been initiated to serve as a common scenario for research activities in the domain of Service-based systems. One of the major goals of the HPI Search Portal project was to provide a common playground for need finding purposes, problem identifications, technology testing and proof of concepts. A second important goal of the HPI Search Portal project is to show and present the benefits of this SOA-based approach and the distributed development and to integrate many different techniques and programming languages and using only interface descriptions for the integration.

While in this joint project every participant benefited from a different perspective, the gain of the author was in the performance - as well as architecture modeling of the system. From the viewpoint of FMC-QE this case study was a proof-of-concept for a larger system with many hierarchical layers as well as a performance test for a larger Tableau.

The author wants the thank the other Research School members contributed to this project, especially Flavius Copaciu, Benjamin Hagedorn, Frank Kaufer, Harald Meyer, Hagen Overdick, Michael Schöbel and Matthias Uflacker.

5.1.1 Architecture

The HPI Search Portal is a Service-based application which provides a distributed search engine inside a research institute in order to find informations about lectures, lecturers, locations and important events. This composite application aggregates information, gathered and combined from different services, in order to offer a unique search interface combining several other (external) search services. An overview on the architecture of this system is provided in figure 5.1.

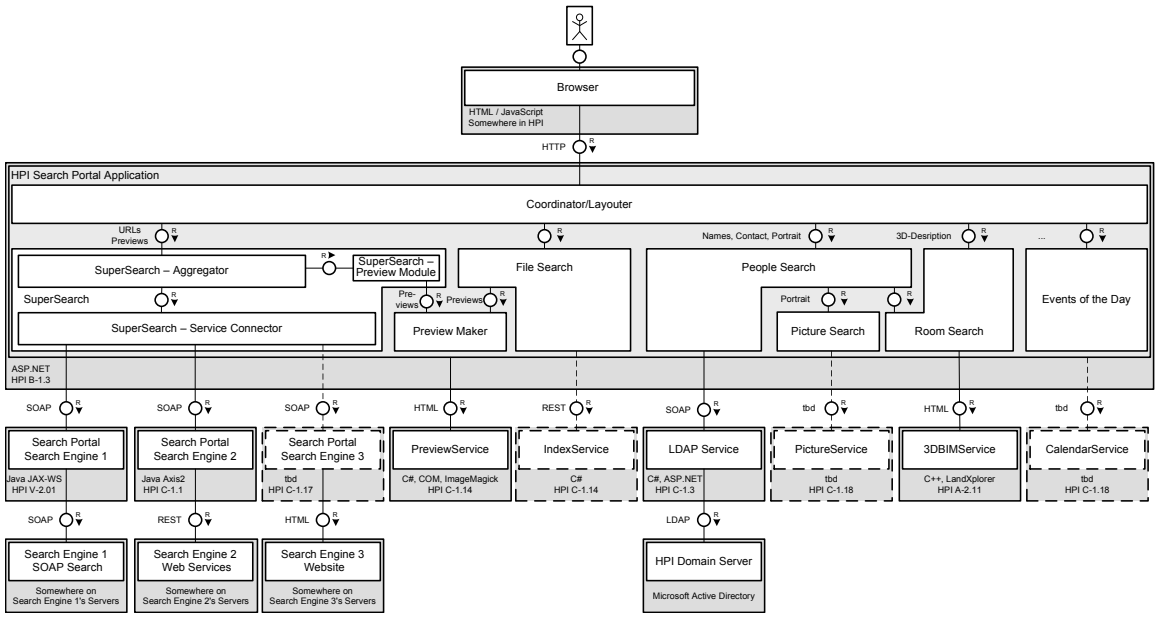


Figure 5.1: HPI Search Portal - Architecture (see Appendix - Figure C.1)

[3] HPI Research School on Service-Oriented Systems Engineering, Website: http://kolleg.hpi.uni-potsdam.de/, February 2010

In the *HPI Search Portal Application* the search for information is divided into 5 sub searches, the *Super Search*, the *File Search*, the *People Search*, the *Room Search* and the *Events of the Day*.

The *Super Search* integrates different public available search engines by using different techniques. The first search engine is integrated by connecting to the SOAP[4] interface of the engine, the second is connected via REST[5] Web Services and the third is integrated by a HTML Website wrapper. The different search responses then are aggregated and ranked and further enriched by previews of the different results.

The *File Search* performs a search for files at the local file servers (Index Search) and also enriches the results with previews.

Names, mail addresses and other contact information of the lecturers and the other institute staff-members could be searched via the *People Search*. The contact information is further enriched by a photo of the person and a 3D path to the person's office.

The *Room Search* could also be directly accessed in order to find a path to a seminar room or an office of a known person.

The search is further complemented by newsworthy event schedules, provided by the *Events of the Day* service.

From a high level the overall service request is composed as follows:

Portal Search Request
- Super Search Request
  - Search Request
  - Preview Request
- File Search Request
  - Index Search
  - Preview Request
- People Search Request
  - List Search Request
  - People Detail Search Request
- Room Search Request
- Event Search Request

In the next subsection on the FMC-QE model this service request structure is further refined and embedded into the technical environment.

5.1.2 FMC-QE Model

The figures 5.2 and 5.3 show an excerpt (service request structures and behavior) of the FMC-QE Model of the HPI Search Portal scenario. Due to the size of the diagram the author will not give a closer description of the model. This figures shall only illustrate that larger FMC-QE models (62 logical servers (44 basic, 18 hierarchical) on 7 hierarchical levels) are also possible and the models and especially the Tableau (table 5.1[6]) scale.

[4]W3C, SOAP Version 1.2, W3C Recommendation 27 April 2007, http://www.w3.org/TR/soap12, February 2010

[5]Representational State Transfer, REST, In: Roy Thomas Fielding, Architectural Styles and the Design of Network-based Software Architectures, Dissertation, University of California, Irvine, 2000, Website: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, February 2010

[6]The performance parameters in this Tableau are test values to show the capabilities of the Tableau and do not reflect the performance behavior of the real system.

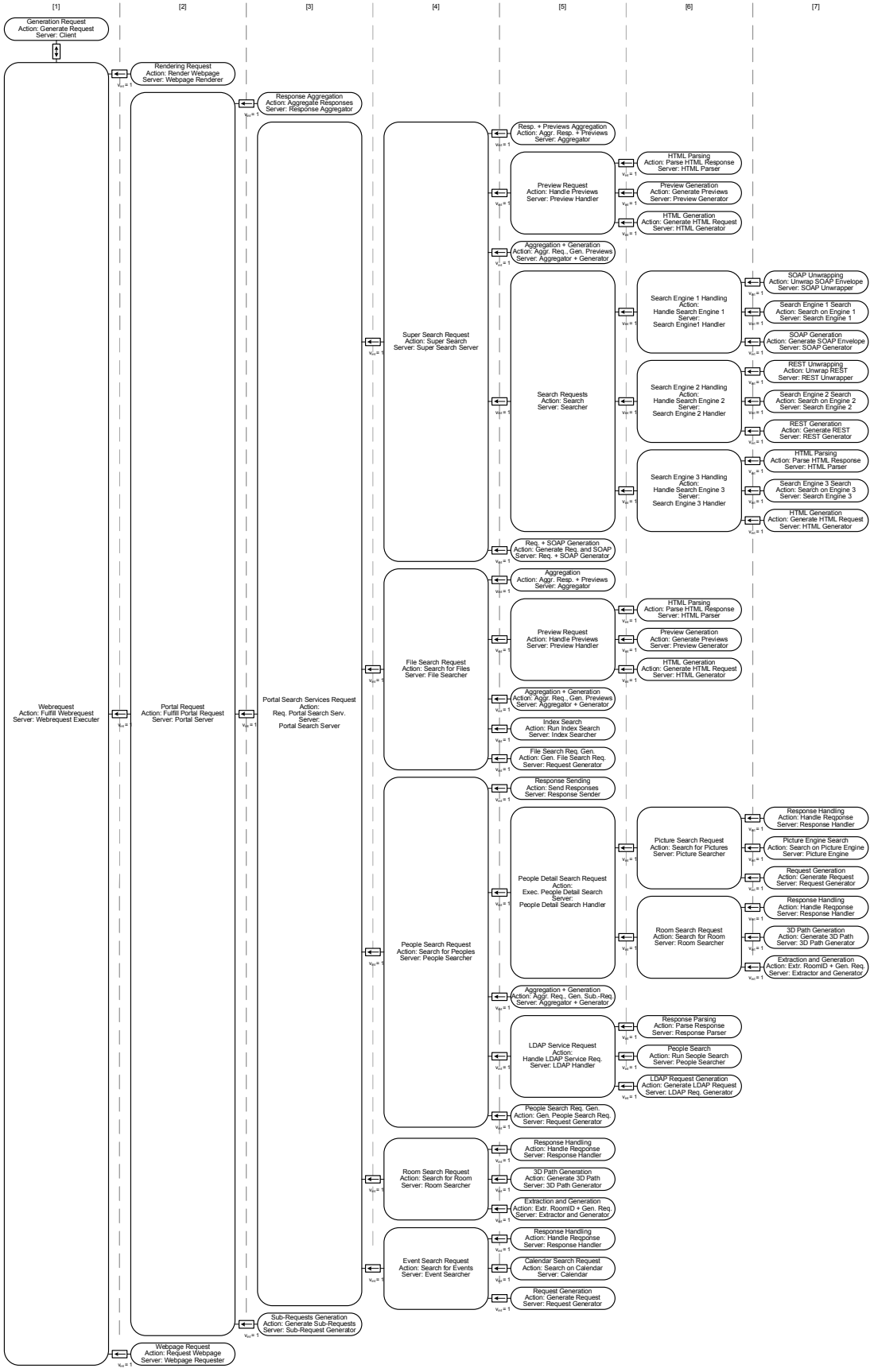


Figure 5.2: HPI Search Portal - Service Request Structure (see Appendix - Figure C.2)

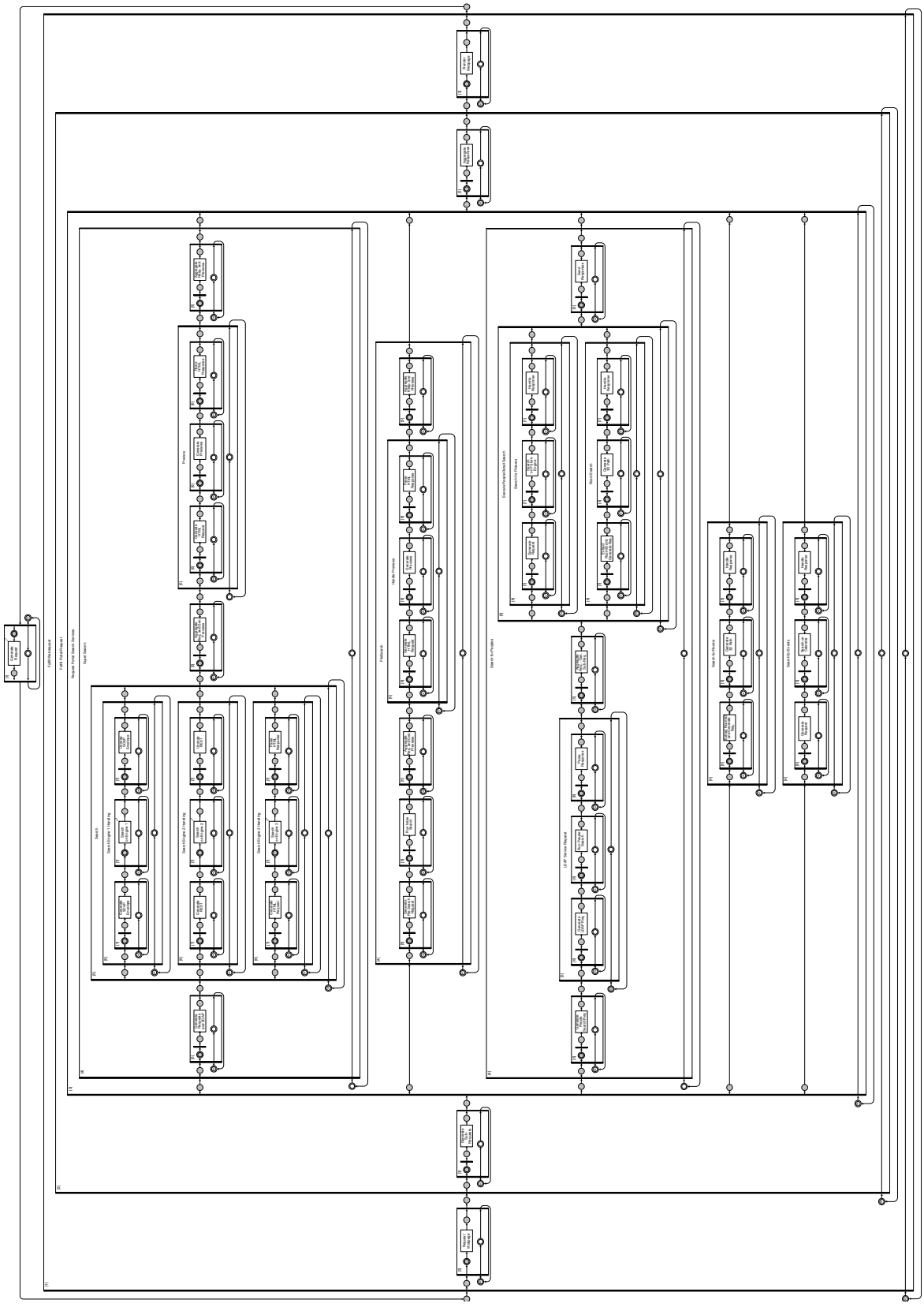


Figure 5.3: HPI Search Portal - Behavior (see Appendix - Figure C.3)

Table 5.1: HPI Search Portal - Tableau (see Appendix - Table B.12)

Experimental Parameters	
n_{ges} [1]	4000
λ_{bott} [1]	12,9313
f	0,9000
λ [1]	11,6382

Service Request Section								Server Section						Dynamic Evaluation Section								
[bb]	i	SRq_i [bb]	$p_{p(i),j}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx$_i$	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
2	1	Rendering Request	1,00	1,00	1,00	1,00	11,638	Webpage Renderer	1	1	1	1	0,0005	0,007	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
3	2	Response Aggregation	1,00	1,00	1,00	1,00	11,638	Response Agg.	1	1	1	1	0,0002	0,002	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
5	3	Resp. + Previews Agg.	1,00	1,00	1,00	1,00	11,638	Aggregator	1	1	1	1	0,0002	0,003	12,931		0,000	0,000	0,900	0,077	0,900	0,077
6	4	HTML Parsing	1,00	1,00	1,00	1,00	11,638	HTML Parser	1	1	1	1	0,0002	0,003	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
6	5	Preview Generation	1,00	1,00	20,00	20,00	232,764	Preview Generator	1	3	3	2	0,0001	1,000	19230,769	0,004	0,000	0,000	0,012	0,000	0,012	0,000
6	6	HTML Generation	1,00	1,00	1,00	1,00	11,638	HTML Generator	1	1	1	1	0,0002	0,003	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
5	7	Preview	1,00	1,00	1,00	1,00	11,638	Preview Handler	1	1	1				12,931		16,200	1,392	1,812	0,156	18,012	1,548
5	8	Aggregation + Generation	1,00	1,00	1,00	1,00	11,638	Aggregator + Gen.	1	1	1	1	0,0003	0,004	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
7	9	SOAP Unwrapping	1,00	5,00	1,00	5,00	58,191	SOAP Unwrapper	3	1	3	3	0,0001	0,152	1515,152	0,013	0,000	0,000	0,038	0,001	0,038	0,001
7	10	Search Engine 1 Search	1,00	5,00	1,00	5,00	58,191	Search Engine 1	3	∞	∞	4	0,0087	1,000	64,657		0,000	0,000	0,938	0,016	0,938	0,016
7	11	SOAP Generation	1,00	5,00	1,00	5,00	58,191	SOAP Generator	3	1	3	3	0,0001	0,182	1515,152	0,013	0,000	0,000	0,038	0,001	0,038	0,001
6	12	Search Engine 1 Handling	1,00	1,00	5,00	5,00	58,191	SE 1 Handler	1	3	3				1515,152		0,000	0,000	1,015	0,017	1,015	0,017
7	13	REST Unwrapping	1,00	5,00	1,00	5,00	58,191	REST Unwrapper	4	1	4	5	0,0008	0,053	65,789	0,221	0,000	0,000	0,885	0,015	0,885	0,015
7	14	Search Engine 2 Search	1,00	5,00	1,00	5,00	58,191	Search Engine 2	4	∞	∞	6	0,0090	1,000	111,111		0,000	0,000	0,885	0,015	0,885	0,015
7	15	REST Generation	1,00	5,00	1,00	5,00	58,191	REST Generator	4	1	4	5	0,0030	0,197	65,789	0,221	0,004	0,000	0,885	0,015	0,888	0,015
6	16	Search Engine 2 Handling	1,00	1,00	5,00	5,00	58,191	SE 2 Handler	1	4	4				65,789		0,004	0,000	2,654	0,046	2,657	0,046
7	17	HTML Parsing	1,00	5,00	1,00	5,00	58,191	HTML Parser	5	1	5	7	0,0024	0,102	42,553	0,273	0,008	0,000	1,367	0,024	1,375	0,024
7	18	Search Engine 3 Search	1,00	5,00	1,00	5,00	58,191	Search Engine 3	5	∞	∞	8	0,0008	1,000	1250,000		0,000	0,000	0,047	0,001	0,047	0,001
7	19	HTML Generation	1,00	5,00	1,00	5,00	58,191	HTML Generator	5	1	5	7	0,0023	0,098	42,553	0,273	0,005	0,000	1,367	0,024	1,373	0,024
6	20	Search Engine 3 Handling	1,00	1,00	5,00	5,00	58,191	SE 3 Handler	1	5	5				42,553		0,013	0,000	2,782	0,048	2,794	0,048
5	21	Search Request	1,00	1,00	1,00	1,00	11,638	Searcher	1	1	1				42,553		0,013	0,001	2,782	0,239	2,794	0,240
5	22	Req. + SOAP Generation	1,00	1,00	1,00	1,00	11,638	Req. + SOAP Gen.	1	1	1	1	0,0008	0,010	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
4	23	Super Search Request	1,00	1,00	1,00	1,00	11,638	Super Search Srv.	1	1	1				12,931		32,413	2,785	7,294	0,627	39,707	3,412
5	24	Aggregation	1,00	1,00	1,00	1,00	11,638	Aggregator	1	1	1	1	0,0006	0,008	12,931	0,900	32,413	2,785	0,900	0,077	33,313	2,862
6	25	HTML Parsing	1,00	1,00	1,00	1,00	11,638	HTML Parser	1	1	1	1	0,0002	0,003	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
6	26	Preview Generation	1,00	1,00	20,00	20,00	232,764	Preview Generator	1	3	3	2	0,0050	1,000	200,000	0,388	0,083	0,000	1,164	0,005	1,247	0,005
6	27	HTML Generation	1,00	1,00	1,00	1,00	11,638	HTML Generator	1	1	1	1	0,0002		12,931	0,900	8,183	0,703	0,900	0,077	9,083	0,780
5	28	Preview	1,00	1,00	1,00	1,00	11,638	Preview Handler	1	1	1				12,931		16,366	1,406	2,964	0,255	19,330	1,661
5	29	Aggregation + Gen.	1,00	1,00	1,00	1,00	11,638	Aggregator + Gen.	1	1	1	1	0,0003	0,004	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
5	30	Index Search	1,00	1,00	1,00	1,00	11,638	Index Searcher	1	1	1	9	0,0050	1,000	200,000	0,058	0,004	0,000	0,058	0,005	0,062	0,005
5	31	File Search Req. Gen.	1,00	1,00	1,00	1,00	11,638	Generator	1	1	1	1	0,0008	0,010	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
4	32	File Search	1,00	1,00	1,00	1,00	11,638	File Searcher	1	1	1				12,931		64,983	5,584	5,722	0,492	70,705	6,075
5	33	Response Sending	1,00	1,00	1,00	1,00	11,638	Response Sender	1	1	1	1	0,0054	0,070	12,931	0,900	0,000	0,000	0,900	0,077	0,900	0,077
7	34	Response Handling	1,00	10,00	1,00	10,00	116,382	Response Handler	1	1	1	1	0,0015	0,194	129,313	0,900	0,000	0,000	0,900	0,008	0,900	0,008
7	35	Picture Engine Search	1,00	10,00	1,00	10,00	116,382	Picture Engine	1	1	1	10	0,0040	1,000	250,000	0,466	0,405	0,003	0,466	0,004	0,871	0,007
7	36	Request Generation	1,00	10,00	1,00	10,00	116,382	Request Generator	1	1	1	1	0,0057	0,737	129,313	0,900	8,100	0,070	0,900	0,008	9,000	0,077
6	37	Picture Search	0,40	5,00	5,00	10,00	116,382	Picture Searcher	1	1	1				129,313		8,505	0,073	2,266	0,019	10,771	0,093
7	38	Response Handling	1,00	5,00	1,00	5,00	58,191	Response Handler	1	1	1	1	0,0025	0,162	64,657	0,900	8,100	0,139	0,900	0,015	9,000	0,155
7	39	3D Path Generation	1,00	5,00	1,00	5,00	58,191	3D Path Generator	1	2	2	11	0,0025	1,000	400,000	0,073	0,001	0,000	0,145	0,003	0,146	0,003
7	40	Extraction and Generation	1,00	5,00	1,00	5,00	58,191	Extractor and Gen.	1	1	1	1	0,0021	0,136	64,657	0,900	8,101	0,139	0,900	0,015	9,001	0,155
6	41	Room Search	1,00	5,00	1,00	5,00	58,191	Room Searcher	1	1	1				64,657		16,202	0,278	1,945	0,033	18,147	0,312
5	42	People Detail Search	1,00	1,00	5,00	5,00	58,191	People Search H.	1	1	1				64,657		24,707	0,425	4,211	0,072	28,918	0,497
5	43	Aggregation + Generation	1,00	1,00	1,00	1,00	11,638	Aggregator + Gen.	1	1	1	1	0,0098	0,127	12,931	0,900	0,000	0,000	0,900	0,077	0,900	0,077
6	44	Response Parsing	1,00	1,00	1,00	1,00	11,638	Response Parser	1	1	1	12	0,0090	1,000	111,111	0,105	0,000	0,000	0,105	0,009	0,105	0,009
6	45	People Search	1,00	1,00	1,00	1,00	11,638	People Searcher	1	1	1	13	0,0024	1,000	416,667	0,028	0,001	0,000	0,028	0,002	0,029	0,002
6	46	LDAP Request Generation	1,00	1,00	1,00	1,00	11,638	LDAP Req. Gen.	1	1	1	12	0,0025	1,000	400,000	0,029	0,001	0,000	0,029	0,003	0,030	0,003
5	47	LDAP Service Request	1,00	1,00	1,00	1,00	11,638	LDAP Handler	1	1	1				400,000		0,002	0,000	0,162	0,014	0,163	0,014
5	48	People Search Req. Gen.	1,00	1,00	1,00	1,00	11,638	Request Generator	1	1	1	1	0,0080	0,103	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
4	49	People Search	1,00	1,00	1,00	1,00	11,638	People Searcher	1	1	1				12,931		32,809	2,819	7,073	0,608	39,881	3,427
5	50	Response Handling	1,00	3,00	1,00	3,00	34,915	Response Handler	1	1	1	1	0,0012	0,047	38,794	0,900	40,909	1,172	0,900	0,026	41,809	1,197
5	51	Calendar Search Req.	1,00	3,00	1,00	3,00	34,915	Calendar	1	1	1	14	0,0090	1,000	111,111	0,314	0,144	0,004	0,314	0,009	0,458	0,013
5	52	Request Generation	1,00	3,00	1,00	3,00	34,915	Request Generator	1	1	1	1	0,0090	0,349	38,794	0,900	8,100	0,232	0,900	0,026	9,000	0,258
4	53	Event Search Request	1,00	1,00	3,00	3,00	34,915	Event Searcher	1	1	1				38,794		49,153	1,408	2,114	0,061	51,267	1,468
5	54	Response Handling	1,00	3,00	1,00	3,00	34,915	Response Handler	1	1	1	1	0,0001	0,005	38,794	0,900	49,153	1,408	0,900	0,026	50,053	1,434
5	55	3D Path Generation	1,00	3,00	1,00	3,00	34,915	3D Path Generator	1	2	2	11	0,0025	1,000	400,000	0,044	0,000	0,000	0,087	0,003	0,087	0,003
5	56	Extraction and Generation	1,00	3,00	1,00	3,00	34,915	Extractor and Gen.	1	1	1	1	0,0002	0,006	38,794	0,900	8,100	0,232	0,900	0,026	9,000	0,258
4	57	Room Search	1,00	1,00	3,00	3,00	34,915	Room Searcher	1	1	1				38,794		57,253	1,640	1,887	0,054	59,140	1,694
3	58	Portal Search Serv. Req.	1,00	1,00	1,00	1,00	11,638	Portal Searcher	1	1	1				12,931		64,983	5,584	7,294	0,627	72,277	6,210
3	59	Sub-Request Generation	1,00	1,00	1,00	1,00	11,638	Sub-Request Gen.	1	1	1	1	0,0003	0,004	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
2	60	Portal Request	1,00	1,00	1,00	1,00	11,638	Portal Server	1	1	1				12,931		81,183	6,976	9,094	0,781	90,277	7,757
2	61	Webpage Request	1,00	1,00	1,00	1,00	11,638	Webpage Req.	1	1	1	1	0,0002	0,003	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
1	62	Webrequest	1,00	1,00	1,00	1,00	11,638	Webrequest Exec.	1	1	1				12,931		97,383	8,368	10,894	0,936	108,277	9,304
1	63	Job Generation	1,00	1,00	1,00	1,00	11,638	Generator	1	1	1	Clien	334,3927		0,003		0,000	0,000	3891,723	334,393	3891,723	334,393

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	HPI B-1.3 - Main	2	0,155
2	HPI C-1.14 - Preview Srv.	8	0,101
3	HPI V-2.01 - SE 1 Serv.Srv.	1	0,001
4	Search Engine 1 Server	∞	
5	HPI C1.1 - SE 2 Serv. Srv.	1	0,019
6	Search Engine 2 Server	∞	
7	HPI C1.17 - SE 3 Serv. Srv.	1	0,024
8	Search Engine 3 Server	∞	
9	HPI C-1.14 - Index Serv.Srv	4	0,005
10	HPI C1.18 - Pic. Serv. Srv.	2	0,040
11	HPI A-2.11 - 3DBIMService	4	0,020
12	HPI C1.3 - LDAP Serv. Srv.	2	0,009
13	HPI LDAP Server	4	0,002
14	HPI C-1.18 - Event Srv.	4	0,027

With the help of the Tableau some performance predictions, examples in figure 5.4 - figure 5.6, could be performed.

The chart in figure 5.4 shows the overall response time R_{ges} dependency of the overall arrival rate λ. While in the initial system configuration in table 5.1 the bottleneck throughput is $\lambda_{bott} \approx 12,93 \frac{[SRq]}{[s]}$, the overall response time grows significant when $\lambda > 10 \frac{[SRq]}{[s]}$. The Tableau shows that the *Main Server* (HPI B-1.3 - Main) is the bottleneck because all associated logical servers have the highest utilization of $\rho = 0,9$, and therefore, an improvement of the Main Server could

speed up the whole system. Three configurations, one with one more CPU ($m_1 = 3$), one with two more CPUs ($m_1 = 4$) and one with three more CPUs ($m_1 = 5$), are also shown in figure 5.4. It can be seen that the upgrade of the Main Server results in a better overall performance of the system for $m_1 = 3$ and $m_1 = 4$. A further upgrade of the Main Server to 5 CPUs does not result in such a huge performance gain as then the *Picture Engine* is the new bottleneck.

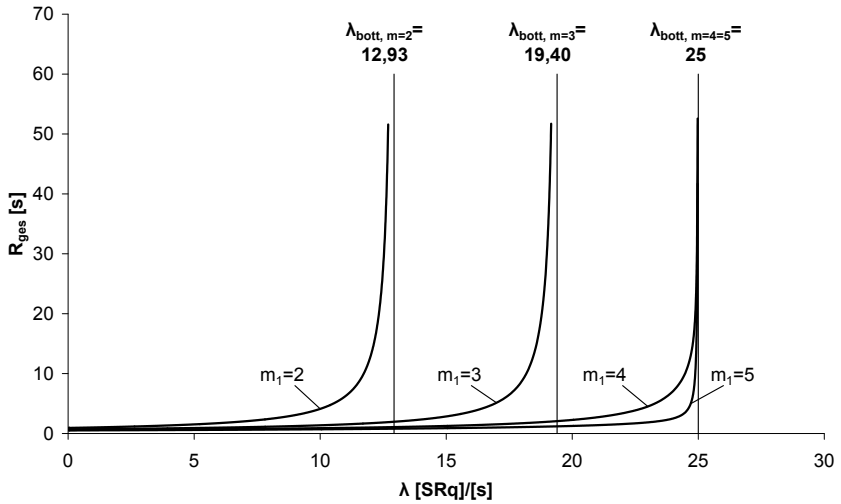


Figure 5.4: HPI Search Portal - Chart: Overall Response Time - Arrival Rate

Figure 5.5 shows the relation between the overall arrival rate λ and the external service time X_{ext} for the initial system configuration of table 5.1 with a mean overall population of $n_{ges} = 4.000[SRq]$. In this configuration a desired overall arrival rate λ in a range from $\lambda = 1\frac{[SRq]}{[s]}$ to $\lambda = 10\frac{[SRq]}{[s]}$ results in an external service time from approximately 7 to approximately 66 minutes (more precisely 396s to 3.965s). If then, in this configuration, the mean external service time would be longer that 7 minutes, referring to figure 5.4, the overall response time R_{ges} is still small ($\lambda \leq 10\frac{[SRq]}{[s]}$ for the initial configuration of a Main Server with 2 CPUs). In the example everyone of the 4.000 students could request the portal ($n_{ges} = 4.000[SRq]$) every 7 minutes.

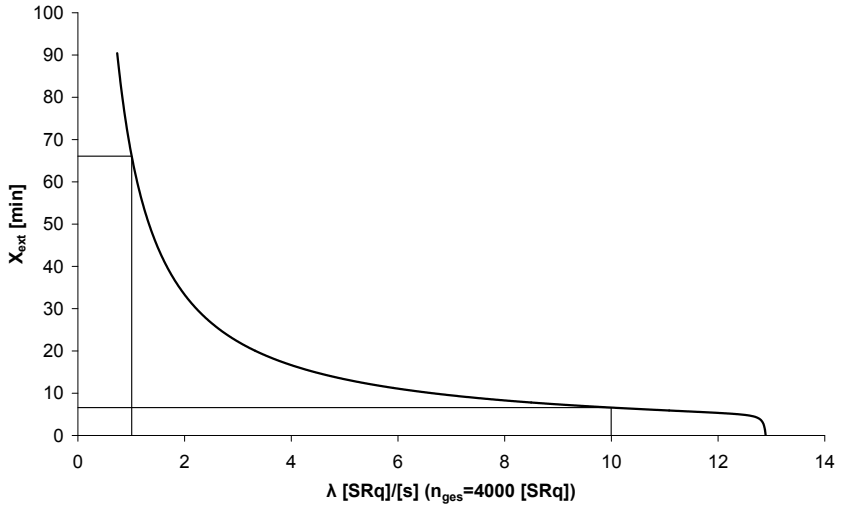


Figure 5.5: HPI Search Portal - Chart: External Service Time - Arrival Rate

Figure 5.6 shows the relation of the overall response time R_{ges} from different configurations of the number of processors in the main server (HPI B-1.3 (Main)), which is referred as multiplexer 1 ($m_{j=1}$). For two different overall arrival rates ($\lambda = 8\frac{[SRq]}{[s]}$ (+) and $\lambda = 12\frac{[SRq]}{[s]}$ (x)) the corresponding overall response times are plotted. This chart then shows that for the configuration with an arrival rate of $\lambda = 8\frac{[SRq]}{[s]}$, two or three processors ($m_{j=1} = 2$) would be the best choice because the change from three processors to more processors does not significantly gain in more performance in this specific scenario. For the configuration with an arrival rate of $\lambda = 12\frac{[SRq]}{[s]}$ it is highly appreciated to install at least three processors because there is a huge performance increase from two to three processors.

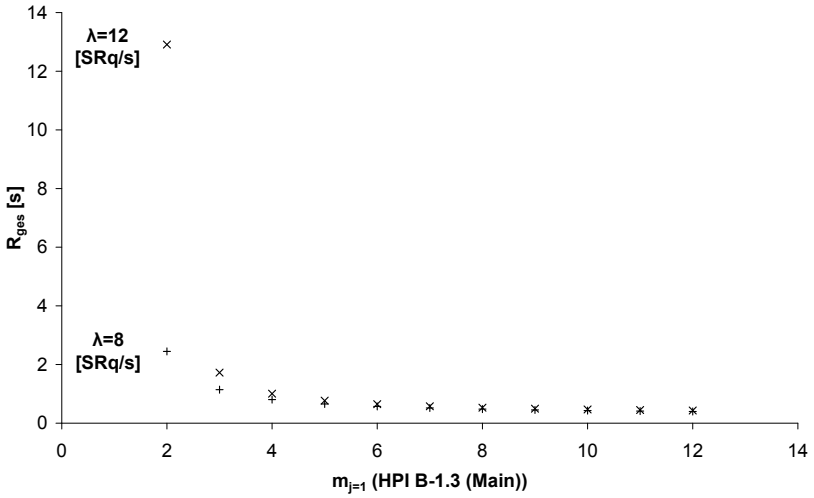


Figure 5.6: HPI Search Portal - Chart: Response Time - Number of Parallel Main Processors

5.1.3 Summary

In this section is was shown that FMC-QE is also suitable for larger models (62 logical servers (44 basic, 18 hierarchical) on 7 hierarchical levels). With the help of the Tableau and the exemplary charts in figures 5.4 to 5.6 different system and load configurations could be evaluated in order to have indications for future system dimensioning of the modeled system. Furthermore, this section gave further insights into FMC-QE, as figure 5.2 shows a slightly modified, more compact layout for larger service request structures. Also this case study shows that the calculations in the Tableau are very fast. For a performance evaluation study 1.000 parameters-changes (recalculation of the whole Tableau + value logging) in the Tableau in table 5.1 were performed in 3,0 seconds [7], which means that a performance prediction for one system configuration of the performance parameters in the Tableau is calculated in approximately 3 milliseconds. In another case study [116], which is not a part of this thesis, another scenario with 46 actions in two classes was evaluated with a prototype version of the FMC-QE Tool by Tomasz Porzucek in approximately 1 millisecond [8].

[7] Tableau developed in Microsoft Office Excel 2003 (11.8307.8221) SP3 in Microsoft Windows XP Professional (Version 5.1.2600 Service Pack 3 Build 2600) on Dell OptiPlex GX620, Intel Pentium 4 HT 3192 Mhz (Family 15 Model 4 Stepping 3) with 2 GB DDR2 RAM

[8] FMC-QE Tool developed under Java 1.6 and deployed on Microsoft Windows XP Professional (Service Pack 3) on Apple MacBook Pro, Intel Core2Duo 2,16 GHz with 2 GB DDR2 RAM

5.2 Modeling of a Service based System: ERMF

In a joint Bachelor's Project of the Research Groups of Prof. Dr. Hasso Plattner / Dr. Alexander Zeier and Prof. Dr.-Ing. Werner Zorn at the Hasso-Plattner-Institute in 2006/2007, called *Perfact,too*, the performance simulation framework *Perfact*, developed in preliminary Bachelor's Project, was further developed and extended. With the help of Perfact the performance of a service-oriented system could be simulated through implementing dummy-components in the real system environment in order to detect bottlenecks and performance problems in the early design phase of the development. As a proof-of-concept for the performance predictions of the *Perfact* framework and the FMC-QE predictions, a benchmark of an existing system was compared to the corresponding *Perfact* simulation and the FMC-QE model. For this case study the Emergency Risk Management Framework (ERMF) was used, a system in the SAP NetWeaver and SAP WebAS Environment[9], originally developed by SAP Research France. Besides the modeling part the main interest for the FMC-QE development was the comparison and validation of the performance values of the three approaches and the analysis of a Service based system.

As a result of this work the paper "Comparison of performance modeling and simulation - a case study" was written and presented at the 15th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2008) in Belfast, UK [120]. This section is based on this paper, especially on the FMC-QE part.

This section is structured as follows: After a short introduction in 5.2.1 the service request structures and the behavior of the system is modeled in 5.2.2. The measurements taken are explained in 5.2.3. In 5.2.4 the corresponding FMC-QE Tableau is shown. In section 5.2.5 the *Perfact* Simulation is explained. Finally, some results are shown in section 5.2.6.

The author would like to thank the co-authors of the paper "Comparison of performance modeling and simulation - a case study" and the other project members of the "*Perfact, too*" Bachelor's Project. In addition, the author wants to thank SAP Research France, especially Cedric Ulmer, Volker Gersabeck and Martin Grund, providing the case study and supporting the project.

5.2.1 Introduction

The system modeled and analyzed in the case study is a Service-based application for the automatic generation of alerts based on pre-defined rules, web service calls and an Event-Condition-Action (ECA) rule engine.

The sketch of the flow in the system is as follows: Due to dynamically defined rules some web services are called from the system. This web services deliver environment data with which alerts could be generated.

[9]SAP AG, SAP NetWeaver, Website: http://www.sap.com/germany/plattform/netweaver/index.epx, August 2009

Figure 5.7 shows the static structures of the system and its environment. The *Timer Application* generates the service requests for the system. The different service requests are received then by the *Control Unit* and dispatched for further processing. The main part of the system is the ECA Engine (Event-Condition-Action Engine). This component is responsible for the applying of the rules in order to generate the risk estimation. This is done by to choosing the most relevant web services and evaluating the web service responses together with the rules. In this case study public available weather and traffic services were used. The responses of the *ECA Engine* are passed to the *Action Performer* through the *Control Unit*. Finally the alert is generated in the *Action Performer*.

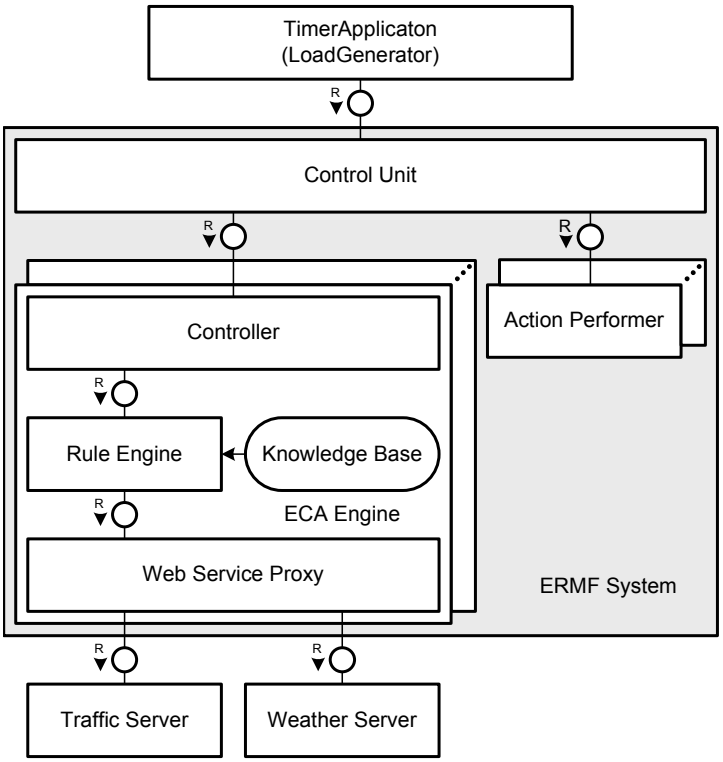


Figure 5.7: ERMF - Static Structure

The test platform was provided by SAP Research France and consists of the SAP NetWeaver Developer Studio 2004s as a development environment and the SAP Web Application Server 6.40 as the application server.

5.2.2 Service Request Structure and Dynamic Behavior

Figures 5.8 and 5.9 illustrate the service request structure and the dynamic behavior of the system. These two diagrams have the same hierarchies and complement each other. While the whole structure with traffic flow coefficients and the corresponding servers are shown in the Entity Relationship Diagram, the dynamic flow with parallelism and the right order of execution is defined in the Petri Net.

As shown in figure 5.8 the *Make Forecast* request is the topmost request which is then hierarchically decomposed into *Initialize Request*, *Evaluate Rules and Call Web Services* and *Generate Alert* requests. Accordingly the *Make Forecast* transition in the dynamic structure (figure 5.9) is de-

composed into a sequence of three corresponding transitions. Furthermore, the *Evaluate Rules and Call Web Services* request was decomposed, as shown in figure 5.8 and figure 5.9.

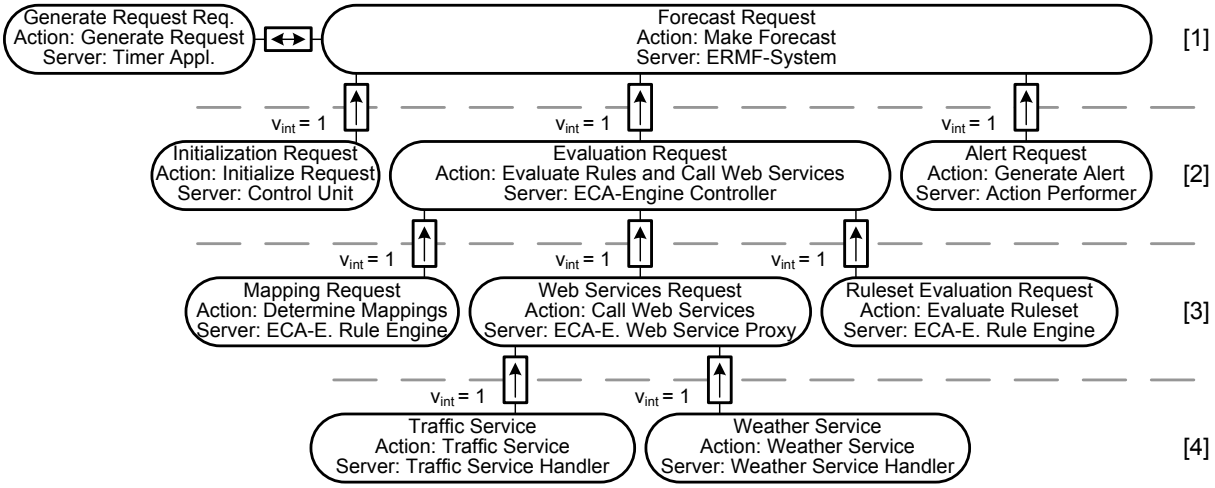


Figure 5.8: ERMF - Service Request Structure

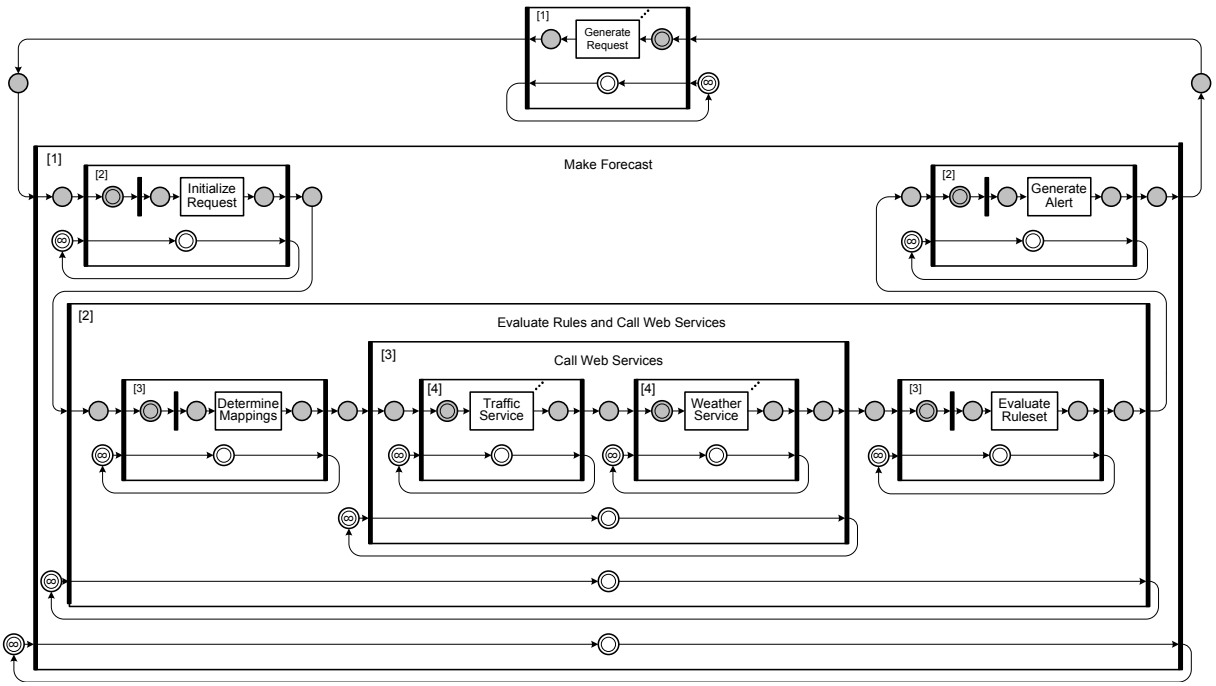


Figure 5.9: ERMF - Dynamic Behavior

5.2.3 Measurements

The original ERMF source code was modified by including a measurement mechanism in order to provide measurements as basis for the simulation and analysis as well as for the comparison of the obtained results. The events handled by the ERMF framework are artificially generated by the Timer Application. These events are traced throughout the system on a limited number of measurement points where the trace data is asynchronous written to a database for later evaluation. For measurements on the real system and during the simulation the same measurement points where used. Thereby, any influence on the systems performance through the tracing is equal on both results.

The charts in figures 5.10 and 5.11 show the response times of the traffic and weather services for 720 respectively 56 measured values. The measured response times have an average of $285.8 \pm 1.09ms$ and $454.91 \pm 8.21ms$ with the corresponding confidence intervals of (95%). The discrete values are a result of measurement resolution.

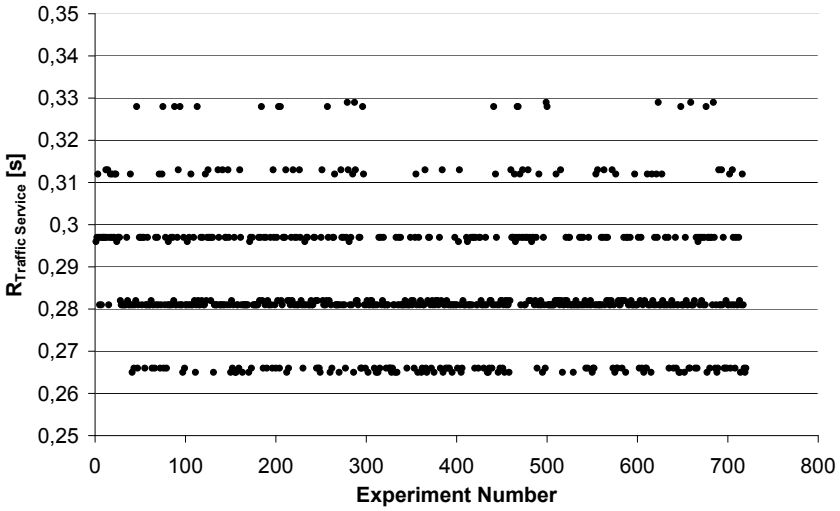


Figure 5.10: ERMF - Chart: Traffic Service - Response Time

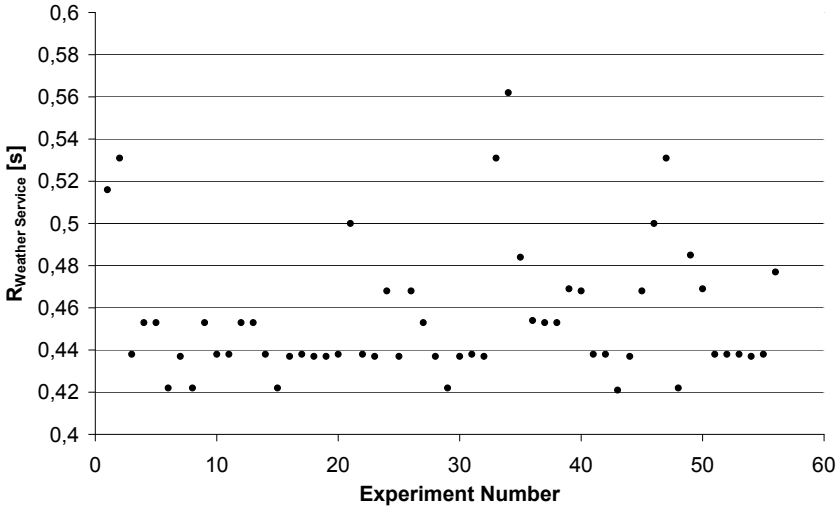


Figure 5.11: ERMF - Chart: Weather Service - Response Time

5.2.4 Analysis

The mean values obtained from the experimental data were used as input parameters for the model evaluation.

In this case study the static structure is simplified to the *ERMF Server* with two processors and the web servers with infinite capacity. The corresponding Tableau is shown in table 5.2. In this Tableau the multiplexer is also exemplified: the queues are at the multiplexers and the logical servers see a fraction of their service request in the multiplexer queue as defined by the multiplexer coefficient:

$$m_{mpx,i} = \frac{v_i^{[bb]} X_i^{[bb]}}{X_j} \frac{m_j}{m_i^{[bb]}}.$$

Table 5.2: ERMF - Tableau (see Appendix - Table B.10)

Experimental Parameters	
$n_{ges}^{[1]}$	3
$\lambda^{[1]}$	1,0000

Service Request Section								Server Section						Dynamic Evaluation Section									
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
2	1	Alert Request	1,00	1,00	1,00	1,00	1,000	Action Performer	1	2	2	1	0,010	0,014	1,408	0,355	0,001	0,001	0,010	0,010	0,011	0,011
3	2	Ruleset Eva. Req.	1,00	1,00	1,00	1,00	1,000	ECA-E. Rule Engine	1	2	2	1	0,150	0,211	1,408	0,355	0,022	0,022	0,150	0,150	0,172	0,172
4	3	Weather Service	1,00	1,00	1,00	1,00	1,000	Weather Serv. Hdl.	1	∞	∞	2	0,470	1,000	2,128		0,000	0,000	0,470	0,470	0,470	0,470
4	4	Traffic Service	1,00	1,00	1,00	1,00	1,000	Traffic Service Hdl.	1	∞	∞	3	0,300	1,000	3,333		0,000	0,000	0,300	0,300	0,300	0,300
3	5	Web Services Req.	1,00	1,00	1,00	1,00	1,000	ECA-E. WS Proxy	1	1	1				∞		0,000	0,000	0,770	0,770	0,770	0,770
3	6	Mapping Request	1,00	1,00	1,00	1,00	1,000	ECA-E. Rule Engine	1	2	2	1	0,200	0,282	1,408	0,355	0,029	0,029	0,200	0,200	0,229	0,229
2	7	Evaluation Request	1,00	1,00	1,00	1,00	1,000	ECA-Engine Cont.	1	1	1				2,817		0,050	0,050	1,120	1,120	1,170	1,170
2	8	Initialization Request	1,00	1,00	1,00	1,00	1,000	Control Unit	1	2	2	1	0,350	0,493	1,408	0,355	0,050	0,050	0,350	0,350	0,400	0,400
1	9	Forecast Request	1,00	1,00	1,00	1,00	1,000	ERMF-System	1	1	1				2,817		0,102	0,102	1,480	1,480	1,582	1,582
1	10	Request Generation	1,00	1,00	1,00	1,00	1,000	Timer Application	1	1	1		1,418		0,705		0,000	0,000	1,418	1,418	1,418	1,418

Multiplexer Section				M/M/m rsp. M/M/∞					
j	$Name_j$	m_j	$X_j^{[1]}$	$\lambda_j^{[bb]}$	$\mu_j^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$n_{i,s}^{[bb]}$	$n_i^{[bb]}$
1	ERMF Server	2	0,710	1,000	1,408	0,355	0,102	0,710	0,812
2	Weather Server	∞	0,470	1,000	2,128		0,000	0,470	0,470
3	Traffic Server	∞	0,300	1,000	3,333		0,000	0,300	0,300

5.2.5 Simulation

For the simulation the Perfact framework is used [42]. In order to run the simulation the following steps are taken iteratively.

First building blocks and their connections are defined. This includes already existing components as well as their interactions. For this case study the external web services exist, whereas the whole ERMF system is simulated. Communication between both sides is taking place on the actual application server. The simulated components also communicate with each other (see figure 5.7).

Next, the system is enriched with setup information needed for the simulation. Connection parameters for the external web services are defined, and the behavior of the generic simulation components (GSC) is customized. In order to run the simulation the dynamic architecture is defined by use cases. The model is based on the UML 2.0 Sequence Charts [10]. All components defined in the first stage can be used in the scenarios. For each activity the performance relevant behavior can be defined by choosing a scenario from a predefined set. Work on storages, algorithmic computation, and main memory consuming processes as the core parts of performance relevant behavior has been identified.

[10]Object Management Group, Unified Modeling Language Specification v. 2.0, OMG UML 2.0 Superstructure Specification, formal/05-07-04, http://www.omg.org/spec/UML/2.0/, July 2005

During the simulation phase generic simulation components are deployed and configured according to this setup. After executing the simulation the system components are triggered according to the workload intensity and performance measurements are traced. The trace data is collected and aggregated then into simulation results where they can be used for comparison and further evaluation of the system.

5.2.6 Summary

Figure 5.12 shows a comparison between simulated results, calculated results and measurements of the real system. The values are computed by changing experimental data in the FMC-QE Tableau and running experiments in the real system and the Perfact simulation. The calculated results are comparable as long as the system is operating under normal conditions ($\rho < 1$). Running at its limits ($\rho \geq 1$), the calculation predicts infinite response times due to the mathematical formulas of Queueing Theory, whereas the real system crashes because of buffer overflows. Similarly the simulation predicts invalid values. In this case the calculated results of FMC-QE helped in understanding the system crashes due to overload and in finding and correcting an error in the implementation of *Perfact*.

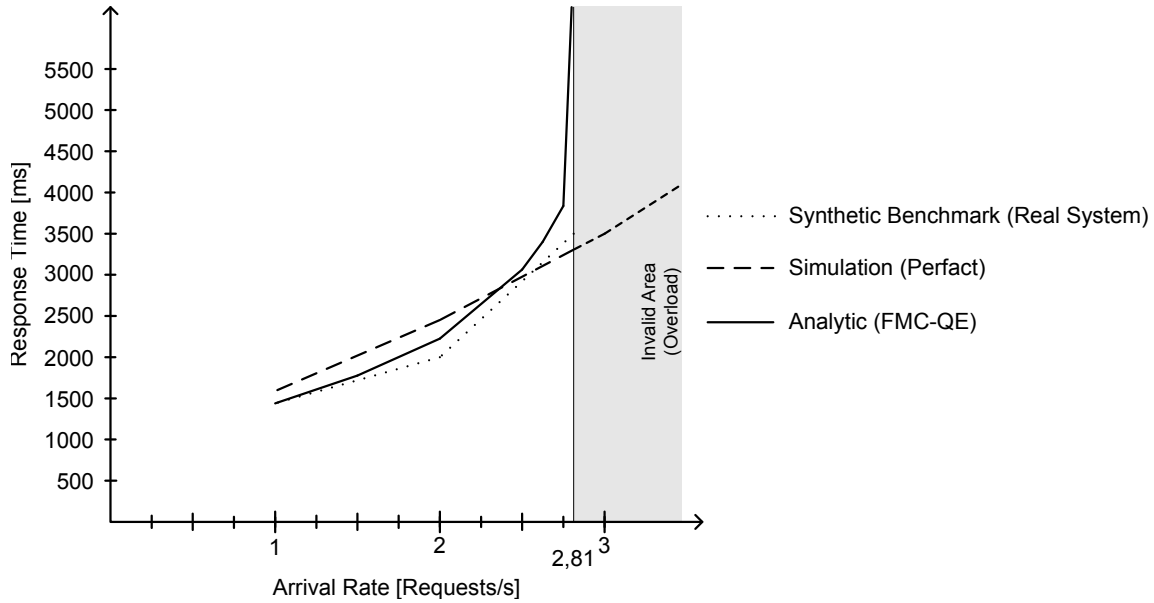


Figure 5.12: ERMF - Chart: Result Comparison

5.3 Modeling of interacting hierarchical Protocol Stacks - Axis2

As a result of a collaboration inside the Research Group, the case study "Analysis and Modeling of the Axis2 Web Service Framework with FMC-QE" was prepared. In this case study the Axis2 Web Service framework was modeled using FMC-QE. This case study was a Proof-of-Concept for the hierarchical modeling and the performance prediction of models with synchronization and multiplexers. Another specialty was the modeling of worker threads and pipelines of sequential tasks. On the basis of this work the paper "Hierarchical Modeling of the Axis2 Web Services Framework with FMC-QE" was written and presented at the 3rd International Conference on COMmunication Systems softWAre and middlewaRE (COMSWARE 2008) in Bangalore, India in 2008 [37].

This section is based on this paper and structured as follows: section 5.3.1 provides an overview on the main components of Axis2, the Apache framework for Web Services and some details of the SOAP processing. In section 5.3.2 the Axis2 FMC-QE model including the Input- and Output-Flows is described. In order to initialize the FMC-QE Tableau in section 5.3.4, performance measurements of an Axis2 installation are conducted in a test environment. This testbed is described in section 5.3.3. Finally, some results are summarized in section 5.3.5.

The author wants to thank the co-authors of the paper "Hierarchical Modeling of the Axis2 Web Services Framework with FMC-QE", especially Flavius Copaciu, for the cooperation.

5.3.1 Axis2 Web Services Framework

Axis2 is a highly modular web service framework developed under the guidance of the Apache Software Foundation[11]. Originally the first implementation was developed by IBM under the name soap4j[12], later donated to the Apache Software Foundation and offered to the public as Apache SOAP[13]. This implementation was a proof of concept with the goal to promote web services and to offer to the users a first contact with this, at that time new, technology. The second iteration, called Apache Axis[14], was developed with the goal of improving the support for the web services specifications (collectively known as WS-$*$) that appeared in the community. Axis2[15], the third iteration, is a redesign of the original Axis code with the purpose of making it easier to provide support for the WS-$*$ standards as well as improving the performance issues that were noticed with the previous implementations.

The Axis2 architecture can be split into two main parts: the core components and the non-core components. Most of the non-core components are available as plug-in modules with multiple implementations available to choose from.

In a system build using Axis2 the framework can be deployed on the service consumer side, the service provider side or both sides of the system. One of such system built with Axis2 is presented in figure 5.13. The client application encapsulates the client logic and makes use of the Axis2 libraries to invoke services offered by the service provider. On the server side, Axis2 is deployed as a web application inside a Tomcat[16] application container. Axis2 does not

[11] Apache Software Foundation, Website: http://www.apache.org/, February 2010

[12] IBM, SOAP for Java, Website: http://www.alphaworks.ibm.com/tech/soap4j/, August 2007

[13] Apache Software Foundation, Apache SOAP, Website: http://ws.apache.org/soap/, August 2007

[14] Apache Software Foundation, Web Services - Axis, Website: http://ws.apache.org/axis/, August 2007

[15] Apache Software Foundation, Apache Axis2 Architecture Guide, Website: http://ws.apache.org/axis2/1_3/Axis2ArchitectureGuide.html, August 2007

[16] Apache Software Foundation, Apache Tomcat Servlet Container, Website: http://tomcat.apache.org, August 2007

depend on Tomcat, it can be deployed as an web application on any J2EE compliant servlet container. The services themselves, containing the business logic, are deployed inside Axis2. For an extensive discussion regarding figure 5.13 as well as the other diagrams that form the model please refer to section 5.3.2.

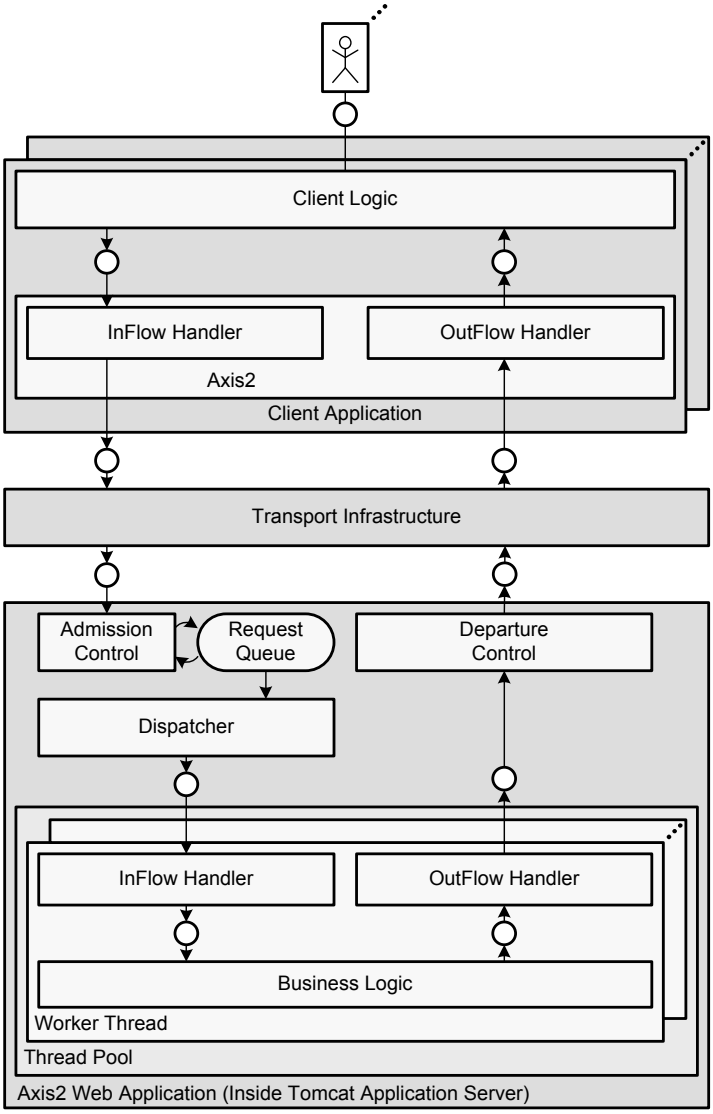


Figure 5.13: Axis2 Based System - Block Diagram

More then a SOAP[17] processing stack Axis2 provides the functionality required to address many of aspects that appear when building a service-oriented environment [112]:

1. Framework to develop, deploy, invoke and manage Web Services.
2. Extensible SOAP processing model.
3. Framework for supporting different Message Exchange Patterns (MEPs), including synchronous as well as asynchronous service invocation.
4. Modular transports and data bindings.

[17] W3C, SOAP Version 1.2, W3C Recommendation 27 April 2007, http://www.w3.org/TR/soap12, February 2010

5. WS-$*$ support via pluggable modules, e.g. WS-Addressing, WS-ReliableMessaging, WS-Coordination or WS-Security [50].
6. Support for Message Transmission and Optimization Mechanism (MTOM[18]).
7. Representational State Transfer (REST[19]) support.

SOAP Processing in Axis2

The Axis2 SOAP processing engine and the different processing stages are in the focus of this case study and will be presented in detail in the following sections. In Axis2 the SOAP processing is implemented as a three layer architecture consisting of handlers, phases and flows [32].

The processing handlers are situated at the lowest abstraction level in this three layer architecture. These handlers are the smallest components in the SOAP processing flow, encapsulating well defined functionality. All handlers implement a well defined interface.

The processing phases are located on the second abstraction level. Each phase manages a specific, logically independent task. The phases implement the same interface as the handlers, and this makes it possible to implement phases through a single handler. In order to realize complex functionality, multiple handlers are chained together implementing a phase. The position of a single handler inside a phase can be specified via a set of rules in one of the configuration files of Axis2. The SOAP processing engine in Axis2 is based on a set of predefined phases and can be extended by user defined phases. The most important phases in Axis2 are[20]:

- Transport - responsible for processing transport related information, e.g. transport headers, and adding this data to the Axis2 message context; in the output flow this phase is responsible for invoking the associated transport handler.
- Pre-Dispatch - in this phase data used for dispatching the message is gathered, e.g. from the HTTP or SOAP headers.
- Dispatch - responsible for matching the incoming message with one of the services deployed in the Axis2 instance.
- User Defined Phases - used for enhancing the capabilities of the framework.

The processing flows belong to the highest hierarchical level in the processing model. These flows are built by chaining multiple processing phases together. Each incoming or outgoing service request is processed by one of the incoming or outgoing flows (*InFlow*, *OutFlow*).

The SOAP processing model of Axis2 can be extended in several ways:

- Adding user defined phases: New phases, implemented via single or multiple handlers, can be inserted in the incoming and outgoing flows. This way new functionality is added to the system.

[18]W3C, SOAP Message Transmission Optimization Mechanism, W3C Recommendation 25 January 2005, http://www.w3.org/TR/soap12-mtom/, February 2010

[19]Representational State Transfer, REST, In: Roy Thomas Fielding, Architectural Styles and the Design of Network-based Software Architectures, Dissertation, University of California, Irvine, 2000, Website: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, February 2010

[20]Apache Software Foundation, Web Services - Axis, Website: http://ws.apache.org/axis/, August 2007

- Adding new handlers: This is done to change or enhance the functionality of one of the existing phases, and these changes can not be done in a user specified phase.
- Adding a new module: Complex Axis2 extensions can also be implemented as modules. These usually are employed when implementing WS-$*$ extensions (e.g. WS-Addressing) and are a set of new modules and/or user phases logically grouped together. The modules are realized as jar files containing a set of handlers and an XML descriptor that specifies the placement of each handler in the processing flows. The modules have to be deployed first and then enabled in the Axis2 configuration files.

5.3.2 Axis2 Model

Axis2 can be used in two different ways, a standalone mode or deployed inside an application server. Due to the limitations of the standalone implementation it is usually deployed on application servers as a web application. This second case is already shown in figure 5.13.

This figure 5.13 describes the mechanism employed when new service requests are received. After the requests pass the *Admission Control*, they are queued in the *Request Queue*. Then the *Dispatcher* is responsible for allocating available threads from the *Thread Pool* for processing the service requests. The allocated thread performs all the processing required by the service request and is returned to the pool as soon as the processing has ended. The size of the thread pool is not static, it grows during the start phase, then it stabilizes between a minimum and a maximum thread count. New threads can be spawned to deal with usage peeks, and idle threads are removed from the pool after a certain period of time. As soon as the processing of a service request has been completed, the service response is sent. The *Departure Control* shows the processing performed upon outgoing service responses.

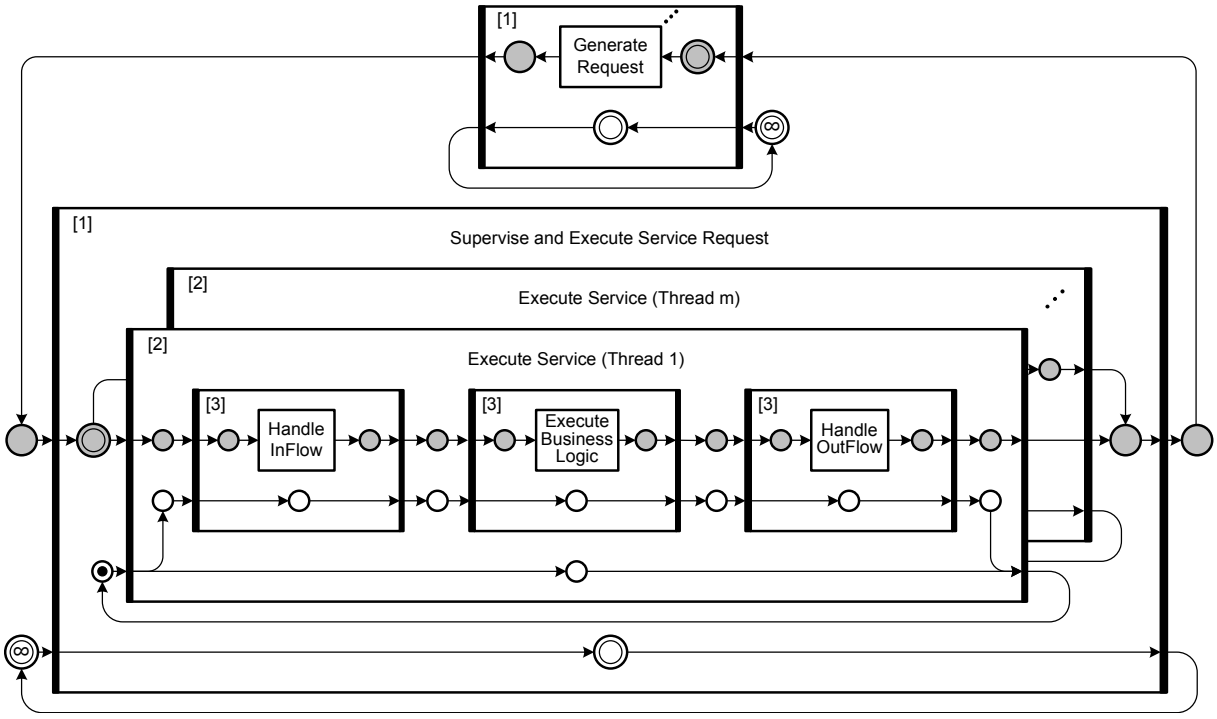


Figure 5.14: Axis2 Dynamic Behavior - Petri Net

The dynamic behavior of the system is presented in figure 5.14. The external world representing the clients generating service requests is modeled via the *Generate Requests* transition. This transition is at hierarchical level [1] just as the *Supervise and Execute Service Request* transition used to model the Axis2 instance. The processing threads are represented using the hierarchical level [2] transition *Execute Service*, while the processing flows and the business logic associated with the service are represented via the corresponding level [3] transitions. In section 5.3.2 the two processing flows are presented in detail. The service requests queue is represented by the infinite place between level [1] and level [2] transitions. By default, Axis2 uses an infinite queue, as shown in the picture. If a specific queue size is set, the model has to be adapted by replacing the infinite place by a finite, multi-token place.

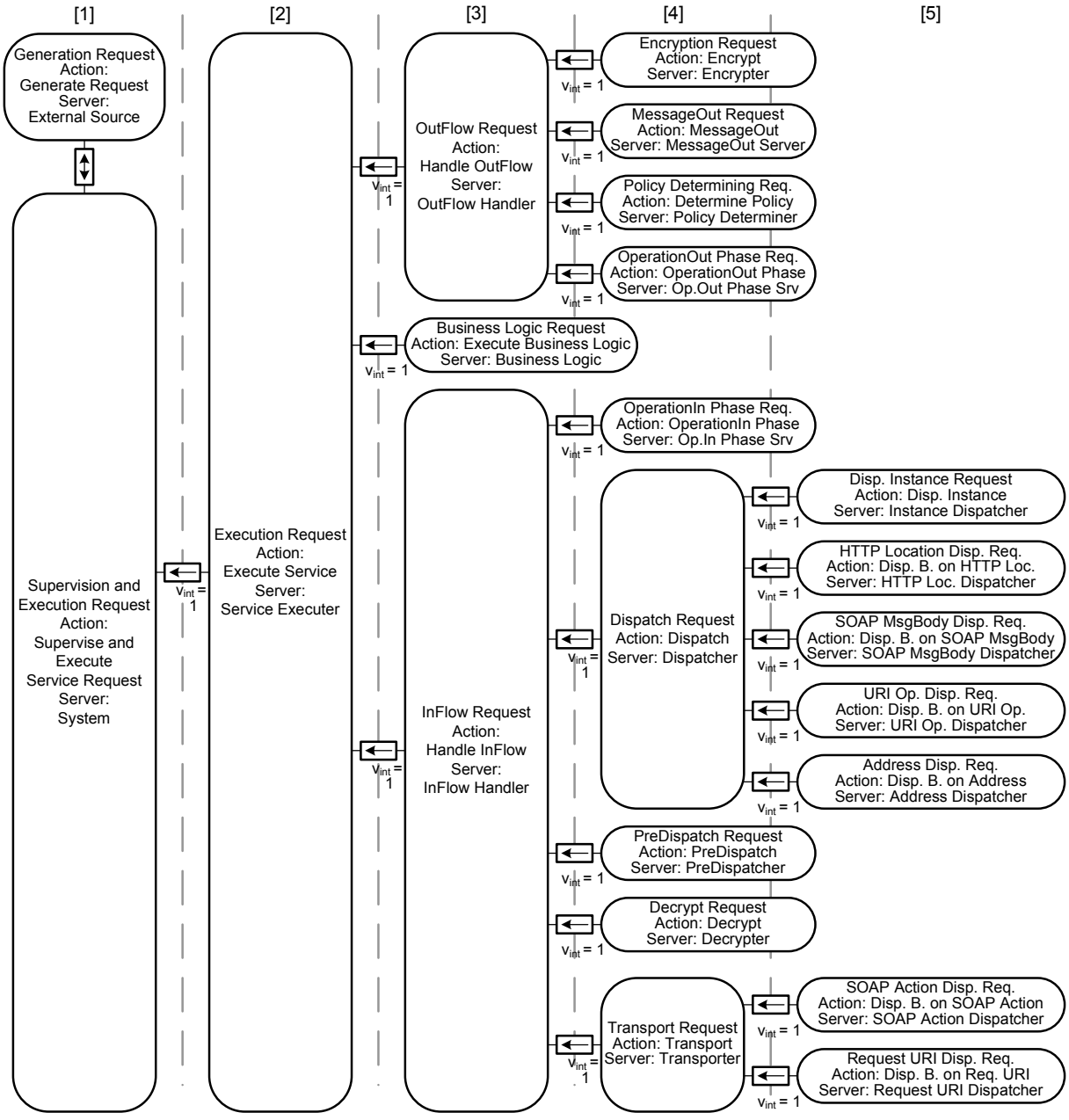


Figure 5.15: Axis2 Hierarchical Service Request Structure - Entity Relationship Diagram

The service request structure is presented in figure 5.15. This figure describes the hierarchical structure of the modeled Axis2 Service Request. This diagram describes among others the mapping between Petri Net transitions and agents in the Block Diagram, for example the *Execute Service* is handled by the *Worker Thread*. Besides this, the hierarchical levels and traffic flow coefficients are presented.

Axis2 Flows

In the Axis2 framework, important parts are the four SOAP processing flows that are part of the framework core: InFlow, OutFlow, InFaultFlow and OutFaultFlow. The first two are responsible for input and respectively output processing, while the last two are responsible for input and output processing in the case that errors have appeared. The error processing flows are quite similar to the regular ones and no longer considered.

The Petri Net detailing the input flow is presented in figure 5.16. This flow is the most complex one in this case study, spawning over three hierarchy levels, from level [3] to level [5].

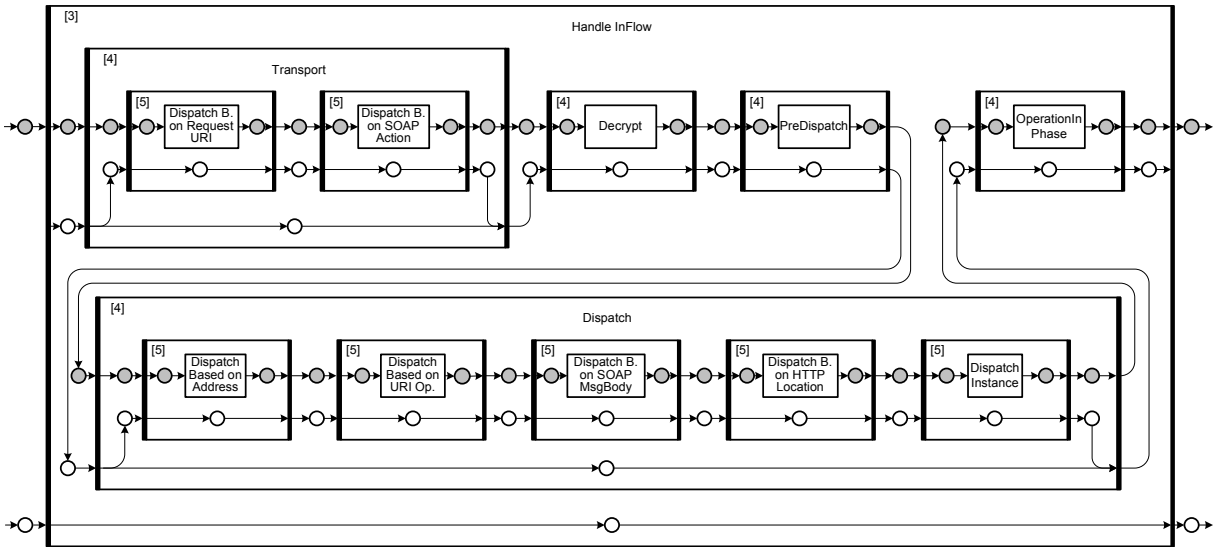


Figure 5.16: Axis2 Input Flow - Petri Net

The output flow, presented in figure 5.17, is simpler than the input flow and has only two hierarchical levels. This simplification is explained by the fact that the output flow has to take into account parameters that have been set during the input flow and by this the range of possible changes that can appear has been reduced. For example, if during the input flow the transport protocol has been decided as HTTP, the output flow will use this information and will not have to do any extra processing in order to determine a transport stack.

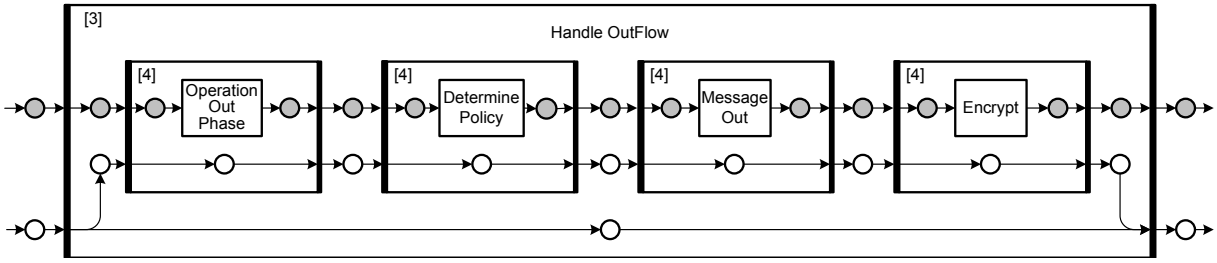


Figure 5.17: Axis2 Output Flow - Petri Net

When comparing the graphical representation from figure 5.14 and 5.16 with the ones found in section 3.3.3, it can be noticed that the transitions have been simplified. This has been done by removing the place representing the queued service requests waiting to be served. This simplification is possible because there are no queues in the processing flows. All the transitions belonging to the same thread will start executing as soon as the operation corresponding to the previous transition has finished. Inside Axis2 the only place where service requests are being queued is the queue in front of the *Thread Pool*.

Extending the Axis2 Model

One of the strengths of Axis is its flexibility and the possibilities to customize its behavior. The core of the system, presented in section 5.3.2, can be extended in order to incorporate other WS-$*$ extension, deployed as modules, that can be enabled or disabled individually for each service or group of services.

When a new module is added to the handler chain and all services make use of that module, the model could be extended. Such changes could be reflected in the dynamic structure of the model by adding new handlers or phases in the existing handler chains, according to the specification of the newly activated module.

It is also possible to have specific modules activated and used only by some of the deployed web services, as mentioned before. In order to cover such a case a decision-making part has to be integrated in the model, as illustrated in figure 5.18, where the *Encrypt* handler is not mandatory but optional. If encryption is activated, the service responses will be processes via the *Encrypt* handler. If encryption is not used, no processing will be performed, as indicated by the no-operation (NOP) transition.

For this situation it is necessary to determine the usage probability of the optional handler and to extend the model to incorporate this, as in figure 5.18. This solution is appropriate if most of the services follow one branch and only a small percent follows the other. The drawback of this solution is the fact that when there are many optional handlers, treating them through averages leads to a loss of representativity of the final results. This can be considered acceptable when such cases account only for a small amount of the total service requests.

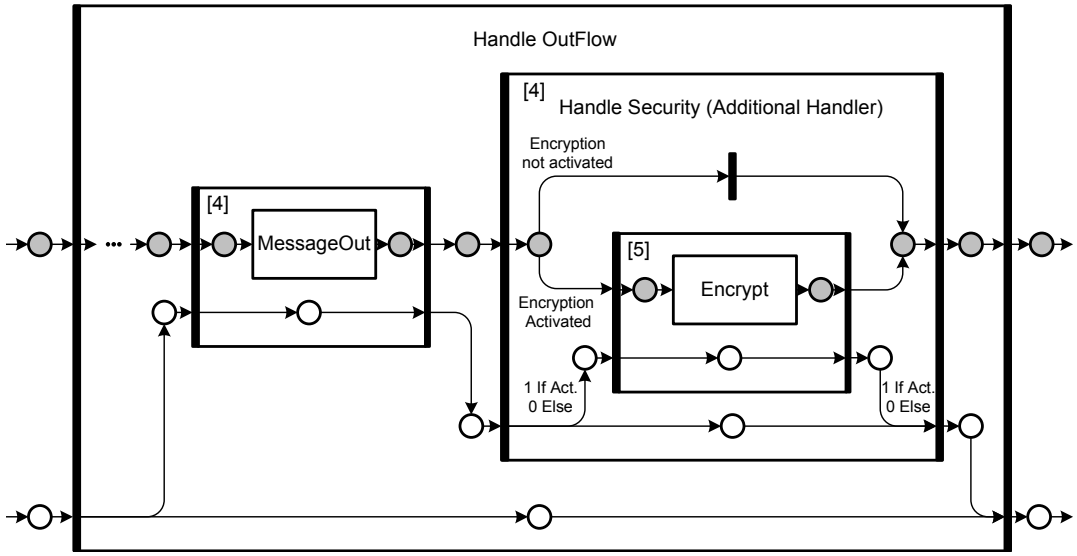


Figure 5.18: Axis2 - Optional Handlers

The second solution to this problem implies transforming the scenario into a multiclass one. Services that use the same path or closely related paths through the handler chain can be grouped together in service classes and the modeling can be done for these classes. Both of these approaches can be modeled and evaluated with FMC-QE.

5.3.3 Testbed Description

In order to perform the performance evaluation and prediction using FMC-QE a benchmark of a server running Axis2 has been done. The benchmarking has been done by using the Java method call System.getNano(). According to JAVA API documentation [21] the method returns the current value of the most precise available system timer in nanoseconds and provides nanosecond precision, but not necessarily nanosecond accuracy. On Apple Mac OS X 10.4[22] this method delivers results with micro second precision. The Axis2 code has been extended with measuring points connected to each handler.

The server instance runs on an Apple MacBook Pro machine with an Intel Core 2 Duo CPU at 2.16 GHz and 2 GB RAM. The machine runs Axis2 version 1.2 inside a Tomcat 6.0.10 application server on a Mac OS X 10.4.10 with Java 1.5.0_07. The experiments were repeated 1.000 times, and the mean value corresponding to each handler has been calculated. For the experiments the service *Version* available by default with each Axis2 distribution has been invoked.

5.3.4 FMC-QE Tableau

With the the help of the Tableau, shown in table 5.3, performance predictions could be derived. In this Tableau, the special behavior of *worker threads* and a *worker pool*, as described in this case study, is modeled. The worker threads and the corresponding logical servers (level [2] and

[21] Oracle Corporation, Java 2 Platform Standard Edition 5.0 API Specification, http://java.sun.com/j2se/1.5.0/docs/api/, February 2010

[22] Apple Inc., Mac OS X, newer Version 10.6 Snow Leopard, Website: http://www.apple.com/macosx/, February 2010

below) are modeled as infinite servers with a multiplex coefficient of $m_{i,mpx}^{[bb]} = \frac{\#Workers}{\#CPUs}$. While the queued service requests are queued at the worker pool, the *system* is modeled as an M/M/n server with the corresponding queue and the aggregated service rates of the worker threads.

Table 5.3: Axis2 - Tableau (see Appendix - Table B.11)

Experimental Parameters	
$n_{ges}^{[1]}$	1000
$\lambda_{bott}^{[1]}$	10,526
f	0,800
$\lambda^{[1]}$	8,421

Service Request Section								Server Section						Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
4	1	Encryption Req.	1,00	1,00	1,00	1,00	8,42	Encrypter	5	1	5	1	0,0006	0,400	689,66	0,002	0,000	0,000	0,012	0,001	0,012	0,001
4	2	MessageOut Req.	1,00	1,00	1,00	1,00	8,42	MessageOut Srv.	5	1	5	1	0,0045	0,400	89,89	0,019	0,000	0,000	0,094	0,011	0,094	0,011
4	3	Policy Det. Req.	1,00	1,00	1,00	1,00	8,42	Policy Det.	5	1	5	1	0,0004	0,400	1000,00	0,002	0,000	0,000	0,008	0,001	0,008	0,001
4	4	OpOut Phase Req.	1,00	1,00	1,00	1,00	8,42	OpOut Phase Srv.	5	1	5	1	0,0004	0,400	930,23	0,002	0,000	0,000	0,009	0,001	0,009	0,001
3	5	OutFlow Request	1,00	1,00	1,00	1,00	8,42	OutFlow Handler	5	1	5				170,65		0,000	0,000	0,123	0,015	0,123	0,015
3	5	Business Logic R.	1,00	1,00	1,00	1,00	8,42	Business Logic	5	1	5	1	0,0325	0,400	12,33	0,137	0,000	0,000	0,683	0,081	0,683	0,081
4	6	OpIn Phase Req.	1,00	1,00	1,00	1,00	8,42	OpIn Phase Srv	5	1	5	1	0,0006	0,400	645,16	0,003	0,000	0,000	0,013	0,002	0,013	0,002
5	7	Disp. Instance Req.	1,00	1,00	1,00	1,00	8,42	Instance Disp.	5	1	5	1	0,0329	0,400	12,15	0,139	0,000	0,000	0,693	0,082	0,693	0,082
5	8	HTTP Loc. Disp. R.	1,00	1,00	1,00	1,00	8,42	HTTP Loc. Disp.	5	1	5	1	0,0003	0,400	1481,48	0,001	0,000	0,000	0,006	0,001	0,006	0,001
5	9	SOAP MB. Disp. R.	1,00	1,00	1,00	1,00	8,42	SOAP MB Disp.	5	1	5	1	0,0001	0,400	4000,00	0,000	0,000	0,000	0,002	0,000	0,002	0,000
5	10	URI Op. Disp. Req.	1,00	1,00	1,00	1,00	8,42	URI Op. Disp.	5	1	5	1	0,0001	0,400	4000,00	0,000	0,000	0,000	0,002	0,000	0,002	0,000
5	11	Address Disp. Req.	1,00	1,00	1,00	1,00	8,42	Address Disp.	5	1	5	1	0,0287	0,400	13,93	0,121	0,000	0,000	0,604	0,072	0,604	0,072
4	12	Dispatch Request	1,00	1,00	1,00	1,00	8,42	Dispatcher	5	1	5				16,10		0,000	0,000	1,308	0,155	1,308	0,155
4	13	PreDispatch Req.	1,00	1,00	1,00	1,00	8,42	PreDispatcher	5	1	5	1	0,0050	0,400	80,00	0,021	0,000	0,000	0,105	0,013	0,105	0,013
4	14	Decrypt Req.	1,00	1,00	1,00	1,00	8,42	Decrypter	5	1	5	1	0,0005	0,400	816,33	0,002	0,000	0,000	0,010	0,001	0,010	0,001
5	15	SOAP Act. Disp. R.	1,00	1,00	1,00	1,00	8,42	SOAP Act. Disp.	5	1	5	1	0,0397	0,400	10,07	0,167	0,000	0,000	0,836	0,099	0,836	0,099
5	16	Req.t URI Disp. R.	1,00	1,00	1,00	1,00	8,42	Req. URI Disp.	5	1	5	1	0,0437	0,400	9,14	0,184	0,000	0,000	0,921	0,109	0,921	0,109
4	17	Transport Request	1,00	1,00	1,00	1,00	8,42	Transporter	5	1	5				11,98		0,000	0,000	1,757	0,209	1,757	0,209
3	18	InFlow Request	1,00	1,00	1,00	1,00	8,42	InFlow Handler	5	1	5				6,59		0,000	0,000	3,193	0,379	3,193	0,379
2	19	Execution Request	1,00	1,00	1,00	1,00	8,42	Service Exec.	5	1	5				5,26		0,000	0,000	4,000	0,475	4,000	0,475
1	20	Sup. and Ex. Req.	1,00	1,00	1,00	1,00	8,42	System	1	5	5			0,400	2,11	0,800	2,216	0,263	4,000	0,475	6,216	0,738
1	21	Generation Request	1,00	1,00	1,00	1,00	8,42	External Source	1	1	1		118,0118		8,42				993,784	118,012	993,784	118,012

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	CPU	2	0,190

Figure 5.19 shows the prediction of the response time of the whole system R_{ges} in relation to the overall arrival rate λ for a configuration with 5 worker threads and 2 CPUs. While in this configuration the bottleneck throughput is $\lambda_{bott} \approx 10,5\frac{[SRq]}{[s]}$, the response times grow rapidly for $\lambda > 9,5\frac{[SRq]}{[s]}$. In addition to this, three other system configurations with more processors are also shown in this figure as an example for the support in future system configuration questions with different ranges of possible arrival rates.

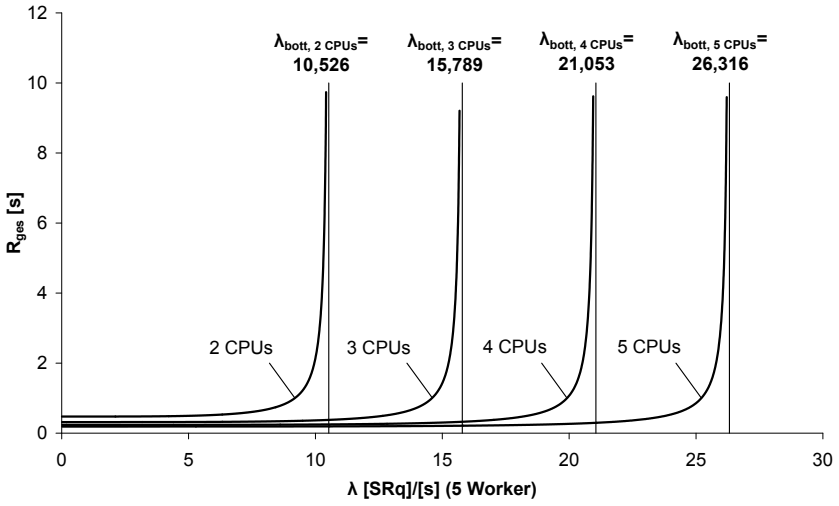


Figure 5.19: Axis2 - Chart: Response Time - Arrival Rate

Figure 5.20 shows the dependence on the total number of service requests (n_{ges}) and the external service time (X_{ext}) as a result of the Response Time Law $X_{ext} = (n_{ges}/A_i) - R_{sys}$. For the initial configuration of 5 workers and 2 CPUs and an desired bottleneck utilization $f = 0,8$ ($\lambda = 8,421\frac{[SRq]}{[s]}$, 1.000 clients could request the server every 2 minutes (118s) each. Diagrams like this could lead to a better understanding of the clients behavior and the relation between the number of clients and their external service time (*Think Time* [43]).

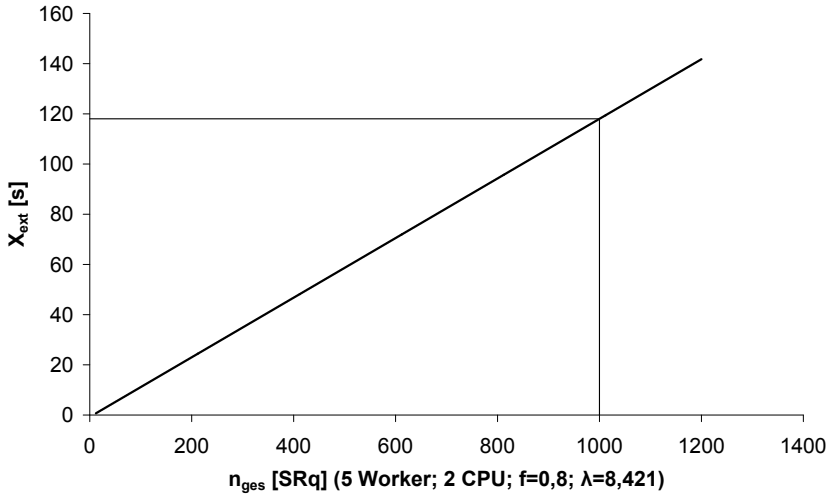


Figure 5.20: Axis2 - Chart: External Service Time - Population

In figure 5.21 the number of available CPUs is increased for the configuration of 5 workers and an overall arrival rate of $\lambda = 8,421\frac{[SRq]}{[s]}$ as an example for decision support of a system configuration question under a specific load scenario. It can be seen that the increase to 3 CPUs results in a strong improvement of the system performance, while in this special configuration more CPUs do not result in a significant performance improvement. A configuration of 1 CPU is not shown here because an arrival rate of $\lambda = 8,421\frac{[SRq]}{[s]}$ is larger than the maximum bottleneck throughput for this configuration ($\lambda_{bott,\ 1CPU} = 5,262\frac{[SRq]}{[s]}$).

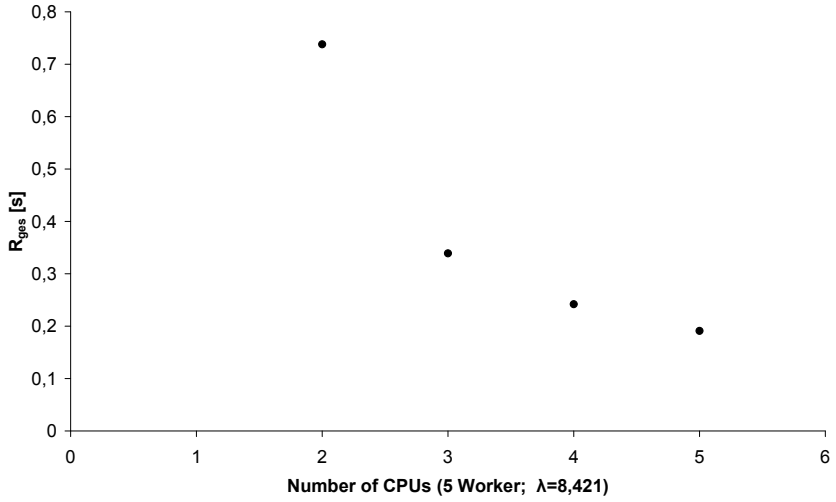


Figure 5.21: Axis2 - Chart: Response Time - Number of CPUs

During the evaluation differences have been noticed between the hierarchically estimated service times and some measurements done at the phase level. The hypothesis is that the differences are due to Java object instantiation time, partially not taken into account in the benchmark.

5.3.5 Summary

In this section an analysis on Axis2 has been done and based on this, an FMC-QE model of the framework has been developed. A complex system has been modeled using FMC-QE and the methodology has shown to be suitable for modeling such systems and depicting their hierarchical structure. A performance model of Axis2 has been done using measurements from a test system. The results have confirmed the methodology and the usability of FMC-QE for performance estimations. The experience and knowledge gathered through this process have provided a better understanding of Axis2 and systems based on Axis2, and is to be of further use in the implementation an adaptive system for service-based Systems by Flavius Copaciu [36].

After setting up the FMC-QE model and Tableau of Axis2, performance predictions could be performed in a fast and simple way. A set of parameters, e.g probabilities, service time or number of CPUs, can be changed in the Tableau and allow to investigate the behavior of the system under a broad range of possible configurations, as depicted in figures 5.19 to 5.21.

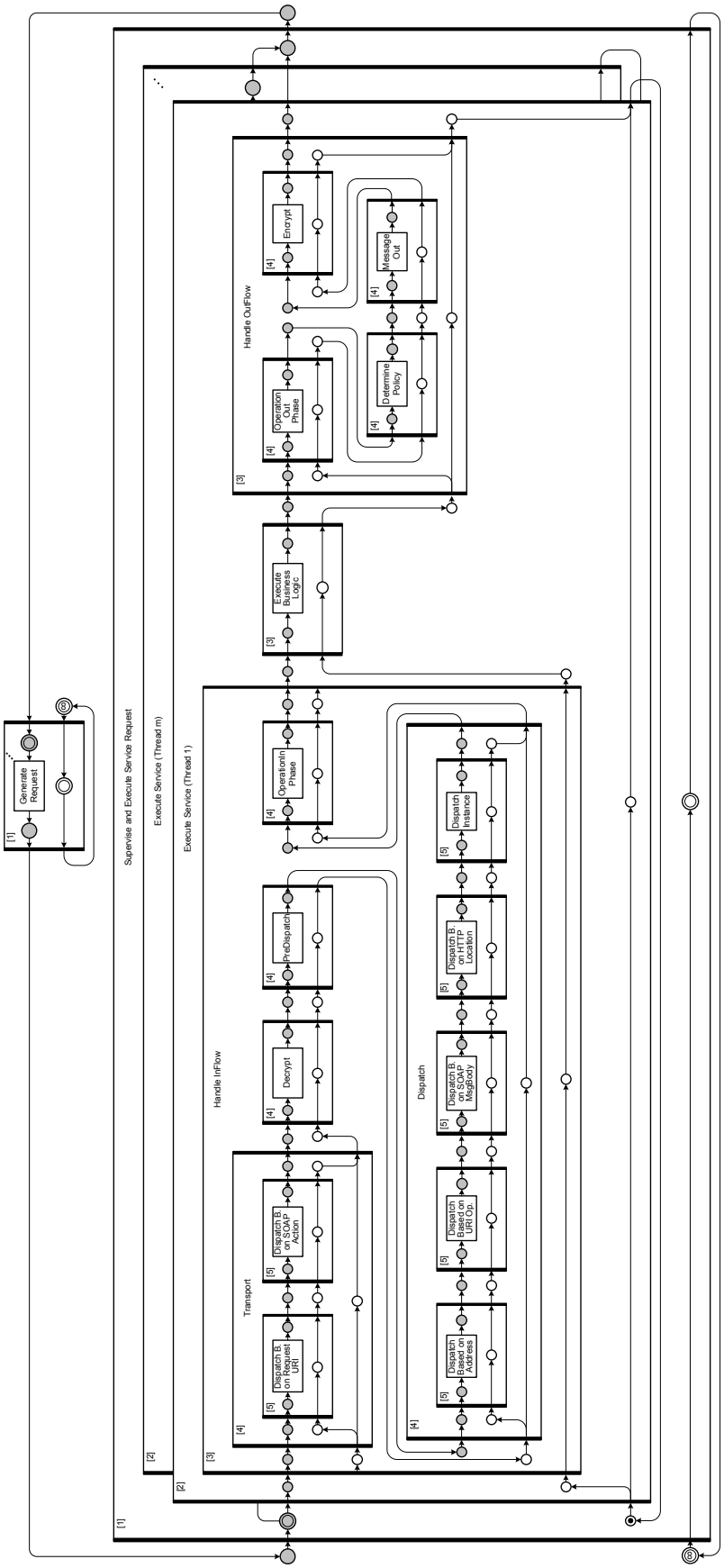


Figure 5.22: Axis2 - Dynamic - All (see Appendix - Figure C.4)

Chapter 6

Conclusions

This chapter summarizes the main contributions and gives an outlook on future work.

Contributions of FMC-QE

The new methodology FMC-QE contributes through:

- The extension of the modeling technique FMC for quantitative modeling and performance predictions as a powerful approach to predict the quantitative behavior of systems adapting FMCs 3-dimensional modeling space with service request structures, server structures and dynamic control flow.

- A consistent modeling from the perspective of hierarchical service requests with a hierarchical multi-level modeling through the usage of the Forced Traffic Flow Law and the modeling of service requests as a tuple of value and [unit] like physical units. Through this hierarchical modeling the model transformations and the complexity reduction through distinction of operational and control states allow the modeling and analysis of complex hierarchical systems in steady state.

- A calculus based on hierarchical equilibriums in steady state with Little's Law for relations within a hierarchical layer (horizontal) and the Forced Traffic Flow Law for relations among hierarchical layers (vertical), deducing a system of equations used in development of interpretable Tableau with easy to change system- and load-parameters which scales for complex systems, without the state space explosion problem through achieving an algorithmical complexity of the Tableau of approximately O(n) –> (n = number of service stations).

- With the different levels of parallelism in both logical and multiplexer server structures, FMC-QE gives the ability to compute a very broad range of possible system configurations instantly. This number of possible system configurations is extended by the ability to change probabilities and experimental parameters, like arrival rate and number of circulating service requests through a strictly modeling of the outside world as servers with a handling of open and closed system as special cases.

- The ability to model and compute multiplexer servers that serve many logical servers (multiplex) which enables FMC-QE to model complex scenarios and the handling of multiclass scenarios integrated through multiplexers and partitioning.

Contributions of the Author

While early ideas of FMC-QE were developed by Prof. Dr.-Ing. Werner Zorn, the author contributed through completing and validating the methodology and making it accessible to a broader community through this thesis including foundations, a description of the whole methodology and the extension and usage of this methodology in examples and case studies. These contributions are in particular:

In the chapters 2 and 3 the author contributed through validating, completing and updating the results of Prof. Dr.-Ing. Werner Zorn through:

- The further development of the FMC-QE graphical notations.
- The development of a more implementation oriented Tableau in cooperation with Tomasz Porzucek and Flavius Copaciu with a specification for parallel and branch request handling and a generalization and standard integration of the multiplexer in the Tableau as well as the further development and integration of parallelism in both logical and multiplexer servers.

The chapter 4 contains the exclusive contribution of the author. These contributions are:

- The handling of closed networks through the integration of the summation method in an iterative computation which complexity is independent from the overall number of service requests in the system.
- The handling of semaphore synchronization scenarios through extending the idea of modeling the critical resource through a multiplexer and combine this approach with the summation method and ideas of the method of complementary delays with the goal of precision and the reduction of computational complexity.
- The comparison of FMC-QE to other performance modeling and evaluation approaches.
- The integration of multiclass problems through a modified multiplexer.

In chapter 5, in cooperations with others, the author applied FMC-QE through case studies as a proof of concept, mostly in the context of service based systems. The contributions of the author were:

- The development of a larger FMC-QE model in order to show the scalability of the approach.
- The support in the quantitative modeling of a service based system as a case study to compare the performance values of a real system, a simulation and the FMC-QE predictions.
- The FMC-QE part in the modeling of the Axis2 Web Service Framework in order to improve the hierarchical modeling diagrams and to focus on performance prediction of multiplexers and synchronizations.

Future Work

The most important future work is actually the implementation of an FMC-QE Tool in order to open the methodology to a broader usage. This includes the development of a FMC-QE tableaux interpreter as well as the development of transformations of other, possibly non hierarchical, models to FMC-QE in order to further broaden the addressed systems, available for modeling with FMC-QE.

Furthermore, this includes the extension of statistical evaluations, e.g. quantiles and approximation of mathematical distributions through preprocessing, like Weibull or log-normal for the service times and arrival rates. Suggested papers from the area of High Performance Computing are: [76, 85] (Bags-of-Tasks, distributed systems, power awareness), [97] (cluster job start time predictions through traces and simulation) and [47] (workload model for parallel computers). A further investigation of performance bounds, like in [48, 49], is also interesting.

Another field of interest is a systematic comparison of the algorithmic complexity of FMC-QE with different other methodologies.

Also the investigation of further usage domains and case studies is of interest. In the area of dependable systems [99] the integration of performance modeling and performance prediction into the research of dependable systems, called performability, is one possible topic. The prediction of the influence in the performance for a gain in availability is an important question and addressable. Related work in this area is also done in the research of Layered Queueing Networks [59] (Dependable-LQN) where especially the performance modeling of a quorum pattern in [109] is interesting. Another possible area of interest are mathematical/analytical models for production planning in comparison to existing simulation methods. An interesting domain is also the usage of the performance predictions of FMC-QE in the negotiation and guaranteeing of service level agreements (SLAs).

Publications

Journal Articles

Stephan Kluth, Tomasz Porzucek, Flavius Copaciu, Werner Zorn: *Quantitative Modellierung und Analyse mit FMC-QE*, PIK Special Issue on Service-oriented Computing, PIK - Praxis der Informationsverarbeitung und Kommunikation, 2008/4, pages 218-224, Special Issue Editors: Nicolas Repp, Sebastian Hudert, Steffen Bleul, Editor: Hans Meuer, K. G. Saur Verlag, December 2008, ISSN Print 0930-5157, ISSN Online 1865-8342

Conference Contributions

Tomasz Porzucek, Mathias Fritzsche, Stephan Kluth and David Redlich: *Combination of a Discrete Event Simulation and an Analytical Performance Analysis through Model-Transformations*, In Roy Sterrit, Brandon Eames, Jonathan Sprinkle (Ed.): Proceedings of the 17th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2010 - Oxford), 183-192, March 2010, IEEE Computer Society, Los Alamitos, CA, USA, ISBN: 978-0-7695-4005-4

Tomasz Porzucek, Stephan Kluth, Flavius Copaciu and Werner Zorn: *Modeling and Evaluation Framework for FMC-QE*, In: Ted Bapty, Brandon Eames (Ed.): Proceedings of the 16th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2009 - San Francisco), 237-243, IEEE Computer Society, Los Alamitos, CA, USA, April, 2009, ISBN: 978-0-7695-3602-6

Marcel Seelig, Stephan Kluth, Tomasz Porzucek, Flavius Copaciu, Nico Naumann, Steffen Kühn: *Comparison of Simulation and Performance Modeling - A Case Study*, in: David W. Bustard and Roy Sterritt (Ed.): Proceedings of the 15th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008 - Belfast), 49-56, IEEE Computer Society, Los Alamitos, CA, USA, March 2008, ISBN: 978-0-7695-3141-0

Flavius Copaciu, Stephan Kluth, Tomasz Porzucek, Werner Zorn: *Hierarchical Modeling of the Axis2 Web Services Framework with FMC-QE*, In: 3rd International Conference on COMmunication Systems softWAre and middlewaRE (COMSWARE 2008 - Bangalore), 74-81, IEEE Computer Society Press, Los Alamitos, CA, USA, January, 2008, ISBN: 978-1-4244-1796-4

Doctoral Symposia Contributions

Stephan Kluth: *Quantitative Modellierung des Leistungsverhaltens SOA-basierter Systeme mit FMC-QE*, In: Thomas Kühne, Wolfgang Reisig and Friedrich Steimann: Modellierung 2008 (Proceedings of the Modellierung 2008 Doctoral Symposium), 229-232, In Series: Lecture Notes in Informatics (LNI), Volume P-127, Gesellschaft für Informatik e.V. (GI), Bonn, Germany, March 2008, ISBN: 978-3-88579-221-5

Reports

Stephan Kluth: *Spring 2010 Activity Report*, Presented at the Spring 2010 Workshop of the HPI Research School on Service-Oriented Systems Engineering, Hasso-Plattner-Institute for Software Systems Engineering, Potsdam, Germany, April 2010

Stephan Kluth: *Handling of Closed Networks in FMC-QE*, Presented at the Fall 2009 Workshop of the HPI Research School on Service-Oriented Systems Engineering, Hasso-Plattner-Institute for Software Systems Engineering, Döllnsee, Germany, October 2009. In: Christoph Meinel, Hasso Plattner, Jürgen Döllner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann and Holger Giese (edt.): "Proceedings of the 4th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering", Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam, Vol. 31, Universitätsverlag Potsdam, Potsdam, 2010, ISBN: 978-3-86956-036-6

Stephan Kluth: *Spring 2009 Activity Report*, Presented at the Spring 2009 Workshop of the HPI Research School on Service-Oriented Systems Engineering, Hasso-Plattner-Institute for Software Systems Engineering, Potsdam, Germany, April 2009

Stephan Kluth: *FMC-QE - Hierarchies, Transformations and Rules*, Presented at the Fall 2008 Workshop of the HPI Research School on Service-Oriented Systems Engineering, Hasso Plattner Institute for Software Systems Engineering, Potsdam, Germany, October 2008. In: Christoph Meinel, Hasso Plattner, Jürgen Döllner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann and Holger Giese (edt.): "Proceedings of the 3rd Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering", Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam, Vol. 27, Universitätsverlag Potsdam, Potsdam, 2009, ISBN: 978-3-940793-81-2

Stephan Kluth: *FMC-QE - Calculus*, Presented at the Spring 2008 Workshop of the HPI Research School on Service-Oriented Systems Engineering, Hasso-Plattner-Institute for Software Systems Engineering, Potsdam, Germany, April 2008

Stephan Kluth: *FMC-QE - Case Studies*, Presented at the Fall 2007 Workshop of the HPI Research School on Service-Oriented Systems Engineering, Hasso Plattner Institute for Software Systems Engineering, Potsdam, Germany, October 2007. In: Profs. Dres. Christoph Meinel, Andreas Polze, Mathias Weske, Jürgen Döllner, Robert Hirschfeld, Felix Naumann, Holger Giese and Hasso Plattner (edt.): "Proceedings of the 2. Ph.D. retreat of the HPI Research School on Service-oriented Systems Engineering", Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam, Vol. 23, Universitätsverlag Potsdam, Potsdam, 2008, ISBN: 978-3-940793-42-3

Stephan Kluth: *FMC-QE - Positioning, Basic Definitions and Graphical Representation*, Presented at the Spring 2007 Workshop of the HPI Research School on Service-Oriented Systems Engineering, Hasso-Plattner-Institute for Software Systems Engineering, Potsdam, Germany, April 2007

Other Publications

Stephan Kluth: *Quantitative Modeling and Analysis with FMC-QE*, In: Kai Bollue, Dominique Gückel, Ulrich Loup, Jacob Spönemann, Melanie Winkler (Ed.): Dagstuhl 2010: Proceedings of the Joint Workshop of the German Research Training Groups in Computer Science, DFG Research Training Group 1298 AlgoSyn, RWTH Aachen University, Verlagshaus Mainz GmbH, Aachen, 2010, p. 198, ISBN: 3-86130-146-6

Stephan Kluth: *Performance Modeling and Performance Prediction with FMC-QE*, In: Artin Avanes, Dirk Fahland, Joanna Geibig, Siamak Haschemi, Sebastian Heglmeier, Daniel A. Sadilek, Falko Theisselmann, Guido Wachsmuth, Stephan Weißleder (Ed.): Dagstuhl 2009: Proceedings des gemeinsamen Workshops der Informatik-Graduiertenkollegs und Forschungskollegs, Graduiertenkolleg METRIK, Institut für Informatik, Humboldt Universität zu Berlin, GITO mbH - Verlag für Industrielle Informationstechnik und Organisation, Berlin, Germany, June 2009, p. 197-198, ISBN: 978-3-940019-73-8

Stephan Kluth: *Quantitative Modeling and Analysis of Service-Based Systems*, In: Malte Diehl, Henrik Lipskoch, Roland Meyer, Christian Storm (Ed.): Proceedings des gemeinsamen Workshops der Graduiertenkollegs 2008, In: Trustworthy Software Systems, Graduiertenkolleg vertrauenswürdiger Software - Systeme (Trustsoft), University of Oldenburg, GITO mbH - Verlag für Industrielle Informationstechnik und Organisation, Berlin, Germany, May 2008, p. 96, ISBN: 978-3-940019-39-4

Stephan Kluth: *Quantitative Modeling and Analysis of Service-oriented Architectures*, In: Dagstuhl zehn plus eins, Volume 1, Verlagshaus Mainz GmbH, Aachen, Germany, June 2007, p. 198, ISBN: 3-86130-882-7

Bibliography

[1] Rémy Apfelbacher and Anne Rozinat. FMC Notation Reference Sheets. Online, May 2005. URL http://www.fmc-modeling.org.

[2] Rémy Apfelbacher, Andreas Knöpfel, Peter Aschenbrenner, and Sebastian Preetz. *FMC Visualization Guidelines*. Hasso-Plattner-Institute, Potsdam, Germany, January 2005. URL http://www.fmc-modeling.org.

[3] Francois Baccelli, Armand M. Makowski, and Adam Shwartz. The Fork-Join Queue and Related Systems with Synchronization Constraints: Stochastic Ordering and Computable Bounds. *Advances in Applied Probability*, 21(3):629–660, September 1989. ISSN 00018678. URL http://www.jstor.org/stable/1427640.

[4] Gianfranco Balbo, Steven C. Bruell, and Subbarao Ghanta. Combining Queueing Networks and Generalized Stochastic Petri Nets for the Solution of Complex Models of System Behavior. *IEEE Transactions on Computers*, 37(10):1251–1268, October 1988. ISSN 0018-9340. DOI: http://doi.ieeecomputersociety.org/10.1109/12.5986.

[5] Gianfranco Balbo, Matteo Sereno, and Steven C. Bruell. Embedded Processes in Generalized Stochastic Petri Nets. In *Proceedings of the 9th IEEE International Workshop on Petri Nets and Performance Models (PNPM'01)*, Los Alamitos, CA, USA, 2001. IEEE Computer Society. ISBN 0-7695-1248-8. DOI: http://doi.ieeecomputersociety.org/10.1109/PNPM.2001.953357.

[6] Gianfranco Balbo, Steven C. Bruell, and Matteo Sereno. Product Form Solution for Generalized Stochastic Petri Nets. *IEEE Transactions on Software Engineering (TSE)*, 28(10):915–932, October 2002. ISSN 0098-5589. DOI: http://dx.doi.org/10.1109/TSE.2002.1041049.

[7] Simonetta Balsamo and Giuseppe Iazeolla. An Extension of Norton's Theorem for Queueing Networks. *IEEE Transactions on Software Engineering (TSE)*, 8(4):298–305, July 1982. ISSN 0098-5589. DOI: http://doi.ieeecomputersociety.org/10.1109/TSE.1982.235424.

[8] Forest Baskett, K. Mani Chandy, Richard R. Muntz, and Fernando G. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM (JACM)*, 22(2):248–260, 1975. ISSN 0004-5411. DOI: http://doi.acm.org/10.1145/321879.321887.

[9] Falko Bause. Queueing Petri Nets: A Formalism for the Combined Qualitative and Quantitative Analysis of Systems. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models (PNPM93 - Toulouse, France)*, pages 14–23, Los Alamitos, CA, USA, October 1993. IEEE Computer Society. ISBN 0-8186-4250-5. DOI: http://doi.ieeecomputersociety.org/10.1109/PNPM.1993.393439.

[10] Falko Bause and Heinz Beilner. Eine Modellwelt zur Integration von Warteschlangen- und Petri-Netz-Modellen. In Günther Stiege and J. S. Lie, editors, *Proceedings of the 5. GI/ITG-Fachtagung: Messung, Modellierung und Bewertung von Rechensystemen, Braunschweig, 26.-28. September 1989*, volume 218 of *Informatik-Fachberichte*, pages 190–204. Springer, September 1989. ISBN 3-540-51713-8.

[11] Falko Bause and Peter Buchholz. Product Form Queueing Petri Nets: A Combination of Product Form Queueing Networks and Product Form Stochastic Petri Nets. Technical Report 529, Fachbereich Informatik Universität Dortmund, Dortmund, Germany, 1994.

[12] Falko Bause and Peter Buchholz. Aggregation and Disaggregation in Product Form Queueing Petri Nets. In *Proceedings of the 7th IEEE International Workshop on Petri Nets and Performance Models (PNPM'97)*, pages 16–25, Los Alamitos, CA, USA, 1997. IEEE Computer Society. DOI: http://doi.ieeecomputersociety.org/10.1109/PNPM.1997.595533.

[13] Falko Bause and Peter Kemper. Queueing Petri Nets. In E. Schnieder, editor, *Proceedings of the 3. Fachtagung "Entwurf komplexer Automatisierungssystem, Braunschweig (Germany), In: Methoden, Anwendungen und Tools auf Basis von Petri-Netzen*, pages 219–236. Verlag Technische Uni Braunschweig Inst. f. Regelungs- und Automatisierungstechnik, May 1993.

[14] Falko Bause and Pieter S. Kritzinger. *Stochastic Petri Nets*. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig / Wiesbaden, Germany, 2 edition, 2002. ISBN 3-528-15535-3.

[15] Falko Bause, Peter Buchholz, and Peter Kemper. *Quantitative Evaluation of Computing and Communication Systems - 8th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation Performance Tools '95*, volume 977 of *Lecture Notes in Computer Science (LNCS)*, chapter QPN-Tool for the specification and analysis of hierarchically combined Queueing Petri nets, pages 224–238. Springer, Berlin / Heidelberg, Germany, 1995. ISBN 978-3-540-60300-9. DOI: http://dx.doi.org/10.1007/BFb0024318.

[16] Frank-Michael Becker, Gunter Boortz, Volkmar Dietrich, Lutz Engelmann, Christine Ernst, Günter Fanghänel, Hein Höhne, Rudi Lenertat, Günter Liesenberg, Lothar Meyer, Christa Pews-Hocke, Gerd-Dietrich Schmidt, Reinhard Stamm, and Karlheinz Weber. *Formeln und Tabellen für die Sekundarstufen I und II*. Peatec, Gesellschaft für Bildung und Technik mbH, Berlin, Germany, 6 edition, 1996. ISBN 3-89517-253-7.

[17] Gunter Bolch, Georg Fleischmann, and R. Schreppel. Ein funktionales Konzept zur Analyse von Warteschlangennetzen und Optimierung von Leistungsgrößen. In Ulrich Herzog and Martin Paterok, editors, *Messung, Modellierung und Bewertung von Rechensystemen, 4. GI/ITG-Fachtagung, Erlangen, 29. September - 1. Oktober 1987, Proceedings*, volume 154 of *Informatik-Fachberichte*, pages 327–342, Berlin / Heidelberg, Germany, September - October 1987. Springer. ISBN 3-540-18406-6 (Springer Berlin, Heidelberg, New York), 0-387-18406-6 (Springer New York, Berlin, Heidelberg).

[18] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor Shridharbhai Trivedi. *Queueing Networks and Markov Chains : Modeling and Performance Evaluation With Computer Science Applications*. John Wiley & Sons, Inc., 1998. ISBN 0-471-19366-6.

[19] Richard J. Boucherie and Matteo Sereno. A structural characterisation of product form stochastic Petri nets. Technical Report BS-R9402, Department of Operations Reasearch, Statistics, and System Theory, Centrum voor Wiskunde en Informatica, Amsterdam, Amsterdam, The Netherlands, 1994.

[20] Richard J. Boucherie and Matteo Sereno. On Closed Support T-Invariants and the Traffic Equations. *Journal of Applied Probability*, 35(2):473–481, 1998. ISSN 00219002. URL http://www.jstor.org/stable/3215700.

[21] Nikolaos Bourbakis and Anya Tascillo. An SPN-Neural Planning Methodology for Coordination of two Robotic Hands with Constrained. *Journal of Intelligent and Robotic Systems - Springer Netherlands*, 19(3):321–337, July 1997. DOI: http://dx.doi.org/10.1023/A:1007985805475.

[22] Ilja N. Bronstein, Konstantin A. Semendjajew, Gerhard Musiol, and Heiner Muehlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Thun, Frankurt am Main, 2 edition, 1995. ISBN 3-8171-2002-8.

[23] Giacomo Bucci and Enrico Vicario. Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets. *IEEE Transactions on Software Engineering (TSE)*, 21(12):969–992, December 1995. ISSN 0098-5589. DOI: http://doi.ieeecomputersociety.org/10.1109/32.489073.

[24] Peter Buchholz. Hierarchical Structuring of Superposed GSPNs. *IEEE Transactions on Software Engineering (TSE)*, 25(2):166–181, March/April 1999. ISSN 0098-5589. DOI: http://doi.ieeecomputersociety.org/10.1109/32.761443.

[25] *Burroughs B 6700 Handbook - Volume I Hardware (Form No. 5000276)*. Burroughs Cooperation, City of Industry, CA, USA, January 1972.

[26] *Burroughs B 6700 / B 7700 System Software Handbook (Replaces Volume II of Form No. 5000276)*. Burroughs Cooperation, City of Industry, CA, USA, July 1973.

[27] Jeffrey P. Buzen. Fundamental laws of computer system performance. In *SIGMETRICS '76: Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation*, pages 200–210, New York, NY, USA, 1976. ACM. DOI: http://doi.acm.org/10.1145/800200.806196.

[28] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Kluver Academic Publishers, 1999. ISBN 0-7923-8609-4.

[29] K. Mani Chandy and Doug Neuse. Linearizer: A Heuristic Algorithm for Queueing Network Models of Computing Systems. *Communications of the ACM*, 25(2):126–134, February 1982. ISSN 0001-0782. DOI: http://doi.acm.org/10.1145/358396.358403.

[30] K. Mani Chandy, Ulrich Herzog, and Lin S. Woo. Parametric Analysis of Queuing Networks. *IBM Journal of Research and Development*, 19(1):36–42, January 1975.

[31] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, March 1976. ISSN 0362-5915. DOI: http://doi.acm.org/10.1145/320434.320440.

[32] Eran Chinthaka. Web Services and Axis2 Architecture. IBM developerWorks - SOA and Web services - Technical Library, November 2006. URL http://www-128.ibm.com/developerworks/webservices/library/ws-apacheaxis2/.

[33] Giovanni Chiola, G. Bruno, and T. Demaria. Introducing a Color Formalism into Generalized Stochastic Petri Nets. In *Proceedings of the 9th European Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[34] Gianfranco Ciardo and Christoph Lindemann. Analysis of deterministic and stochastic Petri Nets. In *5th International Workshop on Petri Nets and Performance Models*, pages 160–169, Toulouse, France, October 1993. IEEE Computer Society.

[35] J. L. Coleman, William Henderson, and Peter G. Taylor. Product form equilibrium distributions and a convolution algorithm for stochastic Petri nets. *Performance Evaluation*, 26(3):159–180, 1996. ISSN 0166-5316. DOI: http://dx.doi.org/10.1016/0166-5316(95)00023-2.

[36] Flavius Copaciu. A Framework for Adaptive Transport in Service-Oriented Systems based on Performance Prediction. In *Proceedings of the Fall 2006 Workshop of the HPI Research School on Service-Oriented Systems Engineering*, volume 18. Hasso Plattner Institute for Software Systems Engineering, 2006.

[37] Flavius Copaciu, Stephan Kluth, Tomasz Porzucek, and Werner Zorn. Hierarchical Modeling of the Axis2 Web Services Framework with FMC-QE. In *3rd International Conference on COMmunication Systems softWAre and middlewaRE (COMSWARE 2008, Bangalore, India)*, pages 74–81. IEEE Computer Society, January 2008. ISBN 978-1-4244-1796-4. DOI: http://doi.ieeecomputersociety.org/10.1109/COMSWA.2008.4554382.

[38] Pierre-Jacques Courtois. Decomposability, instabilities, and saturation in multiprogramming systems. *Communications of the ACM*, 18(7):371–377, July 1975. ISSN 0001-0782. DOI: http://doi.acm.org/10.1145/360881.360887.

[39] Pierre-Jacques Courtois. Error Analysis in Nearly-Completely Decomposable Stochastic Systems. *Econometrica*, 43(4):691–709, July 1975. ISSN 00129682. URL http://www.jstor.org/stable/1913078.

[40] Pierre-Jacques Courtois. *Decomposabilty, Queueiing and Computer System Applications*. ACM Monograph Serices. Academic Press, Inc., New York, San Francisco, London, 1977. ISBN 0-12-193750-X.

[41] Horst Czichos, editor. *Hütte - Die Grundlagen der Ingenieurwissenschaften*. Akademischer Verein Hütte e.V., Berlin, Springer Verlag, Berlin Heidelberg, 31 edition, 2000. ISBN 3-540-66882-9.

[42] Gero Decker, Volker Gersabeck, Jan Schaffner, and Marcel Seelig. Architecture-Based Performance Simulation. In Sio Iong Ao, Oscar Castillo, Craig Douglas, David Dagan Feng, and Jeong-A. Lee, editors, *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS 2007)*, Lecture Notes in Engineering and Computer Science, pages 1183–1191, Hong Kong, China, March 2007. IMECS, Newswood Limited. ISBN 978-988-98671-4-0, 978-988-98671-7-1.

[43] Peter J. Denning and Jeffrey P. Buzen. The Operational Analysis of Queueing Network Models. *ACM Computing Surveys (CSUR)*, 10(3):225–261, September 1978. ISSN 0360-0300. DOI: http://doi.acm.org/10.1145/356733.356735.

[44] *DIN66200:1992-03 - Betrieb von Rechensystemen - Begriffe, Auftragsbeziehungen*. Deutsches Institut für Normung e.V., March 1992.

[45] Edsger Wybe Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages: NATO Advanced Study Institute*, pages 43–112. Academic Press, 1968.

[46] Susanna Donatelli and Matteo Sereno. On the Product Form Solution for Stochastic Petri Nets. In Kurt Jensen, editor, *Proceedings of the 13th International Conference on Application and Theory of Petri Nets*, number 616 in Lecture Notes in Computer Science, pages 154–172, London, UK, June 1992. Springer. ISBN 3-540-55676-1. DOI: http://dx.doi.org/10.1007/3-540-55676-1_9.

[47] Allen B. Downey. A parallel workload model and its implications for processor allocation. *Cluster Computing*, 1(1):133–145, May 1998. ISSN 1386-7857 (Print) 1573-7543 (Online). DOI: http://dx.doi.org/10.1023/A:1019077214124.

[48] Derek L. Eager and Kenneth C. Sevcik. Performance bound hierarchies for queueing networks. *ACM Transactions on Computer Systems (TOCS)*, 1(2):99–115, 1983. ISSN 0734-2071. DOI: http://doi.acm.org/10.1145/357360.357363.

[49] Derek L. Eager and Kenneth C. Sevcik. Bound hierarchies for multiple-class queuing networks. *Journal of the ACM (JACM)*, 33(1):179–206, January 1986. ISSN 0004-5411. DOI: http://doi.acm.org/10.1145/4904.4992.

[50] Jaliya Ekanayake and Dennis Gannon. Common Architecture for Functional Extensions on Top of Apache Axis2. Technical report, Indiana University Bloomington, 2006.

[51] Muhammad El-Taha. Lecture Notes - Queueing Networks (incompete classnotes). Department of Mathematics and Statistics University of Southern Maine, August 2007.

[52] Gerard Florin and Stéphane Natkin. Generalization of Queueing Network Product Form Solutions to Stochastic Petri Nets. *IEEE Transactions on Software Engineering (TSE)*, 17 (2):99–107, 1991. ISSN 0098-5589. DOI: http://doi.ieeecomputersociety.org/10.1109/32.67591.

[53] Greg Franks and C. Murray Woodside. Performance of Multi-level Client-Server Systems with Parallel Service Operations. In *Proceedings of the First International Workshop on Software and Performance (WOSP98 - Santa Fe, New Mexico, United States)*, pages 120–130, New York, NY, USA, October 1998. ACM. ISBN 1-58113-060-0. DOI: http://doi.acm.org/10.1145/287318.287346.

[54] Greg Franks and C. Murray Woodside. Effectiveness of Early Replies in Client-Server Systems. *Performance Evaluation*, 36-37:165–184, August 1999. ISSN 0166-5316. DOI: http://dx.doi.org/10.1016/S0166-5316(99)00034-6.

[55] Greg Franks and C. Murray Woodside. Multiclass Multiservers with Deferred Operations in Layered Queueing Networks, with Software System Applications. In *Proceedings of the 12th IEEE / ACM Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004)*, pages 239–248, Los Alamitos, CA, USA, 2004. IEEE Computer Society. DOI: http://doi.ieeecomputersociety.org/10.1109/MASCOT.2004.1348262.

[56] Greg Franks, Shikharesh Majumdar, John E. Neilson, Dorina C. Petriu, Jerome A. Rolia, and C. Murray Woodside. Performance Analysis of Distributed Server Systems. In *Proceedings of the Sixth International Conference on Software Quality*, pages 15–26, Ottawa, Canada, October 28-30 1996.

[57] Greg Franks, Peter Maly, Murray Woodside, Dorina C. Petriu, and Alex Hubbard. *Layered Queueing Network Solver and Simulator User Manual*. Department of Systems and

Computer Engineering, Carleton University, Ottawa, Canada, 6840 edition, December 2005.

[58] Greg Franks, Dorina Petriu, Murray Woodside, Jing Xu, and Peter Tregunno. Layered Bottlenecks and Their Mitigation. In *Proceedings of the 3rd International Conference on Quantitative Evaluation of Systems (QEST2006)*, pages 103–114, Los Alamitos, CA, USA, 2006. IEEE Computer Society. ISBN 0-7695-2665-9. DOI: http://doi.ieeecomputersociety.org/10.1109/QEST.2006.23.

[59] Greg Franks, Tariq Al-Omari, C. Murray Woodside, Olivia Das, and Salem Derisavi. Enhanced Modeling and Solution of Layered Queueing Networks. *IEEE Transactions on Software Engineering (TSE)*, 35(2):148–161, 2009. ISSN 0098-5589. DOI: http://doi.ieeecomputersociety.org/10.1109/TSE.2008.74.

[60] Jörn Freiheit and Armin Zimmermann. A Divide and Conquer Approach for the Performance Evaluation of Large Stochastic Petri Nets. In *Proceedings of the 9th IEEE International Workshop on Petri Nets and Performance Models (PNPM'01)*, Los Alamitos, CA, USA, 2001. IEEE Computer Society. DOI: http://doi.ieeecomputersociety.org/10.1109/PNPM.2001.953359.

[61] Mathias Fritzsche and Jendrik Johannes. Putting Performance Engineering into Model-Driven Engineering: Model-Driven Performance Engineering. In Holger Giese, editor, *MoDELS 2007 Workshops in: Models in Software Engineering*, volume 5002/2008 of *Lecture Notes in Computer Science (LNCS)*, pages 164–175, Berlin / Heidelberg, Germany, 2008. Springer. ISBN 978-3-540-69069-6. DOI: http://dx.doi.org/10.1007/978-3-540-69073-3_18.

[62] Mathias Fritzsche, Michael Picht, Wasif Gilani, Ivor Spence, John Brown, and Peter Kilpatrick. Extending BPM Environments of your choice with Performance related Decision Support. In U. Dayal et al., editor, *Business Process Management (BPM2009)*, volume 5701/2009 of *Lecture Notes in Computer Science (LNCS)*, pages 97–112, Berlin / Heidelberg, Germany, 2009. Springer. ISBN 978-3-642-03847-1. DOI: http://dx.doi.org/10.1007/978-3-642-03848-8_8.

[63] Victor M. Glushkov. Automata theory and structural design problems of digital machines. *Cybernetics and Systems Analysis*, 1(1):3–9, January 1965. ISSN 1060-0396 (Print) 1573-8337 (Online). DOI: http://dx.doi.org/10.1007/BF01071436.

[64] William J. Gordon and Gordon F. Newell. Closed queueing systems with exponential servers. *Operations Research*, 15(2):254–265, March - April 1967. ISSN 0030364X. URL http://www.jstor.org/stable/168557.

[65] Baerbel Grimm, Willi Woerstenfeld, Peter Pfeil, and Karlheinz Martin. *Das grosse Tafelwerk*. Volk und Wissen Verlag GmbH, Berlin, Germany, first edition, 1994. ISBN 3-06-000730-6.

[66] Bernhard Gröne. *Konzeptionelle Patterns und ihre Darstellung*. PhD thesis, Hasso Plattner Institut für Softwaresystemtechnik an der Universität Potsdam, Potsdam, Germany, August 2004.

[67] Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc., New York, NY, USA, 3rd edition, 1998. ISBN 0471170836.

[68] Martin Haas and Werner Zorn. *Methodische Leistungsanalyse von Rechensystemen*. R. Oldenbourg Verlag GmbH, München, Germany / Vienna, Austria, 1995. ISBN 3-486-20779-2.

[69] Serge Haddad and Patrice Moreaux. Evaluation of High Level Petri nets by Means of Aggregation and Decomposition. In *Proceedings of the 6th IEEE International Workshop on Petri Nets and Performance Models (PNPM'95)*, pages 11–20, Los Alamitos, CA, USA, 1995. IEEE Computer Society. DOI: http://doi.ieeecomputersociety.org/10.1109/PNPM.1995.524311.

[70] Serge Haddad, Patrice Moreaux, Matteo Sereno, and Manuel Silva. Structural Characterization and Qualitative Properties of Product Form Stochastic Petri Nets. In J.-M. Colom and M. Koutny, editors, *Applications and Theory of Petri Nets 2001 (ICATPN 2001)*, volume 2075/2001 of *Lecture Notes in Computer Science (LNCS)*, pages 164–183, Berlin / Heidelberg, Germany, 2001. Springer. ISBN 978-3-540-42252-5. DOI: http://dx.doi.org/10.1007/3-540-45740-2_11.

[71] Philip Heidelberger and Kishor S. Trivedi. Analytic Queueing Models for Programs with Internal Concurrency. *IEEE Transactions on Computers*, 32(1):73–82, January 1983. ISSN 0018-9340. DOI: http://doi.ieeecomputersociety.org/10.1109/TC.1983.1676125.

[72] William Henderson and Peter G. Taylor. Embedded Processes in Stochastic Petri Nets. *IEEE Transactions on Software Engineering (TSE)*, 17(2):108–116, 1991. ISSN 0098-5589. DOI: http://doi.ieeecomputersociety.org/10.1109/32.67592.

[73] William Henderson, D. Lucic, and Peter G. Taylor. A Net Level Performance Analysis of Stochastic Petri Nets. *J. Australian Math. Soc. Series B*, 31(2):176–187, 1989.

[74] Reinhard Höllerer. *Modellierung und Optimierung von Bürgerdiensten am Beispiel der Stadt Landshut*. PhD thesis, Hasso-Plattner-Institute at the University of Potsdam, Potsdam, Germany, June 2009. Draft Version.

[75] Anatol Holt and Frederic Commoner. Events and Conditions. In Jack B. Dennis, editor, *Record of the Project MAC conference on concurrent systems and parallel computation*, pages 3–52, New York, NY, USA, 1970. ACM.

[76] Alexandru Iosup, Ozan Sonmez, Shanny Anoep, and Dick Epema. The performance of bags-of-tasks in large-scale distributed systems. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 97–108, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-997-5. DOI: http://doi.acm.org/10.1145/1383422.1383435.

[77] James R. Jackson. Networks of waiting lines. *Operations Reseach*, 5(4):518–521, August 1957. ISSN 0030364X. URL http://www.jstor.org/stable/167249.

[78] James R. Jackson. Jobshop-like Queuing Systems. *Management Science*, 10(1):131–142, October 1963. ISSN 00251909. URL http://www.jstor.org/stable/2627213.

[79] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, New York, NY, USA, 1991. ISBN 0471503363.

[80] Kurt Jensen. *Coloured Petri Nets, Volume 1, Basic Concepts, Analysis Methods and Practical Use*. Springer, Berlin, Germany, 2nd edition, Februar 1997. ISBN 3-540-60943-1.

[81] Don H. Johnson. Origins of the equivalent circuit concept: the voltage-source equivalent. *Proceedings of the IEEE*, 91(4):636– 640, April 2003. ISSN 0018-9219. DOI: http://doi.ieeecomputersociety.org/10.1109/JPROC.2003.811716.

[82] Frank Keller. *Über die Rolle von Architekturbeschreibungen im Software-Entwicklungsprozess*. PhD thesis, Hasso Plattner Institut für Softwaresystemtechnik an der Universität Potsdam, Potsdam, Germany, August 2003.

[83] David George Kendall. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 24(3):338–354, September 1953. ISSN 00034851. URL http://www.jstor.org/stable/2236285.

[84] Cheeha Kim and Ashok K. Agrawala. Analysis of the Fork-Join Queue. *IEEE Transactions on Computers*, 38(2):250–255, February 1989. ISSN 0018-9340. DOI: http://doi.ieeecomputersociety.org/10.1109/12.16501.

[85] Kyong Hoon Kim, Rajkumar Buyya, and Jong Kim. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID '07 -Rio De Janeiro)*, pages 541–548, Washington, DC, USA, May 14-17 2007. IEEE Computer Society. ISBN 0-7695-2833-3. DOI: http://doi.ieeecomputersociety.org/10.1109/CCGRID.2007.85.

[86] Matthias Kirschnick. The Performance Evaluation and Prediction SYstem for Queueing NetworkS PEPSY-QNS. Technical Report TR-I4-18-94, Computer Science Department Operating Systems - IMMD IV, Friedrich-Alexander-University , Erlangen-Nürnberg, Erlangen, Germany, June 1994.

[87] Leonard Kleinrock. *Communication Nets: Stochastic Message Flow and Delay*. Dover Publications, New York, NY, USA, 1973. ISBN 0-486-61105-1.

[88] Leonard Kleinrock. *Queueing Systems Volume I: Theory*. John Wiley & Sons, New York, NY, USA, 1975. ISBN 0-471-49110-1.

[89] Leonard Kleinrock. *Queueing Systems Volume II: Computer Applications*. John Wiley & Sons, New York, NY, USA, 1976. ISBN 0-471-49111-X.

[90] Stephan Kluth. Analyse und Modellierung des ERP-Systems Sage Office Line. Master's thesis, Hasso Plattner Institute for Software Systems Engineering, Potsdam, Germany, November 2005.

[91] Andreas Knöpfel. *Konzepte der Beschreibung interaktiver Systeme*. PhD thesis, Hasso Plattner Institut für Softwaresystemtechnik an der Universität Potsdam, Potsdam, Germany, August 2004.

[92] Andreas Knöpfel, Bernhard Gröne, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. John Wiley & Sons, März 2006. ISBN 0-470-02710-X.

[93] Stephen S. Lavenberg and Martin Reiser. Stationary State Probabilities at Arrival Instants for Closed Queueing Networks with Multiple Types of Customers. *Journal of Applied Probability*, 17(4):1048–1061, December 1980. ISSN 00219002. URL http://www.jstor.org/stable/3213214.

[94] Aurel A. Lazar and Thomas G. Robertazzi. Markovian Petri Net protocols with product form solution. *Performance Evaluation*, 12(1):67–77, January 1991. DOI: http://dx.doi.org/10.1016/0166-5316(91)90016-V.

[95] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, Februar 1984. ISBN 0137469756.

[96] Abigail S. Lebrecht and William J. Knottenbelt. Response Time Approximations in Fork-Join Queues. In *Proceedings of the 23rd Annual UK Performance Engineering Workshop (UKPEW)*, Edge Hill University, Ormskirk, Lancashire, UK, June 2007.

[97] Hui Li, David Groep, Jeff Templon, and Lex Wolters. Predicting job start times on clusters. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid (CCGRID '04 - Chicago, IL)*, pages 301–308, Los Alamitos, CA, USA, 2004. IEEE Computer Society. ISBN 0-7803-8430-X. DOI: http://doi.ieeecomputersociety.org/10.1109/CCGrid.2004.1336581.

[98] John D. C. Little. A Proof of the Queueing Formula $L = \lambda * W$. *Operations Research*, 9(3):383–387, May - June 1961. ISSN 0030364X. URL http://www.jstor.org/stable/167570.

[99] Miroslaw Malek, Bratislav Milic, and Nikola Milanovic. Analytical Availability Assessment of IT Services. In T. Nanya et al., editor, *Service Availability - Proceedings of the 5th International Service Availability Symposium (ISAS 2008 Tokyo, Japan)*, volume 5017/2008 of *Lecture Notes in Computer Science*, pages 207–224, Berlin / Heidelberg, Germany, May 2008. Springer. ISBN 978-3-540-68128-1. DOI: http://dx.doi.org/10.1007/978-3-540-68129-8_16.

[100] Manish Malhotra and Kishor S. Trivedi. A methodology for formal expression of hierarchy in model solution. In *Proceedings on the 5th International Workshop on Petri Nets and Performance Models*, pages 258–267, Toulouse, France, October 1993. IEEE Computer Society. DOI: http://doi.ieeecomputersociety.org/10.1109/PNPM.1993.393445.

[101] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS)*, 2(2):93–122, 1984. ISSN 0734-2071. DOI: http://doi.acm.org/10.1145/190.191.

[102] Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley & Sons, Inc., New York, NY, USA, 1995. ISBN 0-471-93059-8.

[103] Hans Ferdinand Mayer. Über das Ersatzschema der Verstärkerröhre [On equivalent circuits for electronic amplifiers]. *Telegraphen- und Fernsprech-Technik*, 15:335–337, 1926.

[104] Michael Karl Molloy. *On the integration of delay and throughput measures in distributed processing models*. PhD thesis, University of California, Los Angeles, 1981.

[105] Tadao Murata. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77-4, pages 541–580. IEEE Computer Society, April 1989. DOI: http://doi.ieeecomputersociety.org/10.1109/5.24143.

[106] Stéphane Natkin. *Les Reseaux de Petri Stochastiques et leur Application a L'evaluation des Systemes Informatiques*. PhD thesis, Le Conservatoire national des arts et métiers (Cnam), Paris, France, 1980.

[107] Edward Lawry Norton. Design of finite networks for uniform frequency characteristic. Technical Report TM26-0-1860, Bell Laboratories, 1926.

[108] Tariq Omari, Greg Franks, C. Murray Woodside, and Amy Pan. Solving Layered Queueing Networks of Large Client Server Systems with Symmetric Replication. In *Proceedings of the 5th International Workshop on Software and Performance (WOSP 2005)*, pages 159–166, New York, NY, USA, July 2005. ACM. ISBN 1-59593-087-6. DOI: http://doi.acm.org/10.1145/1071021.1071038.

[109] Tariq Omari, Salem Derisavi, Greg Franks, and C. Murray Woodside. Performance Modeling of a Quorum Pattern in Layered Service Systems. In *Proceedings of the 4th International Conference on Quantitative Evaluation of SysTems (QEST2007 - Edinburgh)*, pages 201–210, Los Alamitos, CA, USA, September 2007. IEEE Computer Society. ISBN 0-7695-2883-X. DOI: http://doi.ieeecomputersociety.org/10.1109/QEST.2007.25.

[110] Tariq Omari, Greg Franks, C. Murray Woodside, and Amy Pan. Efficient performance models for layered server systems with replicated servers and parallel behavior. *Journal of Systems and Software*, 80(4):510–527, April 2007. ISSN 0164-1212. DOI: http://dx.doi.org/10.1016/j.jss.2006.07.022.

[111] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, New York, NY, USA, 4 edition, 2002. ISBN 0-07-366011-6.

[112] Srinath Perera, Chathura Herath, Jaliya Ekanayake, Eran Chinthaka, Ajith Ranabahu, Deepal Jayasinghe, Sanjiva Weerawarana, and Glen Daniels. Axis2, Middleware for Next Generation Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 833–840, Washington, DC, USA, September 2006. IEEE Computer Society. DOI: http://doi.ieeecomputersociety.org/10.1109/ICWS.2006.36.

[113] Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962.

[114] Brigitte Plateau and Jean-Michel Fourneau. A methodology for solving Markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12(4):370–387, 1991. ISSN 0743-7315. DOI: http://dx.doi.org/10.1016/0743-7315(91)90007-V.

[115] Tomasz Porzucek, Stephan Kluth, Flavius Copaciu, and Werner Zorn. Modeling and Evaluation Framework for FMC-QE. In *Proceedings of the 16th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS2008)*, pages 237–243, Los Alamitos, CA, USA, April 2009. IEEE Computer Society. ISBN 978-0-7695-3602-6. DOI: http://doi.ieeecomputersociety.org/10.1109/ECBS.2009.28.

[116] Tomasz Porzucek, Mathias Fritzsche, Stephan Kluth, and David Redlich. Combination of a Discrete Event Simulation and an Analytical Performance Analysis through Model-Transformations. In *Proceedings of the 17th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS2010)*, pages 183–192, Los Alamitos, CA, USA, March 2010. IEEE Computer Society. ISBN 978-0-7695-4005-4. DOI: http://doi.ieeecomputersociety.org/10.1109/ECBS.2010.26.

[117] Balaji Prabhakar, Nicholas Bambos, and T. S. Mountford. The Synchronization of Poisson Processes and Queueing Networks with Service and Synchronization Nodes. *Advances in Applied Probability*, 32(3):824–843, September 2000. ISSN 00018678. URL http://www.jstor.org/stable/1428415.

[118] Martin Reiser and Stephen S. Lavenberg. Mean-Value Analysis of Closed Multichain Queuing Networks. *Journal of the ACM (JACM)*, 27(2):313–322, 1980. ISSN 0004-5411. DOI: http://doi.acm.org/10.1145/322186.322195.

[119] Thomas G. Robertazzi. Why most stochastic Petri nets are non-product form networks. Technical report, Stony Brook, N.Y.: State University of New York at Stony Brook, College of Engineering, New York, NY, USA, 1991. URL http://hdl.handle.net/1951/37249.

[120] Marcel Seelig, Stephan Kluth, Flavius Copaciu, Tomasz Porzucek, Nico Naumann, and Steffen Kühn. Comparison of Performance Modeling and Simulation - a Case Study. In David W. Bustard and Roy Sterritt, editors, *Proceedings of the 15th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2008 - Belfast, UK)*, pages 49–56, Los Alamitos, CA, USA, March 2008. IEEE Computer Society. DOI: http://doi.ieeecomputersociety.org/10.1109/ECBS.2008.47.

[121] Kenneth C. Sevcik and Isi Mitrani. The Distribution of Queuing Network States at Input and Output Instants. *Journal of the ACM (JACM)*, 28(2):358–371, 1981. ISSN 0004-5411. DOI: http://doi.acm.org/10.1145/322248.322257.

[122] Herbert A. Simon and Albert Ando. Aggregation of Variables in Dynamic Systems. *Econometrica*, 29(2):111–138, April 1961. ISSN 00129682. URL http://www.jstor.org/stable/1909285.

[123] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, USA, 1994. ISBN 0-691-03699-3.

[124] Peter Tabeling. *Softwaresysteme und ihre Modellierung*. Springer, Berlin / Heidelberg, Germany, 2006. ISBN 3-540-25828-0.

[125] Phuoc Tran-Gia. *Einführung in die Leistungsbewertung und Verkehrstheorie*. Oldenbourg Wissenschaftsverlag GmbH, München, Germany, 2 edition, Oktober 2005. ISBN 3-486-57882-0.

[126] Hendrik Vantilborgh. Exact Aggregation in Exponential Queueing Networks. *Journal of the ACM (JACM)*, 25(4):620–629, 1978. ISSN 0004-5411. DOI: http://doi.acm.org/10.1145/322092.322102.

[127] J. Walrand. A Note on Norton's Theorem for Queuing Networks. *Journal of Applied Probability*, 20(2):442–444, June 1983. ISSN 00219002. URL http://www.jstor.org/stable/3213821.

[128] Jiacum Wang. *Timed Petri Nets - Theory and Application*. Kluwer Academic Publishers, Boston / Dordrecht / London, 1998. ISBN 0-7923-8270-6.

[129] Siegfried Wendt. Eine Methode zum Enfwurf komplexer Schaltwerke unter Verwendung speziller Ablaufdiagramme. *Elektronische Rechenanlagen*, 12(6):314–323, December 1970.

[130] Siegfried Wendt. *Nichtphysikalische Grundlagen der Informationstechnik. Interpretierte Formalismen*. Springer, Berlin, Germany, 2 edition, 1991. ISBN 3-540-54452-6.

[131] Siegfried Wendt. Operationszustand versus Steuerzustand - eine äußerst zweckmäßige Unterscheidung. Technical report, University of Kaiserslautern, Kaiserslautern, Germany, Februar 1998.

[132] Andreas Willig. *Lecture Notes - Performance Evaluation Techniques*. Hasso-Plattner-Institute, University of Potsdam, Potsdam, Germany, April 2005.

[133] Armin Zimmermann, Reinhard German, Jörn Freiheit, and Günther Hommel. TimeNET 3.0 Tool Description. In *Proceedings of the International Conference on Petri Nets and Performance Models (PNPM'99)*, 1999.

[134] Armin Zimmermann, Jörn Freiheit, Reinhard German, and Günter Hommel. Petri Net Modelling and Performability Evaluation with TimeNET 3.0. In *Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS '00)*, pages 188–202, London, UK, 2000. Springer. ISBN 3-540-67260-5.

[135] Werner Zorn. Kommunikationssysteme - durch Abstraktion zum Durchblick, am Beispiel eines der meistgenutzen Protokolle. Inaugural Lecture at the Hasso-Plattner-Institue at the University of Potsdam, October 2002.

[136] Werner Zorn. A Different Approach to Network Modelling. Presentation, March 2003.

[137] Werner Zorn. Distinguishing between Control States and Operational States - a well Proven Paradigm to Cope with the Complexity of Discrete Dynamic Systems. Hasso-Plattner-Institute, University of Potsdam, 2005.

[138] Werner Zorn. *Lecture Slides: Quantitative Modeling*. Hasso-Plattner-Institute, University of Potsdam, Potsdam, Germany, 2005.

[139] Werner Zorn. *Lecture Slides: Communication Systems I & II*. Hasso-Plattner-Institute, University of Potsdam, Potsdam, Germany, 2005-2006.

[140] Werner Zorn. Quantitative Modeling as a Special Abstraction of Systems Modeling. Submitted to the 3rd International Conference on the Quantitative Evaluation of SysTems (QEST2006), 2006.

[141] Werner Zorn. Hierarchical Modeling based on Service Requests. Hasso-Plattner-Institute, University of Potsdam, 2007.

[142] Werner Zorn. Calculus Paper. Internal, status 20070313, March 2007.

[143] Werner Zorn. FMC-QE - A New Approach in Quantitative Modeling. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Modeling, Simulation and Visualization Methods (MSV 2007) within WorldComp '07*, pages 280 – 287, Las Vegas, NV, USA, June 2007. CSREA Press. ISBN 1-60132-029-9.

[144] Werner Zorn. Hierarchische Modellierung basierend auf Bedienanforderungen. Presented at the 21.DFN- Arbeitstagung über Kommunikationsnetze, Technical University Kaiserslautern, Germany, May 2007.

[145] Werner Zorn. *Lecture Slides: Communication Systems I*. Hasso-Plattner-Institute, University of Potsdam, Potsdam, Germany, 2007.

[146] Werner Zorn. FMC-QE - Introduction with Examples. Presentation, April 2007. Internal Presentation at the Research Group of Prof. Dr.-Ing. Werner Zorn at the Hasso-Plattner-Institute at the University of Potsdam.

[147] Werner Zorn. Hierarchische Modellierung auf Basis von Bedienanforderungen. Presented at the Institut für Operations Research, Humboldt-Universität zu Berlin, April 2007.

Glossary

A	Arrival rate with deterministic inter-arrival times
B	Service Rate, deterministic
$BSSt$	Basic server station
D	Throughput
f	Desired bottleneck utilization
$HSSt$	Hierarchical server station
K	Queue-capacity (including service requests in service)
K	Overall number of service Requests in a Closed System - also n_{ges}
λ	Arrival rate with stochastic inter-arrival times
$\lambda^{[1]}$	Arrival rate of top level ([1]) service requests
$\lambda_{bott}^{[1]}$	Saturation arrival rate of bottleneck (Unit: [Top level Service requests / Timeunit])
$\lambda_i^{[bb]}$	Arrival rate of service request SRq_i on hierarchy level $[bb]$ (Unit: $[SRq_i/Timeunit]$)
m	Number of parallel servers (multiplicity)
M	Population
μ	Service Rate, stochastic
$m_i^{[bb]}$	Absolute multiplicity of the server of service request i on hierarchy level $[bb]$
$m_{i,mpx}^{[bb]}$	Multiplex coefficient of logical server i
$m_{parent(i)}^{[bb-1]}$	Absolute multiplicity of the server of on the next hierarchical level ($[bb-1]$)
$m_{i,int}^{[bb]}$	Relative multiplicity of the server of service request i on hierarchy level $[bb]$ to the next hierarchical level $[bb-1]$
m_j	Number of parallel (Multiplex-)Servers
$\mu_i^{[bb]}$	Service rate of the server of service request i on hierarchy level $[bb]$
$N_i^{[bb]}$	Normalized service request i on hierarchical level $[bb]$

n	Mean number of service requests in the station
N	Overall number of service stations in the network
$N_i^{e[bb]}$	Normalized unity service request i on hierarchical level $[bb]$
n_{ges}	Overall number of circulating service requests
$n_i^{[bb]}$	Mean number of service request i on hierarchy level $[bb]$
$N_{i,q}^{[bb]}$	Queued normalized service request i on hierarchical level $[bb]$
$N_i^{r[bb]}$	Normalized service response i on hierarchical level $[bb]$
n_q	Mean number of queued service requests
$n_{i,q}^{[bb]}$	Mean number of queued service request i on hierarchy level $[bb]$
$N^{e,r[bb]}$	Normalized unity service response i on hierarchical level $[bb]$
$N_{i,s}^{[bb]}$	Normalized service request i on hierarchical level $[bb]$ in service
n_s	Mean number of service requests in service
$n_{i,s}^{[bb]}$	Mean number of service request i on hierarchy level $[bb]$ in service
ρ	Utilization, stochastic
R	Response time
$\rho_i^{[bb]}$	Utilization of the server of queued service request i on hierarchy level $[bb]$
$R_i^{[bb]}$	Response time of the server of the service request i on hierarchy level $[bb]$
$Server_i$	Corresponding (Multiplex-)Server of service request i
$Server_j$	Non-ambiguous name of a (Multiplex-)Server
$SRq_i^{[bb]}$	Unnormalized service request i on hierarchical level $[bb]$
$SRq_i^{[bb]}$	Non-ambiguous name of the service request i on hierarchy level $[bb]$
$SRs_i^{[bb]}$	Unnormalized service response i on hierarchical level $[bb]$
U	Utilization, deterministic
v	Absolute traffic flow coefficient
v_{ext}	External traffic flow coefficient
v_{int}	Internal traffic flow coefficient
$v_i^{[bb]}$	Absolute traffic flow coefficient of service request i on hierarchy level $[bb]$
$v_{parent(i)}^{[bb-1]}$	Absolute traffic flow coefficient on the next hierarchical level ($[bb-1]$) relative to service request $i^{[bb]}$

$v_{i,int}^{[bb]}$	Relative traffic flow coefficient of service request i on hierarchy level $[bb]$ to the next hierarchical level $[bb+1]$
W	Queue time
$W_i^{[bb]}$	Waiting time for service request i on hierarchy level $[bb]$
X	Service time
$X_i^{[bb]}$	Service time for service request i on hierarchy level $[bb]$
Y	Service duration
$Y_i^{[bb]}$	Service duration for service request i on hierarchy level $[bb]$

Index

Appendix A

Server Performance Values

Table A.1: D/D/1

D/D/1 - Single Server, Deterministic Service Time

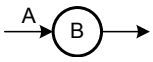
Model:	Arrival Rate [68]:
	A
	Service Time [68]:
	X
	Service Rate [68]:
	$B = \frac{1}{X}$
	Utilization:
	$U = \frac{A}{B} \qquad U \leq 1$
Exp. Number of Queued SRqs:	$n_q = 0$
Exp. Number of SRqs in Service:	$n_s = \frac{A}{B}$
Exp. Number of SRqs in the Station:	$n = n_s = \frac{A}{B}$
Exp. Waiting Time:	$W = 0$
Exp. Response Time:	$R = \frac{1}{B}$

Table A.2: D/D/m

D/D/m - Parallel Server, Deterministic Service Times

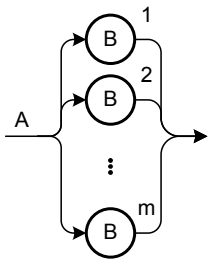
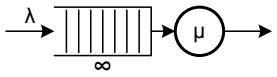
Model:	Arrival Rate [68]:
	A
	Service Time [68]:
	X
	Service Rate:
	$B = \frac{1}{mX}$
	Utilization:
	$U = \frac{A}{B} \qquad U \leq 1$
Exp. Number of Queued SRqs:	$n_q = 0$
Exp. Number of SRqs in Service:	$n_s = m\frac{A}{B}$
Exp. Number of SRqs in the Station:	$n = n_s = m\frac{A}{B}$
Exp. Waiting Time:	$W = 0$
Exp. Response Time:	$R = \frac{1}{B}$

Table A.3: M/M/1

M/M/1 - Single Server, Exponential Distributed Service Time

Model:



Arrival Rate [88]:

$\lambda_k = \lambda \qquad k = 0, 1, 2, ..$

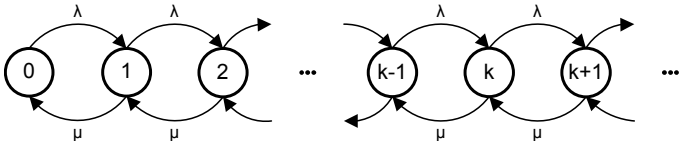
Service Rate [88]:

$\mu_k = \mu \qquad k = 0, 1, 2, ..$

Utilization [67]:

$\rho = \frac{\lambda}{\mu} < 1$

Markov Chain [88]:



State Probabilities [67, 87–89]: $p_k = (1 - \rho)\,\rho^k \qquad k = 0, 1, 2, ..$

Exp. Number of Queued SRqs [67, 79]: $n_q = \frac{\rho^2}{1-\rho}$

Exp. Number of SRqs in Service: $n_s = \rho$

Exp. Number of SRqs in the Station [79, 87, 88]: $n = \frac{\rho}{1-\rho}$

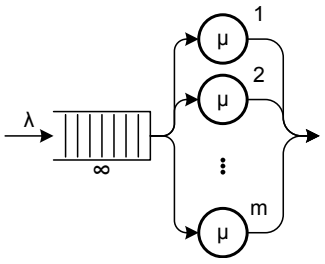
Exp. Waiting Time [67, 88]: $W = \frac{n_q}{\lambda} = \frac{\rho^2}{\lambda(1-\rho)} = \frac{\rho}{\mu-\lambda}$

Exp. Response Time [67, 79, 88]: $R = \frac{n}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu-\lambda}$

Table A.4: M/M/m

M/M/m - Parallel Server, Exponential Distributed Service Times

Model:



Arrival Rate [88]:

$$\lambda_k = \lambda \qquad k = 0, 1, 2, ..$$

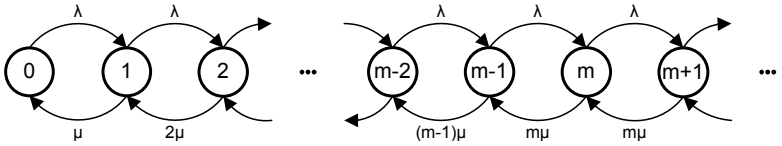
Service Rate [67]:

$$\mu_k = \begin{cases} k\mu & 1 \leq k < m \\ m\mu & m \leq k \end{cases}$$

Utilization [67, 88, 125]:

$$\rho = \frac{\lambda}{m\mu} < 1$$

Markov Chain [88]:



State Probabilities [88]:

$$p_0 = \frac{1}{\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right)}$$

$$p_k = \begin{cases} p_0 \frac{(m\rho)^k}{k!} & 1 \leq k \leq m \\ p_0 \frac{\rho^k m^m}{m!} & k \geq m \end{cases}$$

Exp. Number of Queued SRqs [67, 125]: $n_q = \frac{\frac{\lambda}{\mu}^m \rho}{m!(1-\rho)^2} p_0$

Exp. Number of SRqs in Service [125]: $n_s = m\rho = \frac{\lambda}{\mu}$

Exp. Number of SRqs in the Station [67]: $n = m\rho + \frac{\frac{\lambda}{\mu}^m}{m!(1-\rho)^2} p_0$

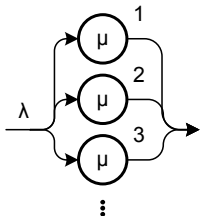
Exp. Waiting Time [67, 125]: $W = \frac{n_q}{\lambda} = \frac{\frac{\lambda}{\mu}^m}{m!(m\mu)(1-\rho)^2} p_0$

Exp. Response Time [67]: $R = \frac{n_q}{\lambda} + \frac{n_s}{\lambda} = \frac{\frac{\lambda}{\mu}^m}{m!(m\mu)(1-\rho)^2} p_0 + \frac{1}{\mu}$

Table A.5: M/M/∞

M/M/∞ - Infinite Server, Exponential Distributed Service Times

Model:



Arrival Rate [88]:

$\lambda_k = \lambda \qquad k = 0, 1, 2, ..$

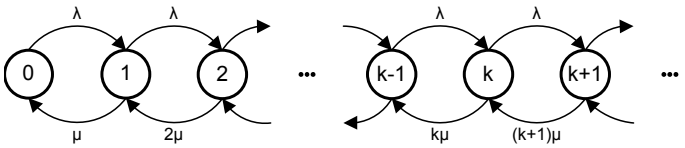
Service Rate[88]:

$\mu_k = k\mu \qquad k = 1, 2, 3, ..$

Utilization:

-

Markov Chain [88]:



State Probabilities [88]: $p_k = \frac{(\lambda/\mu)^k}{k!} e^{-\lambda/\mu} \qquad k = 0, 1, 2, ..$

Exp. Number of Queued SRqs: $n_q = 0$

Exp. Number of SRqs in Service: $n_s = \frac{\lambda}{\mu}$

Exp. Number of SRqs in the Station [88]: $n = \frac{\lambda}{\mu}$

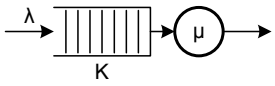
Exp. Waiting Time: $W = 0$

Exp. Response Time [88]: $R = \frac{1}{\mu}$

Table A.6: M/M/1/K

M/M/1/K - Single Server, Exponential Distributed Service Time, Finite Storage

Model:



Arrival Rate [88]:

$$\lambda_k = \begin{cases} \lambda & k < K \\ 0 & k \geq K \end{cases}$$

Effective Arrival Rate [67]:

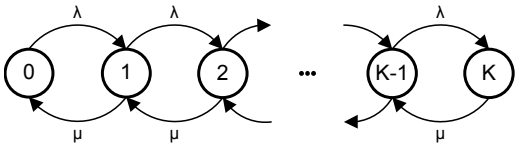
$$\lambda_{eff} = \lambda\left(1 - p_K\right) = \begin{cases} \lambda\left(1 - \frac{1-\rho}{\frac{1}{\rho^K}-\rho}\right) & \rho \neq 1 \\ \lambda\left(1 - \frac{1}{K+1}\right) & \rho = 1 \end{cases}$$

Service Rate [88]:

$$\mu_k = \mu \qquad k = 1, 2, .., K$$

Traffic Intensity [67]: $\rho = \frac{\lambda}{\mu}$

Markov Chain [88]:



State Probabilities [18, 67, 79, 88]:

$$p_0 = \begin{cases} \frac{1-\rho}{1-\rho^{K+1}} & \rho \neq 1 \\ \frac{1}{K+1} & \rho = 1 \end{cases}$$

$$p_k = \begin{cases} \frac{(1-\rho)\rho^k}{1-\rho^{K+1}} & 0 \leq k \leq K; \rho \neq 1 \\ \frac{1}{K+1} & 0 \leq k \leq K; \rho = 1 \\ 0 & else \end{cases}$$

Exp. Number of Queued SRqs [67, 79]:

$$n_q = \begin{cases} \frac{\rho}{1-\rho} - \frac{\rho\left(K\rho^K+1\right)}{1-\rho^{K+1}} & \rho \neq 1 \\ \frac{K(K-1)}{2(K+1)} & \rho = 1 \end{cases}$$

Exp. Number of SRqs in Service [67]:

$$n_s = \left(1 - p_0\right) = \begin{cases} 1 - \frac{1-\rho}{1-\rho^{K+1}} & \rho \neq 1 \\ 1 - \frac{1}{K+1} & \rho = 1 \end{cases}$$

Exp. Number of SRqs in the Station [18, 67, 79]:

$$n = \begin{cases} \frac{\rho}{1-\rho} - \frac{K+1}{1-\rho^{K+1}}\rho^{K+1} & \rho \neq 1 \\ \frac{K}{2} & \rho = 1 \end{cases}$$

Exp. Waiting Time [67]:

$$W = \frac{n_q}{\lambda_{eff}} = \begin{cases} \frac{\frac{1}{\rho^{K-2}} - K\rho + \rho^2(K-1)}{\lambda(1-\rho)^2} & \rho \neq 1 \\ \frac{K-1}{2\lambda} & \rho = 1 \end{cases}$$

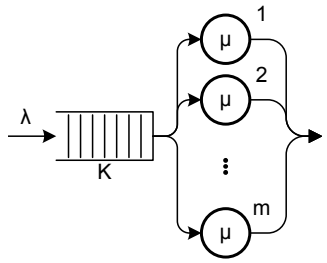
Exp. Response Time [67]:

$$R = \frac{n}{\lambda_{eff}} = \frac{n}{\lambda(1-p_K)}$$

Table A.7: M/M/m/K

M/M/m/K - Parallel Server, Exponential Distributed Service Times, Finite Storage

Model:



Arrival Rate [67]:

$$\lambda_k = \begin{cases} \lambda & k < K \\ 0 & k \geq K \end{cases}$$

Effective Arrival Rate [67]:

$$\lambda_{eff} = \lambda \left(1 - p_K\right)$$

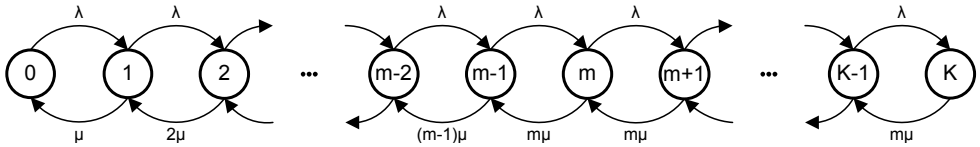
Service Rate [88]:

$$\mu_k = \begin{cases} k\mu & 0 \leq k \leq m \\ m\mu & m \leq k \end{cases}$$

Traffic Intensity [67, 79]:

$$\rho = \frac{\lambda}{m\mu}$$

Markov Chain [79]:



State Probabilities [67, 79]:

$$p_0 = \begin{cases} \left(\sum_{k=0}^{m-1} \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} + \frac{\left(\frac{\lambda}{\mu}\right)^m}{m!} \frac{1-\rho^{K-m+1}}{1-\rho} \right)^{-1} & \rho \neq 1 \\ \left(\sum_{k=0}^{m-1} \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} + \frac{\left(\frac{\lambda}{\mu}\right)^m}{m!} \left(K - n + 1\right) \right)^{-1} & \rho = 1 \end{cases}$$

$$p_k = \begin{cases} \frac{\lambda^k}{k!\mu^k} p_0 & 1 \leq k \leq m \\ \frac{\lambda^k}{m^{k-m} m! \mu^k} p_0 & c \leq k \leq K \end{cases}$$

Exp. Number of Queued SRqs [51, 67]:

$$n_q = \begin{cases} \frac{p_0 \left(\frac{\lambda}{\mu}\right)^m \rho}{m!(1-\rho)^2} \left(1 - \rho^{K-m+1} - (1-\rho)\left(K-m+1\right)\rho^{K-m}\right) & \rho \neq 1 \\ m^{m-1} \frac{(K-m)(K-m+1)}{2} & \rho = 1 \end{cases}$$

Exp. Number of SRqs in Service [67]: $n_s = \frac{\lambda_{eff}}{\mu} = \frac{\lambda(1-p_K)}{\mu}$

Exp. Number of SRqs in the Station [67]: $n = n_q + \frac{\lambda_{eff}}{\mu} = n_q + \frac{\lambda(1-p_K)}{\mu}$

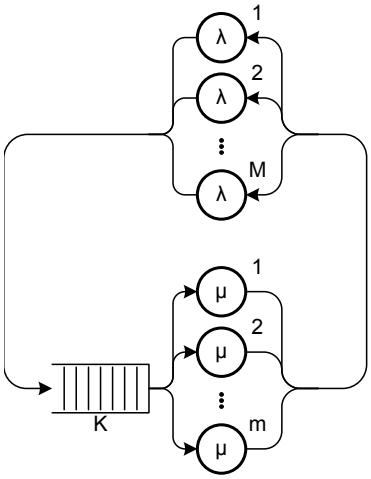
Exp. Waiting Time [67, 79]: $W = R - \frac{1}{\mu} = \frac{n_q}{\lambda_{eff}}$

Exp. Response Time [67, 79]: $R = \frac{n}{\lambda_{eff}} = \frac{n}{\lambda(1-p_K)}$

Table A.8: M/M/m/K/M

M/M/m/K/M - Parallel Server, Exp. Dist. Service Times, Finite Storage, Finite Population

Model:



Population[67, 88]:

M

Arrival Rate [67, 88]:

$$\lambda_k = \begin{cases} \lambda\,(M-k) & 0 \le k \le K-1 \\ 0 & else \end{cases}$$

Effective Arrival Rate [67]:

$$\lambda_{eff} = \sum_{k=0}^{M-1} (M-n)\,\lambda p_k = \lambda\,(M-n)$$

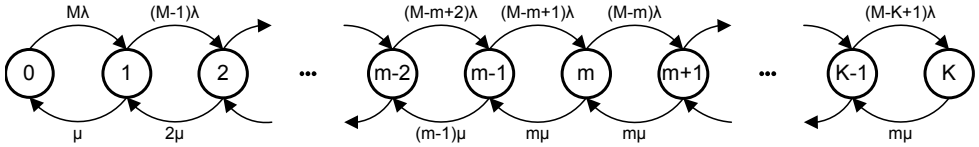
Service Rate[67, 88]:

$$\mu_k = \begin{cases} k\mu & 0 \le k \le m \\ m\mu & k \ge m \end{cases}$$

Traffic Intensity:

$$\rho = \frac{\lambda}{m\mu}$$

Markov Chain [88]:



State Probabilities [28, 88]:

$$p_0 = \frac{1}{1+\sum_{k=1}^{m-1}\binom{M}{k}\left(\frac{\lambda}{\mu}\right)^k + \binom{M}{m-1}\left(\frac{\lambda}{\mu}\right)^{m-1}\sum_{k=m}^{K}\frac{(M-m+1)!}{(M-k)!}\left(\frac{\lambda}{\mu m}\right)^{k-m+1}}$$

$$p_k = \begin{cases} p_0\left(\frac{\lambda}{\mu}\right)^k\binom{M}{k} & 0 \le k \le m-1 \\ p_0\left(\frac{\lambda}{\mu}\right)^k\binom{M}{k}\frac{k!}{m!}m^{m-k} & m \le k \le K \end{cases}$$

Exp. Number of Queued SRqs [67]: $n_q = n - \frac{\lambda_{eff}}{\mu} = n - \frac{\lambda}{\mu}\,(M-n)$

Exp. Number of SRqs in Service [67]: $n_s = \frac{\lambda_{eff}}{\mu} = \frac{\lambda}{\mu}\,(M-n)$

Exp. Number of SRqs in the Station [67]: $n = \sum_{k=1}^{M} k p_k$

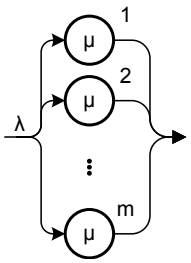
Exp. Waiting Time [67]: $W = \frac{n_q}{\lambda\left(M-n_q\right)}$

Exp. Response Time [67]: $R = \frac{n}{\lambda(M-n)}$

Table A.9: M/M/m/m

M/M/m/m - Parallel Server, Exponential Distributed Service Times, no Storage

Model:



Arrival Rate [88]:

$$\lambda_k = \begin{cases} \lambda & k < m \\ 0 & k \geq m \end{cases}$$

Effective Arrival Rate [67]:

$$\lambda_{eff} = \lambda \left(1 - p_m\right)$$

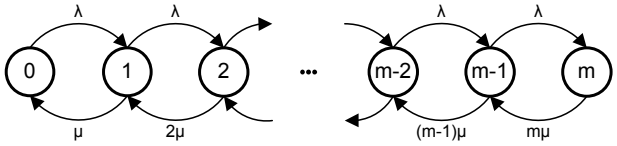
Service Rate[88]:

$$\mu_k = \begin{cases} k\mu & k \leq m \\ 0 & k > m \end{cases}$$

Traffic Intensity:

$$\rho = \frac{\lambda}{m\mu}$$

Markov Chain [88]:



State Probabilities [67, 88]:

$$p_0 = \left(\sum_{k=0}^{m} \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!} \right)^{-1}$$

$$p_k = \begin{cases} p_0 \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!} & k \leq m \\ 0 & k > m \end{cases}$$

Exp. Number of Queued SRqs:

$$n_q = 0$$

Exp. Number of SRqs in Service [125]:

$$n_s = \frac{\lambda_{eff}}{\mu} = \frac{\lambda(1-p_m)}{\mu}$$

Exp. Number of SRqs in the Station [125]:

$$n = n_s = \frac{\lambda_{eff}}{\mu} = \frac{\lambda(1-p_m)}{\mu}$$

Exp. Waiting Time:

$$W = 0$$

Exp. Response Time [125]:

$$R = \frac{1}{\mu}$$

Erlang's loss formula [67, 79, 88, 125]:

$$p_m = \frac{\frac{\left(\frac{\lambda}{\mu}\right)^m}{m!}}{\sum_{k=0}^{m} \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!}}$$

Table A.10: Server Performance Values - Overview

	D/D/1	D/D/m	M/M/1	M/M/m	M/M/∞	M/M/1/K	M/M/m/K	M/M/m/K/K	M/M/m/m
Model									
Arrival Rate	A	A	$\lambda_k = \lambda$ $k = 0,1,2,..$	$\lambda_k = \lambda \quad k = 0,1,2,..$	$\lambda_k = \lambda$ $k = 0,1,2,..$	$\lambda_k = \begin{cases} \lambda & k < K \\ 0 & k \geq K \end{cases}$	$\lambda_k = \begin{cases} \lambda & k < K \\ 0 & k \geq K \end{cases}$	$\lambda_k = \begin{cases} \lambda (M-k) & 0 \leq k \leq K-1 \\ 0 & else \end{cases}$	$\lambda_k = \begin{cases} \lambda & k < m \\ 0 & k \geq m \end{cases}$
Effective Arrival Rate						$\lambda_{eff} = \lambda (1 - p_K)$ $= \begin{cases} \lambda \left(1 - \frac{1-\rho}{1-\rho^{K+1}}\right) & \rho \neq 1 \\ \lambda \left(1 - \frac{1}{K+1}\right) & \rho = 1 \end{cases}$	$\lambda_{eff} = \lambda (1 - p_K)$	$\lambda_{eff} = \sum_{k=0}^{M-1} (M-n)\, \lambda p_k = \lambda (M-n)$	$\lambda_{eff} = \lambda (1 - p_m)$
Service Rate	$B = \frac{1}{X}$	$B = \frac{1}{mX}$	$\mu_k = \mu$ $k = 0,1,2,..$	$\mu_k = \begin{cases} k\mu & 1 \leq k < m \\ m\mu & m \leq k \end{cases}$	$\mu_k = k\mu$ $k = 1,2,3,..$	$\mu_k = \mu \quad k = 1,2,..,K$	$\mu_k = \begin{cases} k\mu & 0 \leq k \leq m \\ m\mu & m \leq k \end{cases}$	$\mu_k = \begin{cases} k\mu & 0 \leq k \leq m \\ m\mu & k \geq m \end{cases}$	$\mu_k = \begin{cases} k\mu & k \leq m \\ 0 & k > m \end{cases}$
Utilization	$U = \frac{A}{B} \leq 1$	$U = \frac{A}{B} \leq 1$	$\rho = \frac{\lambda}{\mu} < 1$	$\rho = \frac{\lambda}{m\mu} < 1$					
Traffic Intensity						$\rho = \frac{\lambda}{\mu}$	$\rho = \frac{\lambda}{m\mu}$	$\rho = \frac{\lambda}{m\mu}$	$\rho = \frac{\lambda}{m\mu}$
State Probabilities			$p_k = (1-\rho)\,\rho^k$ $k = 0,1,2,..$	$p_0 = \frac{1}{\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right)}$ $p_k = \begin{cases} p_0 \frac{(m\rho)^k}{k!} & 1 \leq k \leq m \\ p_0 \frac{\rho^k m^m}{m!} & k \geq m \end{cases}$	$p_k = \frac{(\lambda/\mu)^k}{k!} e^{-\lambda/\mu}$ $k = 0,1,2,..$	$p_0 = \begin{cases} \frac{1-\rho}{1-\rho^{K+1}} & \rho \neq 1 \\ \frac{1}{K+1} & \rho = 1 \end{cases}$ $p_k = \begin{cases} \frac{(1-\rho)\rho^k}{1-\rho^{K+1}} & 0 \leq k \leq K, \rho \neq 1 \\ \frac{1}{K+1} & 0 \leq k \leq K, \rho = 1 \\ 0 & else \end{cases}$	$p_0 = \frac{1}{\sum_{k=0}^{m-1} \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} + \frac{\left(\frac{\lambda}{\mu}\right)^m}{m!} \frac{1-\rho^{K-m+1}}{1-\rho}} \quad \rho \neq 1$ $p_0 = \frac{1}{\sum_{k=0}^{m-1} \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} + \frac{\left(\frac{\lambda}{\mu}\right)^m}{m!} (K-m+1)} \quad \rho = 1$ $p_k = \begin{cases} \frac{\lambda^k}{k!\mu^k} p_0 & 1 \leq k \leq m \\ \frac{\lambda^k}{m^{k-m} m! \mu^k} p_0 & c \leq k \leq K \end{cases}$	$p_0 = \frac{1}{1 + \sum_{k=1}^{m-1} \binom{M}{k} \left(\frac{\lambda}{\mu}\right)^k + \binom{M}{m} \left(\frac{\lambda}{\mu}\right)^{m-1} \sum_{k=m}^{K} \frac{(M-m+1)!}{(M-k)!} \left(\frac{\lambda}{m\mu}\right)^{k-m+1}}$ $p_k = \begin{cases} p_0 \binom{M}{k} \frac{k!}{\mu^k} & 0 \leq k \leq m-1 \\ p_0 \binom{M}{k} \frac{k!}{\mu^k} m^{m-k} & m \leq k \leq K \end{cases}$	$p_0 = \left(\sum_{k=0}^{m} \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!} \right)^{-1}$ $p_k = \begin{cases} p_0 \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!} & k \leq m \\ 0 & k > m \end{cases}$
Exp. Number of Queued SRqs	$n_q = 0$	$n_q = 0$	$n_q = \frac{\rho^2}{1-\rho}$	$n_q = \frac{\lambda^m}{m!(1-\rho)^2} p_0$	$n_q = 0$	$n_q = \begin{cases} \frac{\rho}{1-\rho} - \frac{\rho(K\rho^{K+1})}{1-\rho^{K+1}} & \rho \neq 1 \\ \frac{K(K-1)}{2(K+1)} & \rho = 1 \end{cases}$	$n_q = \begin{cases} \frac{p_0 \left(\frac{\lambda}{\mu}\right)^m \rho}{m!(1-\rho)^2} \left(1 - \rho^{K-m+1}\right) \\ - (1-\rho)(K-m+1)\rho^{K-m} \end{cases} \quad \rho \neq 1$ $n_q = \frac{(K-m)(K-m+1)}{2} \quad \rho = 1$	$n_q = n - \frac{\lambda_{eff}}{\mu} = n - \frac{\lambda}{\mu}(M-n)$	$n_q = 0$
Exp. Number of SRqs in Service	$n_s = \frac{A}{B}$	$n_s = m\frac{A}{B}$	$n_s = \rho$	$n_s = m\rho = \frac{\lambda}{\mu}$	$n_s = \frac{\lambda}{\mu}$	$n_s = \begin{cases} 1 - \frac{1-\rho}{1-\rho^{K+1}} & \rho \neq 1 \\ 1 - \frac{K}{K+1} & \rho = 1 \end{cases}$	$n_s = \frac{\lambda(1-p_K)}{\mu}$	$n_s = \frac{\lambda}{\mu}(M-n)$	$n_s = \frac{\lambda(1-p_m)}{\mu}$
Exp. Number of SRqs in Station	$n = \frac{A}{B}$	$n = m\frac{A}{B}$	$n = \frac{\rho}{1-\rho}$	$n = m\rho + \frac{\lambda^m}{m!(1-\rho)^2} p_0$	$n = \frac{\lambda}{\mu}$	$n = \begin{cases} \frac{\rho}{1-\rho} - \frac{K+1}{1-\rho^{K+1}} \rho^{K+1} & \rho \neq 1 \\ \frac{K}{2} & \rho = 1 \end{cases}$	$n = n_q + \frac{\lambda(1-p_K)}{\mu}$	$n = \sum_{k=1}^{M} k p_k$	$n = \frac{\lambda(1-p_m)}{\mu}$
Exp. Waiting Time	$W = 0$	$W = 0$	$W = \frac{\rho}{\mu - \lambda}$	$W = \frac{\lambda^m}{m!(m\mu)(1-\rho)^2} p_0$	$W = 0$	$W = \begin{cases} \frac{\rho^2 - K\rho + \rho^2(K-1)}{\lambda(1-\rho)^2} & \rho \neq 1 \\ \frac{K-1}{2\lambda} & \rho = 1 \end{cases}$	$W = \frac{n_q}{\lambda_{eff}}$	$W = \frac{n_q}{\lambda(M-n_q)}$	$W = 0$
Exp. Response Time	$R = \frac{1}{B}$	$R = \frac{1}{B}$	$R = \frac{1}{\mu - \lambda}$	$R = \frac{\lambda^m}{m!(m\mu)(1-\rho)^2} p_0 + \frac{1}{\mu}$	$R = \frac{1}{\mu}$	$R = \frac{n}{\lambda(1-p_K)}$	$R = \frac{n}{\lambda(1-p_K)}$	$R = \frac{n}{\lambda(M-n)}$	$R = \frac{1}{\mu}$
Erlang's Loss Formula									$p_m = \frac{\frac{(\frac{\lambda}{\mu})^m}{m!}}{\sum_{k=0}^{m} \frac{(\frac{\lambda}{\mu})^k}{k!}}$

Appendix B

Tables

Table B.1: Tableau Example (Table 3.2)

Experimental Parameters	
$n_{ges}^{[1]}$	30
$\lambda_{bott}^{[1]}$	3,750
f	0,600
$\lambda^{[1]}$	2,250

Service Request Section									Server Section					Dynamic Evaluation Section									
[bb]	i	$SRq_i^{[bb]}$	$p_{Pi(j),j}$	$v_{Pi(j)}^{[bb-1]}$	$v_{j,int}^{[bb]}$	$v_j^{[bb]}$	$\lambda_j^{[bb]}$	$Server_j^{[bb]}$	$m_{Pi(j)}^{[bb-1]}$	$m_{j,int}^{[bb]}$	$m_j^{[bb]}$	Mpx_j	$X_j^{[bb]}$	$m_{j,mpx}^{[bb]}$	$\mu_j^{[bb]}$	$\rho_j^{[bb]}$	$n_{j,q}^{[bb]}$	$W_j^{[bb]}$	$n_{j,s}^{[bb]}$	$Y_j^{[bb]}$	$n_j^{[bb]}$	$R_j^{[bb]}$
2	1	Cash	1,00	1,00	1,00	1,00	2,250	Cashier	1	1	1	1	0,056	1,000	18,000	0,125	0,018	0,008	0,125	0,056	0,143	0,063
3	2	Blow-Dry	0,50	1,00	1,00	0,50	1,125	Blow-Dryer	1	∞	∞	4	0,350	1,000	2,857		0,000	0,000	0,394	0,350	0,394	0,350
2	3	Dry	1,00	1,00	1,00	1,00	2,250	Dryer	1	1	1				∞		0,000	0,000	0,394	0,175	0,394	0,175
3	4	Perm	0,30	1,00	1,00	0,30	0,675	Permer	1	1	1	3	0,350	0,467	1,333	0,506	0,519	0,769	0,506	0,750	1,025	1,519
3	5	Dye	0,20	1,00	1,00	0,20	0,450	Dyer	1	2	2	2	0,333	0,125	0,375	0,600	0,675	1,500	1,200	2,667	1,875	4,167
2	6	Supp. Work	1,00	1,00	1,00	1,00	2,250	Supp.	1	1	1				3,750		1,194	0,531	1,706	0,758	2,900	1,289
3	7	Cut2	0,60	1,00	1,00	0,60	1,350	Cut2-Cutter	1	1	1	2	0,333	0,750	2,250	0,600	0,900	0,667	0,600	0,444	1,500	1,111
3	8	Cut1	0,40	1,00	1,00	0,40	0,900	Cut1-Cutter	1	1	1	3	0,300	0,533	1,778	0,506	0,519	0,577	0,506	0,563	1,025	1,139
2	9	Cut	1,00	1,00	1,00	1,00	2,250	Cutter	1	1	1				3,750		1,419	0,631	1,106	0,492	2,525	1,122
2	10	Wash	1,00	1,00	1,00	1,00	2,250	Washer	1	3	3	1	0,111	1,000	9,000	0,083	0,000	0,000	0,250	0,111	0,250	0,111
1	11	Barbershop Serv.	1,00	1,00	1,00	1,00	2,250	Server	1	1	1				3,750		2,631	1,169	3,581	1,592	6,212	2,761
1	12	Job Generation	1,00	1,00	1,00	1,00	2,250	Generator	1	1	1		10,572		0,095		0,000	0,000	23,788	10,572	23,788	10,572

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Appentice	5	0,167
2	Barber	1	0,267
3	Barber Boss	1	0,225
4	Customer	∞	

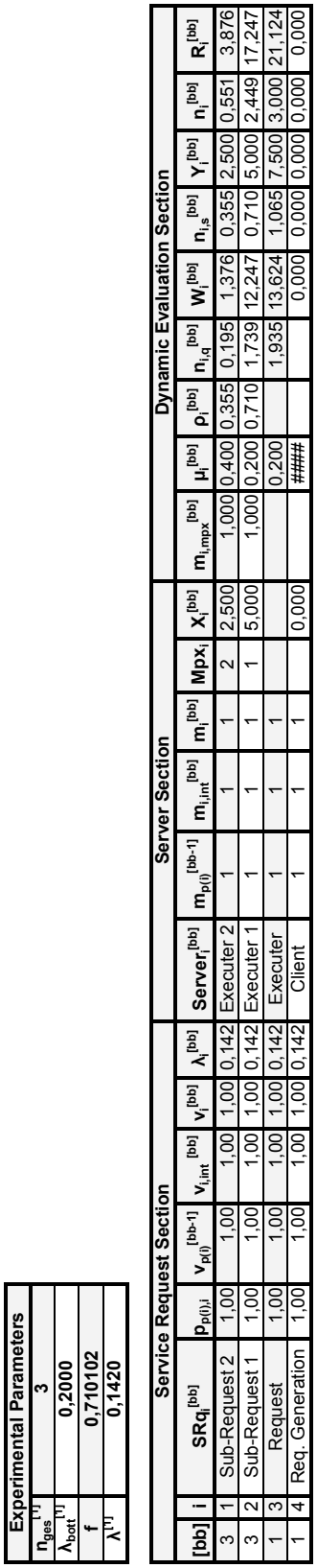
Table B.2: Open Queueing Example - Tableau (Table 3.8)

Experimental Parameters	
n_{ges} [1]	80
λ_{bott} [1]	5,0000
f	0,9000
λ [1]	4,5000

Service Request Section								Server Section						Dynamic Evaluation Section									
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	γ_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	γ_i [bb]	n_i [bb]	R_i [bb]
5	1	Saving Request	0,50	5,00	1,00	2,50	11,250	Request Saver	1	1	1	3	0,060	1,000	16,667	0,675	1,402	0,125	0,675	0,060	2,077	0,185
5	2	Printing Request	0,50	5,00	1,00	2,50	11,250	Request Printer	1	1	1	2	0,030	1,000	33,333	0,338	0,172	0,015	0,338	0,030	0,509	0,045
4	3	Persist Data Request	1,00	5,00	1,00	5,00	22,500	Data Persister	1	1	1				33,333		1,574	0,070	1,013	0,045	2,586	0,115
4	4	Computation Request	1,00	5,00	1,00	5,00	22,500	Request Computer	1	1	1	1	0,040	1,000	25,000	0,900	8,100	0,360	0,900	0,040	9,000	0,400
3	5	Unreliable Execution	1,00	1,00	5,00	5,00	22,500	Unreliable Executer	1	1	1				25,000		9,674	0,430	1,913	0,085	11,586	0,515
2	5	Reliable Execution	1,00	1,00	1,00	1,00	4,500	Reliable Executer	1	1	1				5,000		9,674	2,150	1,913	0,425	11,586	2,575
2	6	Input	1,00	1,00	1,00	1,00	4,500	Input Retriever	1	1	1	4	0,050	1,000	20,000	0,225	0,065	0,015	0,225	0,050	0,290	0,065
1	7	Request	1,00	1,00	1,00	1,00	4,500	Request Handler	1	1	1				5,000		9,739	2,164	2,138	0,475	11,877	2,639
1	8	Request Generation	1,00	1,00	1,00	1,00	4,500	Client	1	1	1		15,139		0,066		0,000	0,000	68,123	15,139	68,123	15,139

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	CPU	1	0,200
2	Printer	1	0,075
3	Disk	1	0,150
4	I/O-Device	1	0,050

Table B.3: Closed Tandem Network - M/M/1 Tableau (Table 4.1)

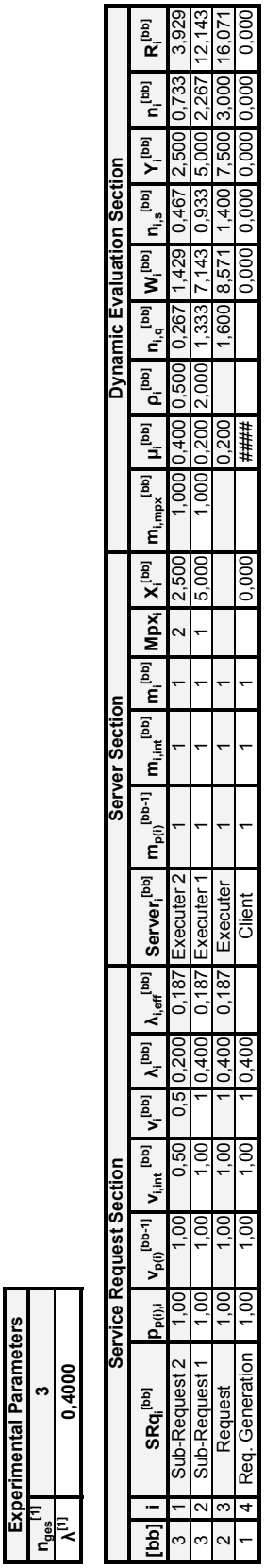


Experimental Parameters	
n_{ges} [1]	3
λ_{bott} [1]	0,2000
f	0,710102
λ [1]	0,1420

Service Request Section									Server Section					Dynamic Evaluation Section									
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
3	1	Sub-Request 2	1,00	1,00	1,00	1,00	0,142	Executer 2	1	1	1	2	2,500	1,000	0,400	0,355	0,195	1,376	0,355	2,500	0,551	3,876
3	2	Sub-Request 1	1,00	1,00	1,00	1,00	0,142	Executer 1	1	1	1	1	5,000	1,000	0,200	0,710	1,739	12,247	0,710	5,000	2,449	17,247
1	3	Request	1,00	1,00	1,00	1,00	0,142	Executer	1	1	1				0,200		1,935	13,624	1,065	7,500	3,000	21,124
1	4	Req. Generation	1,00	1,00	1,00	1,00	0,142	Client	1	1	1		0,000		####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	Server 1	1	2,500
2	Server 2	1	5,000

Table B.4: Closed Tandem Network - M/M/1/K Tableau (Table 4.2)



Experimental Parameters

n_{ges} [1]	3
λ [1]	0,4000

Service Request Section

[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$\lambda_{i,eff}$ [bb]
3	1	Sub-Request 2	1,00	1,00	0,50	0,5	0,200	0,187
3	2	Sub-Request 1	1,00	1,00	1,00	1	0,400	0,187
2	3	Request	1,00	1,00	1,00	1	0,400	0,187
1	4	Req. Generation	1,00	1,00	1,00	1	0,400	

Server Section

$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]
Executer 2	1	1	1	2	2,500
Executer 1	1	1	1	1	5,000
Executer	1	1	1		
Client	1	1	1		0,000

Dynamic Evaluation Section

$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
1,000	0,400	0,500	0,267	1,429	0,467	2,500	0,733	3,929
1,000	0,200	2,000	1,333	7,143	0,933	5,000	2,267	12,143
	0,200		1,600	8,571	1,400	7,500	3,000	16,071
	#####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section

j	$Name_j$	m_j	X_j [1]
1	Server 1	1	5,000
2	Server 2	1	1,250

Table B.5: Closed Tandem Network - Summation Method Tableau (Table 4.3)

Experimental Parameters	
n_{ges} [1]	3
λ_{bat} [1]	0,2000
f	0,9144
λ [1]	0,1829

Service Request Section								Server Section					Dynamic Evaluation Section									
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
3	1	Sub-Request 2	1,00	1,00	1,00	1,00	0,183	Executer 2	1	1	1	2	2,500	1,000	0,400	0,457	0,200	1,096	0,457	2,500	0,658	3,596
3	2	Sub-Request 1	1,00	1,00	1,00	1,00	0,183	Executer 1	1	1	1	1	5,000	1,000	0,200	0,914	1,428	7,808	0,914	5,000	2,342	12,808
1	3	Request	1,00	1,00	1,00	1,00	0,183	Executer	1	1	1				0,200		1,628	8,904	1,372	7,500	3,000	16,404
1	4	Req. Generation	1,00	1,00	1,00	1,00	0,183	Client	1	1	1		0,000		####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	Server 1	1	2,500
2	Server 2	1	5,000

Table B.6: Closed Tandem Network - Tableaux - Comparison

(a) M/M/1

Experimental Parameters	
n_{ges} [1]	3
λ_{bott} [1]	0,2000
f	0,710102
λ [1]	0,1420

		Service Request Section						Server Section					Dynamic Evaluation Section										
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
3	1	Sub-Request 2	1,00	1,00	1,00	1,00	0,142	Executer 2	1	1	1	2	2,500	1,000	0,400	0,355	0,195	1,376	0,355	2,500	0,551	3,876
3	2	Sub-Request 1	1,00	1,00	1,00	1,00	0,142	Executer 1	1	1	1	1	5,000	1,000	0,200	0,710	1,739	12,247	0,710	5,000	2,449	17,247
1	3	Request	1,00	1,00	1,00	1,00	0,142	Executer	1	1	1			0,200			1,935	13,624	1,065	7,500	3,000	21,124
1	4	Req. Generation	1,00	1,00	1,00	1,00	0,142	Client	1	1	1		0,000		#####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	Server 1	1	2,500
2	Server 2	1	5,000

Experimental Parameters	
n_{ges} [1]	3
λ [1]	0,4000

(b) M/M/1/K

		Service Request Section						Server Section					Dynamic Evaluation Section										
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$\lambda_{i,eff}$ [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
3	1	Sub-Request 2	1,00	1,00	0,50	0,5	0,200	0,187	Executer 2	1	1	1	2	2,500	1,000	0,400	0,500	0,267	1,429	0,467	2,500	0,733	3,929
3	2	Sub-Request 1	1,00	1,00	1,00	1	0,400	0,187	Executer 1	1	1	1	1	5,000	1,000	0,200	2,000	1,333	7,143	0,933	5,000	2,267	12,143
2	3	Request	1,00	1,00	1,00	1	0,400	0,187	Executer	1	1	1	1			0,200		1,600	8,571	1,400	7,500	3,000	16,071
1	4	Req. Generation	1,00	1,00	1,00	1	0,400		Client	1	1	1		0,000		#####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	Server 1	1	5,000
2	Server 2	1	1,250

Experimental Parameters	
n_{ges} [1]	3
λ_{bott} [1]	0,2000
f	0,9144
λ [1]	0,1829

(c) Summation Method

		Service Request Section						Server Section					Dynamic Evaluation Section										
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
3	1	Sub-Request 2	1,00	1,00	1,00	1,00	0,183	Executer 2	1	1	1	2	2,500	1,000	0,400	0,457	0,200	1,096	0,457	2,500	0,658	3,596
3	2	Sub-Request 1	1,00	1,00	1,00	1,00	0,183	Executer 1	1	1	1	1	5,000	1,000	0,200	0,914	1,428	7,808	0,914	5,000	2,342	12,808
1	3	Request	1,00	1,00	1,00	1,00	0,183	Executer	1	1	1			0,200			1,628	8,904	1,372	7,500	3,000	16,404
1	4	Req. Generation	1,00	1,00	1,00	1,00	0,183	Client	1	1	1		0,000		#####			0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	Server 1	1	2,500
2	Server 2	1	5,000

Table B.7: Closed Central Server Example - Tableau (Table 4.5)

Experimental Parameters	
n_{ges} [1]	3
λ_{bott} [1]	0,0500
f	0,6895
λ [1]	0,0345

Service Request Section								Server Section						Dynamic Evaluation Section								
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
4	1	Write Request 2	0,56	9,00	1,00	5,00	0,172	Disk2 Writer	1	1	1	3	4,000	1,000	0,250	0,690	0,587	3,403	0,690	4,000	1,276	7,403
4	2	Write Request 1	0,44	9,00	1,00	4,00	0,138	Disk1 Writer	1	1	1	2	2,500	1,000	0,400	0,345	0,103	0,746	0,345	2,500	0,448	3,246
3	3	Data Writing Request	0,90	10,00	1,00	9,00	0,310	Data Writer	1	1	1				0,450		0,690	2,222	1,034	3,333	1,724	5,556
3	4	Calculation Request	1,00	10,00	1,00	10,00	0,345	Calculator	1	1	1	1	2,000	1,000	0,500	0,690	0,587	1,702	0,690	2,000	1,276	3,702
2	5	Internal Request	1,00	1,00	10,00	10,00	0,345	Internal Exec.	1	1	1				0,500		1,276	3,702	1,724	5,000	3,000	8,702
1	6	Transaction	1,00	1,00	1,00	1,00	0,034	Trans. Exec.	1	1	1				0,050		1,966	57,016	1,724	50,000	3,000	87,016
1	7	Transaction Generation	1,00	1,00	1,00	1,00	0,034	Client	1	1	1		0,000		####		0,000	0,000	0,000	0,000	0,000	0,000

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	CPU	1	20,000
2	Disk1	1	20,000
3	Disk2	1	10,000

Table B.8: Semaphore Synchronization - Tableau (Table 4.8)

Experimental Parameters	
$n_{ges}^{[1]}$	1
$\lambda_{bott}^{[1]}$	1,0000
f	0,8800
$\lambda^{[1]}$	0,8800

			Service Request Section						Server Section					Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{P(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
2	1	Critical Action 1	1,00	1,00	1,00	1,00	0,880	CA 1 Executer	1	1	1	1	0,100	0,733	7,333	0,120	0,000	0,000	0,120	0,136	0,120	0,136
1	2	Request 1	1,00	1,00	1,00	1,00	0,880	Executer 1	1	1	1				7,333		0,000	0,000	0,120	0,136	0,120	0,136
1	3	Req. Generation 1	1,00	1,00	1,00	1,00	0,880	Client 1	1	1	1		1,000	1,000	1,000	0,880	0,000	0,000	0,880	1,000	0,880	1,000

Experimental Parameters	
$n_{ges}^{[1]}$	1
$\lambda_{bott}^{[1]}$	2,0000
f	0,7073
$\lambda^{[1]}$	1,4146

			Service Request Section						Server Section					Dynamic Evaluation Section								
[bb]	i	$SRq_i^{[bb]}$	$p_{P(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx_i	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$W_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
2	1	Critical Action 2	1,00	1,00	1,00	1,00	1,415	CA 2 Executer	1	1	1	1	0,200	0,967	4,833	0,293	0,000	0,000	0,293	0,207	0,293	0,207
1	2	Request 2	1,00	1,00	1,00	1,00	1,415	Executer 2	1	1	1				4,833		0,000	0,000	0,293	0,207	0,293	0,207
1	3	Req. Generation 2	1,00	1,00	1,00	1,00	1,415	Client 2	1	1	1		0,500	1,000	2,000	0,707	0,000	0,000	0,707	0,500	0,707	0,500

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	Critical Resource	1	0,300

Table B.9: Multiclass Example - Tableau (Table 4.6)

Experimental Parameters	
n_{ges}	30
λ_{bott}	0,9434
f	0,8000
λ_A	0,7547

		Service Request Section						Server Section					Dynamic Evaluation Section									
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
2	1	Service Request A.2	1,00	1,00	2,00	2,00	1,509	Req. A.2 Srv.	1	1	1	2	0,330	0,623	1,887	0,800	3,200	2,120	0,800	0,530	4,000	2,650
2	2	Service Request A.1	1,00	1,00	1,00	1,00	0,755	Req. A.1 Srv.	1	1	1	1	0,200	0,500	2,500	0,302	0,131	0,173	0,302	0,400	0,432	0,573
1	3	Service Request A	1,00	1,00	1,00	1,00	0,755	Req. A.2.2 Srv.	1	1	1				1,887		3,331	4,413	1,102	1,460	4,432	5,873
1	4	Generate Request A	1,00	1,00	1,00	1,00	0,755	Client A	1	1	1		33,877		0,030			0,000	25,568	33,877	25,568	33,877

Experimental Parameters	
n_{ges}	30
λ_{bott}	0,9434
f	0,5000
λ_B	0,4717

		Service Request Section						Server Section					Dynamic Evaluation Section									
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
3	5	Service Request B.2.2	1,00	2,00	2,00	4,00	1,887	Req. B.2.2 Srv.	1	1	1	3	0,100	1,000	10,000	0,189	0,044	0,023	0,189	0,100	0,233	0,123
3	6	Service Request B.2.1	1,00	2,00	1,00	2,00	0,943	Req. B.2.1 Srv.	1	1	1	2	0,200	0,377	1,887	0,500	0,500	0,530	0,500	0,530	1,000	1,060
2	7	Service Request B.2	1,00	1,00	2,00	2,00	0,943	Req. B.2 Srv.	1	1	1				1,887		0,544	0,577	0,689	0,730	1,233	1,307
2	8	Service Request B.1	1,00	1,00	1,00	1,00	0,472	Req. B.1 Srv.	1	1	1	1	0,200	0,500	2,500	0,189	0,044	0,093	0,189	0,400	0,233	0,493
1	9	Service Request B	1,00	1,00	1,00	1,00	0,472	Req. B Srv.	1	1	1				1,887		0,588	1,246	0,877	1,860	1,465	3,106
1	10	Generate Request B	1,00	1,00	1,00	1,00	0,472	Client B	1	1	1		60,494		0,017			0,000	28,535	60,494	28,535	60,494

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	Server X	1	0,400
2	Server Y	1	1,060
3	Server Z	1	0,400

Table B.10: ERMF - Tableau (Table 5.2)

Experimental Parameters	
n_{ges} [1]	3
λ [1]	1,0000

Service Request Section								Server Section							Dynamic Evaluation Section								
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
2	1	Alert Request	1,00	1,00	1,00	1,00	1,000	Action Performer	1	2	2	1	0,010	0,014	1,408	0,355	0,001	0,001	0,010	0,010	0,011	0,011
3	2	Ruleset Eva. Req.	1,00	1,00	1,00	1,00	1,000	ECA-E. Rule Engine	1	2	2	1	0,150	0,211	1,408	0,355	0,022	0,022	0,150	0,150	0,172	0,172
4	3	Weather Service	1,00	1,00	1,00	1,00	1,000	Weather Serv. Hdl.	1	∞	∞	2	0,470	1,000	2,128		0,000	0,000	0,470	0,470	0,470	0,470
4	4	Traffic Service	1,00	1,00	1,00	1,00	1,000	Traffic Service Hdl.	1	∞	∞	3	0,300	1,000	3,333		0,000	0,000	0,300	0,300	0,300	0,300
3	5	Web Services Req.	1,00	1,00	1,00	1,00	1,000	ECA-E. WS Proxy	1	1	1				∞		0,000	0,000	0,770	0,770	0,770	0,770
3	6	Mapping Request	1,00	1,00	1,00	1,00	1,000	ECA-E. Rule Engine	1	2	2	1	0,200	0,282	1,408	0,355	0,029	0,029	0,200	0,200	0,229	0,229
2	7	Evaluation Request	1,00	1,00	1,00	1,00	1,000	ECA-Engine Cont.	1	1	1				2,817		0,050	0,050	1,120	1,120	1,170	1,170
2	8	Initialization Request	1,00	1,00	1,00	1,00	1,000	Control Unit	1	2	2	1	0,350	0,493	1,408	0,355	0,050	0,050	0,350	0,350	0,400	0,400
1	9	Forecast Request	1,00	1,00	1,00	1,00	1,000	ERMF-System	1	1	1				2,817		0,102	0,102	1,480	1,480	1,582	1,582
1	10	Request Generation	1,00	1,00	1,00	1,00	1,000	Timer Application	1	1	1		1,418		0,705		0,000	0,000	1,418	1,418	1,418	1,418

Multiplexer Section				M/M/m rsp. M/M/∞					
j	$Name_j$	m_j	X_j [1]	λ_j [bb]	μ_j [bb]	ρ_j [bb]	$n_{i,q}$ [bb]	$n_{i,s}$ [bb]	n_i [bb]
1	ERMF Server	2	0,710	1,000	1,408	0,355	0,102	0,710	0,812
2	Weather Server	∞	0,470	1,000	2,128		0,000	0,470	0,470
3	Traffic Server	∞	0,300	1,000	3,333		0,000	0,300	0,300

Table B.11: Axis2 - Tableau (Table 5.3)

Experimental Parameters	
n_{ges} [1]	1000
λ_{bott} [1]	10,526
f	0,800
λ [1]	8,421

Service Request Section									Server Section						Dynamic Evaluation Section										
[bb]	i	SRq_i [bb]	$p_{p(i),i}$	$v_{p(i)}$ [bb-1]	$v_{i,int}$ [bb]	v_i [bb]	λ_i [bb]	$Server_i$ [bb]	$m_{p(i)}$ [bb-1]	$m_{i,int}$ [bb]	m_i [bb]	Mpx_i	X_i [bb]	$m_{i,mpx}$ [bb]	μ_i [bb]	ρ_i [bb]	$n_{i,q}$ [bb]	W_i [bb]	$n_{i,s}$ [bb]	Y_i [bb]	n_i [bb]	R_i [bb]
4	1	Encryption Req.	1,00	1,00	1,00	1,00	8,42	Encrypter	5	1	5	1	0,0006	0,400	689,66	0,002	0,000	0,000	0,012	0,001	0,012	0,001
4	2	MessageOut Req.	1,00	1,00	1,00	1,00	8,42	MessageOut Srv.	5	1	5	1	0,0045	0,400	89,89	0,019	0,000	0,000	0,094	0,011	0,094	0,011
4	3	Policy Det. Req.	1,00	1,00	1,00	1,00	8,42	Policy Det.	5	1	5	1	0,0004	0,400	1000,00	0,002	0,000	0,000	0,008	0,001	0,008	0,001
4	4	OpOut Phase Req.	1,00	1,00	1,00	1,00	8,42	OpOut Phase Srv.	5	1	5	1	0,0004	0,400	930,23	0,002	0,000	0,000	0,009	0,001	0,009	0,001
3	5	OutFlow Request	1,00	1,00	1,00	1,00	8,42	OutFlow Handler	5	1	5				170,65		0,000	0,000	0,123	0,015	0,123	0,015
3	5	Business Logic R.	1,00	1,00	1,00	1,00	8,42	Business Logic	5	1	5	1	0,0325	0,400	12,33	0,137	0,000	0,000	0,683	0,081	0,683	0,081
4	6	OpIn Phase Req.	1,00	1,00	1,00	1,00	8,42	OpIn Phase Srv	5	1	5	1	0,0006	0,400	645,16	0,003	0,000	0,000	0,013	0,002	0,013	0,002
5	7	Disp. Instance Req.	1,00	1,00	1,00	1,00	8,42	Instance Disp.	5	1	5	1	0,0329	0,400	12,15	0,139	0,000	0,000	0,693	0,082	0,693	0,082
5	8	HTTP Loc. Disp. R.	1,00	1,00	1,00	1,00	8,42	HTTP Loc. Disp.	5	1	5	1	0,0003	0,400	1481,48	0,001	0,000	0,000	0,006	0,001	0,006	0,001
5	9	SOAP MB. Disp. R.	1,00	1,00	1,00	1,00	8,42	SOAP MB Disp.	5	1	5	1	0,0001	0,400	4000,00	0,000	0,000	0,000	0,002	0,000	0,002	0,000
5	10	URI Op. Disp. Req.	1,00	1,00	1,00	1,00	8,42	URI Op. Disp.	5	1	5	1	0,0001	0,400	4000,00	0,000	0,000	0,000	0,002	0,000	0,002	0,000
5	11	Address Disp. Req.	1,00	1,00	1,00	1,00	8,42	Address Disp.	5	1	5	1	0,0287	0,400	13,93	0,121	0,000	0,000	0,604	0,072	0,604	0,072
4	12	Dispatch Request	1,00	1,00	1,00	1,00	8,42	Dispatcher	5	1	5				16,10		0,000	0,000	1,308	0,155	1,308	0,155
4	13	PreDispatch Req.	1,00	1,00	1,00	1,00	8,42	PreDispatcher	5	1	5	1	0,0050	0,400	80,00	0,021	0,000	0,000	0,105	0,013	0,105	0,013
4	14	Decrypt Req.	1,00	1,00	1,00	1,00	8,42	Decrypter	5	1	5	1	0,0005	0,400	816,33	0,002	0,000	0,000	0,010	0,001	0,010	0,001
5	15	SOAP Act. Disp. R.	1,00	1,00	1,00	1,00	8,42	SOAP Act. Disp.	5	1	5	1	0,0397	0,400	10,07	0,167	0,000	0,000	0,836	0,099	0,836	0,099
5	16	Req.t URI Disp. R.	1,00	1,00	1,00	1,00	8,42	Req. URI Disp.	5	1	5	1	0,0437	0,400	9,14	0,184	0,000	0,000	0,921	0,109	0,921	0,109
4	17	Transport Request	1,00	1,00	1,00	1,00	8,42	Transporter	5	1	5				11,98		0,000	0,000	1,757	0,209	1,757	0,209
3	18	InFlow Request	1,00	1,00	1,00	1,00	8,42	InFlow Handler	5	1	5				6,59		0,000	0,000	3,193	0,379	3,193	0,379
2	19	Execution Request	1,00	1,00	1,00	1,00	8,42	Service Exec.	5	1	5				5,26		0,000	0,000	4,000	0,475	4,000	0,475
1	20	Sup. and Ex. Req.	1,00	1,00	1,00	1,00	8,42	System	1	5	5			0,400	2,11	0,800	2,216	0,263	4,000	0,475	6,216	0,738
1	21	Generation Request	1,00	1,00	1,00	1,00	8,42	External Source	1	1	1		0,0118		8,42				993,784	118,012	993,784	118,012

Multiplexer Section			
j	$Name_j$	m_j	X_j [1]
1	CPU	2	0,190

Table B.12: HPI Search Portal - Tableau (Table 5.1)

Experimental Parameters	
$n_{gen}^{[1]}$	4000
$\lambda_{bott}^{[1]}$	12,9313
f	0,9000
$\lambda^{[1]}$	11,6382

Service Request Section								Server Section							Dynamic Evaluation Section									
[bb]	i	$SRq_i^{[bb]}$	$p_{p(i),i}$	$v_{p(i)}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	$\lambda_i^{[bb]}$	$Server_i^{[bb]}$	$m_{p(i)}^{[bb-1]}$	$m_{i,int}^{[bb]}$	$m_i^{[bb]}$	Mpx$_i$	$X_i^{[bb]}$	$m_{i,mpx}^{[bb]}$	$\mu_i^{[bb]}$	$\rho_i^{[bb]}$	$n_{i,q}^{[bb]}$	$w_i^{[bb]}$	$n_{i,s}^{[bb]}$	$Y_i^{[bb]}$	$n_i^{[bb]}$	$R_i^{[bb]}$
2	1	Rendering Request	1,00	1,00	1,00	1,00	11,638	Webpage Renderer	1	1	1	1	0,0005	0,007	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
3	2	Response Aggregation	1,00	1,00	1,00	1,00	11,638	Response Agg.	1	1	1	1	0,0002	0,002	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
5	3	Resp. + Previews Agg.	1,00	1,00	1,00	1,00	11,638	Aggregator	1	1	1	1	0,0002	0,003	12,931		0,000	0,000	0,900	0,077	0,900	0,077
6	4	HTML Parsing	1,00	1,00	1,00	1,00	11,638	HTML Parser	1	1	1	1	0,0002	0,003	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
6	5	Preview Generation	1,00	1,00	20,00	20,00	232,764	Preview Generator	1	3	3	2	0,0001	1,000	19230,769	0,004	0,000	0,000	0,012	0,000	0,012	0,000
6	6	HTML Generation	1,00	1,00	1,00	1,00	11,638	HTML Generator	1	1	1	1	0,0002	0,003	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
5	7	Preview	1,00	1,00	1,00	1,00	11,638	Preview Handler	1	1	1				12,931		16,200	1,392	1,812	0,156	18,012	1,548
5	8	Aggregation + Generation	1,00	1,00	1,00	1,00	11,638	Aggregator + Gen.	1	1	1	1	0,0003	0,004	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
7	9	SOAP Unwrapping	1,00	5,00	1,00	5,00	58,191	SOAP Unwrapper	3	1	3	3	0,0001	0,152	1515,152	0,013	0,000	0,000	0,038	0,001	0,038	0,001
7	10	Search Engine 1 Search	1,00	5,00	1,00	5,00	58,191	Search Engine 1	3	∞	∞	4	0,0087	1,000	64,657		0,000	0,000	0,938	0,016	0,938	0,016
7	11	SOAP Generation	1,00	5,00	1,00	5,00	58,191	SOAP Generator	3	1	3	3	0,0001	0,182	1515,152	0,013	0,000	0,000	0,038	0,001	0,038	0,001
6	12	Search Engine 1 Handling	1,00	1,00	5,00	5,00	58,191	SE 1 Handler	1	3	3				1515,152		0,000	0,000	1,015	0,017	1,015	0,017
7	13	REST Unwrapping	1,00	5,00	1,00	5,00	58,191	REST Unwrapper	4	1	4	5	0,0008	0,053	65,789	0,221	0,000	0,000	0,885	0,015	0,885	0,015
7	14	Search Engine 2 Search	1,00	5,00	1,00	5,00	58,191	Search Engine 2	4	∞	∞	6	0,0090	1,000	111,111		0,000	0,000	0,885	0,015	0,885	0,015
7	15	REST Generation	1,00	5,00	1,00	5,00	58,191	REST Generator	4	1	4	5	0,0030	0,197	65,789	0,221	0,004	0,000	0,885	0,015	0,888	0,015
6	16	Search Engine 2 Handling	1,00	1,00	5,00	5,00	58,191	SE 2 Handler	1	4	4				65,789		0,004	0,000	2,654	0,046	2,657	0,046
7	17	HTML Parsing	1,00	5,00	1,00	5,00	58,191	HTML Parser	5	1	5	7	0,0024	0,102	42,553	0,273	0,008	0,000	1,367	0,024	1,375	0,024
7	18	Search Engine 3 Search	1,00	5,00	1,00	5,00	58,191	Search Engine 3	5	∞	∞	8	0,0008	1,000	1250,000		0,000	0,000	0,047	0,001	0,047	0,001
7	19	HTML Generation	1,00	5,00	1,00	5,00	58,191	HTML Generator	5	1	5	7	0,0023	0,098	42,553	0,273	0,005	0,000	1,367	0,024	1,373	0,024
6	20	Search Engine 3 Handling	1,00	1,00	5,00	5,00	58,191	SE 3 Handler	1	5	5				42,553		0,013	0,000	2,782	0,048	2,794	0,048
5	21	Search Request	1,00	1,00	1,00	1,00	11,638	Searcher	1	1	1				42,553		0,013	0,001	2,782	0,239	2,794	0,240
5	22	Req. + SOAP Generation	1,00	1,00	1,00	1,00	11,638	Req. + SOAP Gen.	1	1	1	1	0,0008	0,010	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
4	23	Super Search Request	1,00	1,00	1,00	1,00	11,638	Super Search Srv.	1	1	1				12,931		32,413	2,785	7,294	0,627	39,707	3,412
5	24	Aggregation	1,00	1,00	1,00	1,00	11,638	Aggregator	1	1	1	1	0,0006	0,008	12,931	0,900	32,413	2,785	0,900	0,077	33,313	2,862
6	25	HTML Parsing	1,00	1,00	1,00	1,00	11,638	HTML Parser	1	1	1	1	0,0002	0,003	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
6	26	Preview Generation	1,00	1,00	20,00	20,00	232,764	Preview Generator	1	3	3	2	0,0050	1,000	200,000	0,388	0,083	0,000	1,164	0,005	1,247	0,005
6	27	HTML Generation	1,00	1,00	1,00	1,00	11,638	HTML Generator	1	1	1	1	0,0002		12,931	0,900	8,183	0,703	0,900	0,077	9,083	0,780
5	28	Preview	1,00	1,00	1,00	1,00	11,638	Preview Handler	1	1	1				12,931		16,366	1,406	2,964	0,255	19,330	1,661
5	29	Aggregation + Gen.	1,00	1,00	1,00	1,00	11,638	Aggregator + Gen.	1	1	1	1	0,0003	0,004	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
5	30	Index Search	1,00	1,00	1,00	1,00	11,638	Index Searcher	1	1	1	9	0,0050	1,000	200,000	0,058	0,004	0,000	0,058	0,005	0,062	0,005
5	31	File Search Req. Gen.	1,00	1,00	1,00	1,00	11,638	Generator	1	1	1	1	0,0008	0,010	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
4	32	File Search	1,00	1,00	1,00	1,00	11,638	File Searcher	1	1	1				12,931		64,983	5,584	5,722	0,492	70,705	6,075
5	33	Response Sending	1,00	1,00	1,00	1,00	11,638	Response Sender	1	1	1	1	0,0054	0,070	12,931	0,900	0,000	0,000	0,900	0,077	0,900	0,077
7	34	Response Handling	1,00	10,00	1,00	10,00	116,382	Response Handler	1	1	1	1	0,0015	0,194	129,313	0,900	0,000	0,000	0,900	0,008	0,900	0,008
7	35	Picture Engine Search	1,00	10,00	1,00	10,00	116,382	Picture Engine	1	1	1	10	0,0040	1,000	250,000	0,466	0,405	0,003	0,466	0,004	0,871	0,007
7	36	Request Generation	1,00	10,00	1,00	10,00	116,382	Request Generator	1	1	1	1	0,0057	0,737	129,313	0,900	8,100	0,070	0,900	0,008	9,000	0,077
6	37	Picture Search	0,40	5,00	5,00	10,00	116,382	Picture Searcher	1	1	1				129,313		8,505	0,073	2,266	0,019	10,771	0,093
7	38	Response Handling	1,00	5,00	1,00	5,00	58,191	Response Handler	1	1	1	1	0,0025	0,162	64,657	0,900	8,100	0,139	0,900	0,015	9,000	0,155
7	39	3D Path Generation	1,00	5,00	1,00	5,00	58,191	3D Path Generator	1	2	2	11	0,0025	1,000	400,000	0,073	0,001	0,000	0,145	0,003	0,146	0,003
7	40	Extraction and Generation	1,00	5,00	1,00	5,00	58,191	Extractor and Gen.	1	1	1	1	0,0021	0,136	64,657	0,900	8,101	0,139	0,900	0,015	9,001	0,155
6	41	Room Search	1,00	5,00	1,00	5,00	58,191	Room Searcher	1	1	1				64,657		16,202	0,278	1,945	0,033	18,147	0,312
5	42	People Detail Search	1,00	1,00	5,00	5,00	58,191	People Search H.	1	1	1				64,657		24,707	0,425	4,211	0,072	28,918	0,497
5	43	Aggregation + Generation	1,00	1,00	1,00	1,00	11,638	Aggregator + Gen.	1	1	1	1	0,0098	0,127	12,931	0,900	0,000	0,000	0,900	0,077	0,900	0,077
6	44	Response Parsing	1,00	1,00	1,00	1,00	11,638	Response Parser	1	1	1	12	0,0090	1,000	111,111	0,105	0,000	0,000	0,105	0,009	0,105	0,009
6	45	People Search	1,00	1,00	1,00	1,00	11,638	People Searcher	1	1	1	13	0,0024	1,000	416,667	0,028	0,001	0,000	0,028	0,002	0,029	0,002
6	46	LDAP Request Generation	1,00	1,00	1,00	1,00	11,638	LDAP Req. Gen.	1	1	1	12	0,0025	1,000	400,000	0,029	0,001	0,000	0,029	0,003	0,030	0,003
5	47	LDAP Service Request	1,00	1,00	1,00	1,00	11,638	LDAP Handler	1	1	1				400,000		0,002	0,000	0,162	0,014	0,163	0,014
5	48	People Search Req. Gen.	1,00	1,00	1,00	1,00	11,638	Request Generator	1	1	1	1	0,0080	0,103	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
4	49	People Search	1,00	1,00	1,00	1,00	11,638	People Searcher	1	1	1				12,931		32,809	2,819	7,073	0,608	39,881	3,427
5	50	Response Handling	1,00	3,00	1,00	3,00	34,915	Response Handler	1	1	1	1	0,0012	0,047	38,794	0,900	40,909	1,172	0,900	0,026	41,809	1,197
5	51	Calendar Search Req.	1,00	3,00	1,00	3,00	34,915	Calendar	1	1	1	14	0,0090	1,000	111,111	0,314	0,144	0,004	0,314	0,009	0,458	0,013
5	52	Request Generation	1,00	3,00	1,00	3,00	34,915	Request Generator	1	1	1	1	0,0090	0,349	38,794	0,900	8,100	0,232	0,900	0,026	9,000	0,258
4	53	Event Search Request	1,00	1,00	3,00	3,00	34,915	Event Searcher	1	1	1				38,794		49,153	1,408	2,114	0,061	51,267	1,468
5	54	Response Handling	1,00	3,00	1,00	3,00	34,915	Response Handler	1	1	1	1	0,0001	0,005	38,794	0,900	49,153	1,408	0,900	0,026	50,053	1,434
5	55	3D Path Generation	1,00	3,00	1,00	3,00	34,915	3D Path Generator	1	2	2	11	0,0025	1,000	400,000	0,044	0,000	0,000	0,087	0,003	0,087	0,003
5	56	Extraction and Generation	1,00	3,00	1,00	3,00	34,915	Extractor and Gen.	1	1	1	1	0,0002	0,006	38,794	0,900	8,100	0,232	0,900	0,026	9,000	0,258
4	57	Room Search	1,00	1,00	3,00	3,00	34,915	Room Searcher	1	1	1				38,794		57,253	1,640	1,887	0,054	59,140	1,694
3	58	Portal Search Serv. Req.	1,00	1,00	1,00	1,00	11,638	Portal Searcher	1	1	1				12,931		64,983	5,584	7,294	0,627	72,277	6,210
3	59	Sub-Request Generation	1,00	1,00	1,00	1,00	11,638	Sub-Request Gen.	1	1	1	1	0,0003	0,004	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
2	60	Portal Request	1,00	1,00	1,00	1,00	11,638	Portal Server	1	1	1				12,931		81,183	6,976	9,094	0,781	90,277	7,757
2	61	Webpage Request	1,00	1,00	1,00	1,00	11,638	Webpage Req.	1	1	1	1	0,0002	0,003	12,931	0,900	8,100	0,696	0,900	0,077	9,000	0,773
1	62	Webrequest	1,00	1,00	1,00	1,00	11,638	Webrequest Exec.	1	1	1				12,931		97,383	8,368	10,894	0,936	108,277	9,304
1	63	Job Generation	1,00	1,00	1,00	1,00	11,638	Generator	1	1	1	Clien	334,3927		0,003		0,000	0,000	3891,723	334,393	3891,723	334,393

Multiplexer Section			
j	$Name_j$	m_j	$X_j^{[1]}$
1	HPI B-1.3 - Main	2	0,155
2	HPI C-1.14 - Preview Srv.	8	0,101
3	HPI V-2.01 - SE 1 Serv.Srv.	1	0,001
4	Search Engine 1 Server	∞	
5	HPI C1.1 - SE 2 Serv. Srv.	1	0,019
6	Search Engine 2 Server	∞	
7	HPI C1.17 - SE 3 Serv. Srv.	1	0,024
8	Search Engine 3 Server	∞	
9	HPI C-1.14 - Index Serv.Srv.	4	0,005
10	HPI C1.18 - Pic. Serv. Srv.	2	0,040
11	HPI A-2.11 - 3DBIMService	4	0,020
12	HPI C1.3 - LDAP Serv. Srv.	2	0,009
13	HPI LDAP Server	4	0,002
14	HPI C-1.18 - Event Srv.	4	0,027

Appendix C

Figures

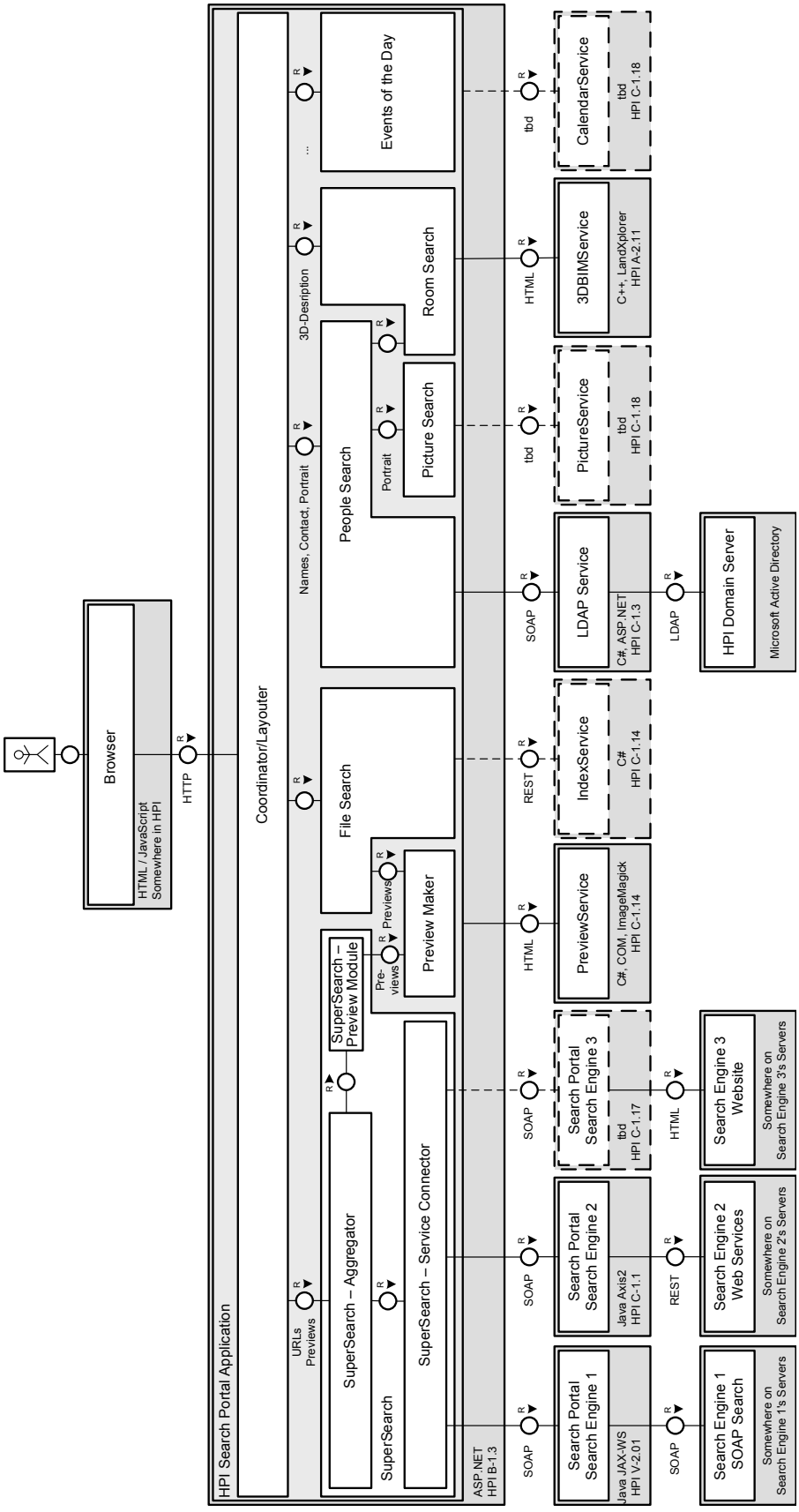


Figure C.1: HPI Search Portal - Architecture (Figure 5.1)

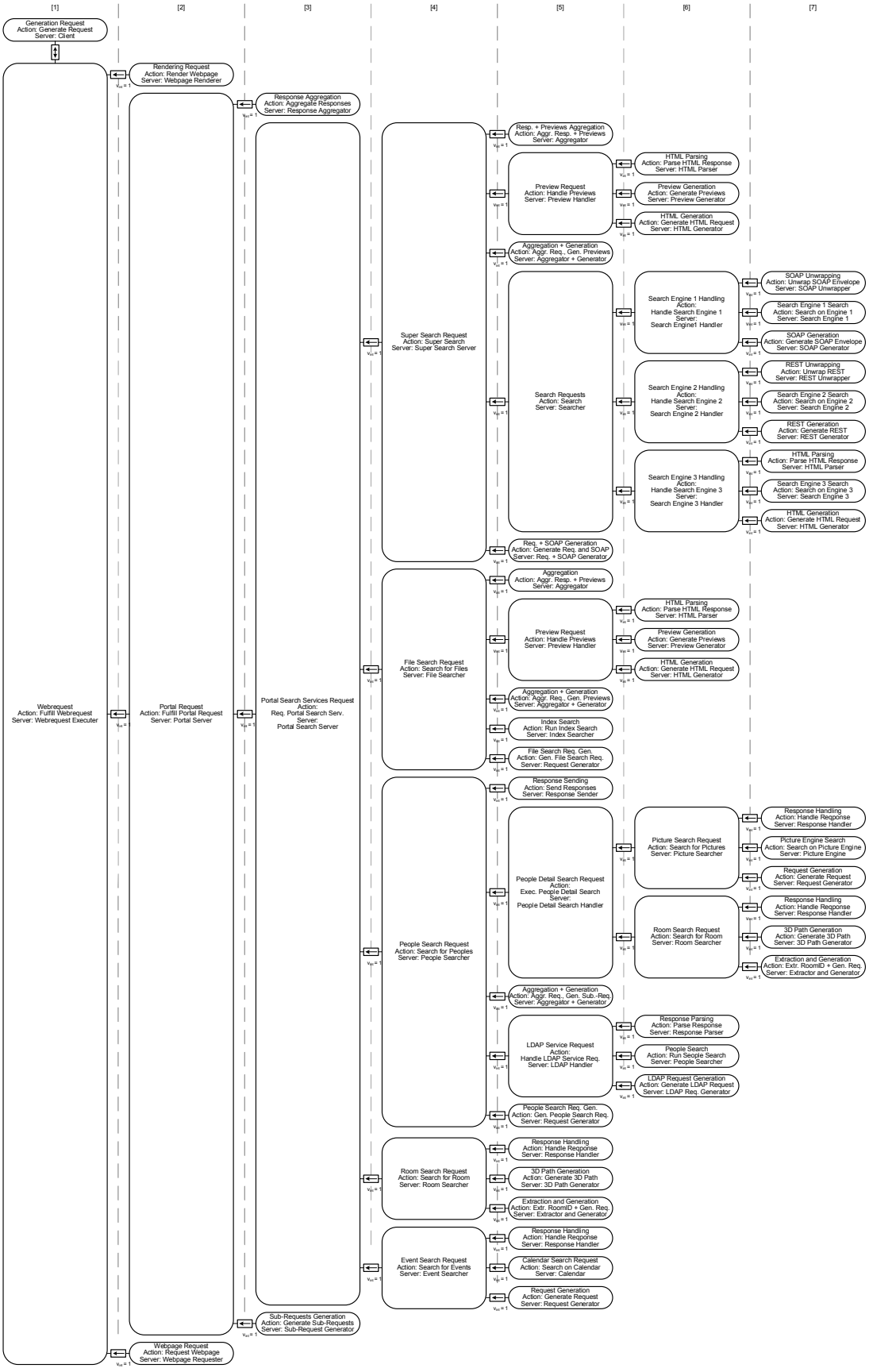


Figure C.2: HPI Search Portal - Service Request Structure (Figure 5.2)

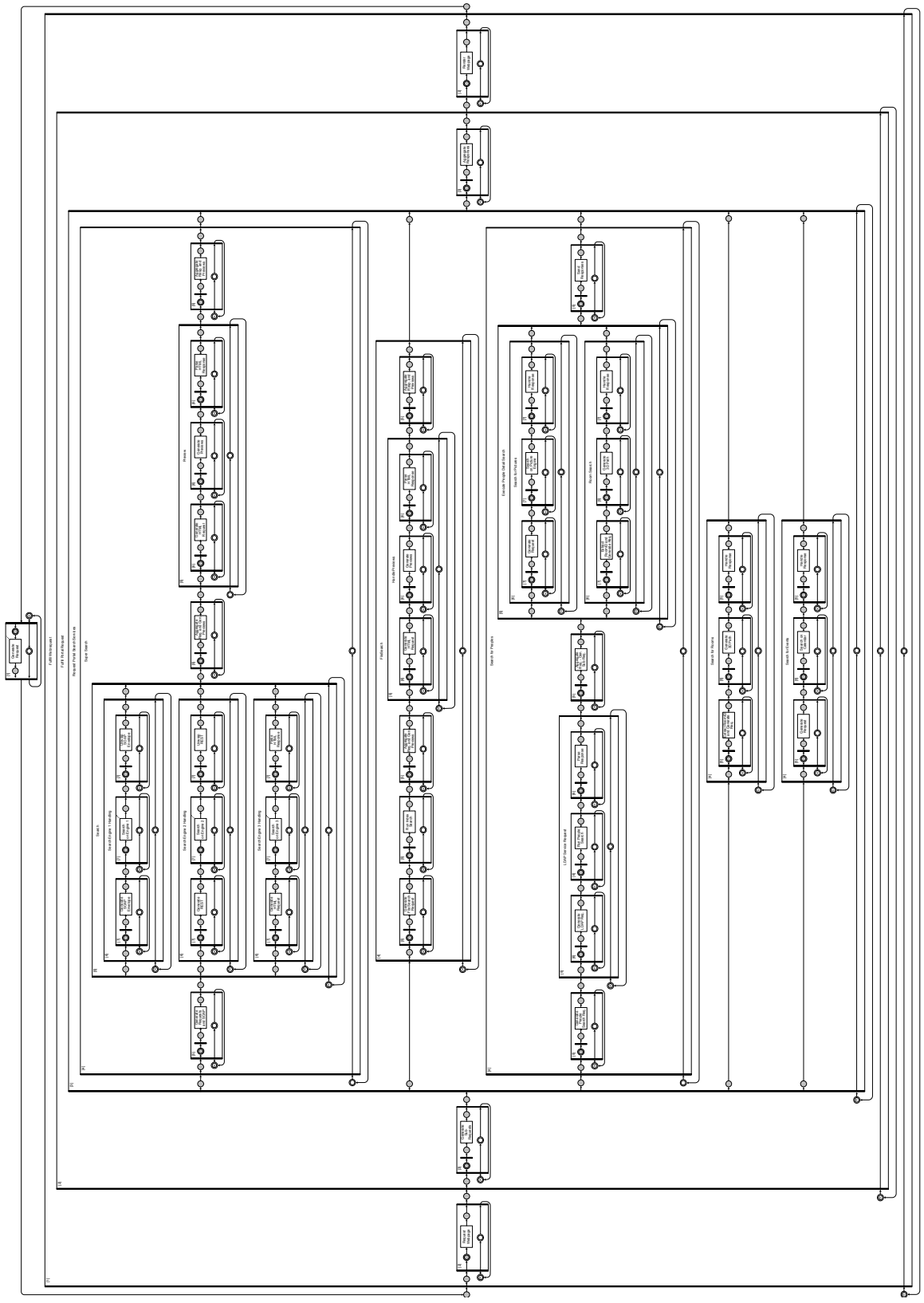


Figure C.3: HPI Search Portal - Behavior (Figure 5.3)

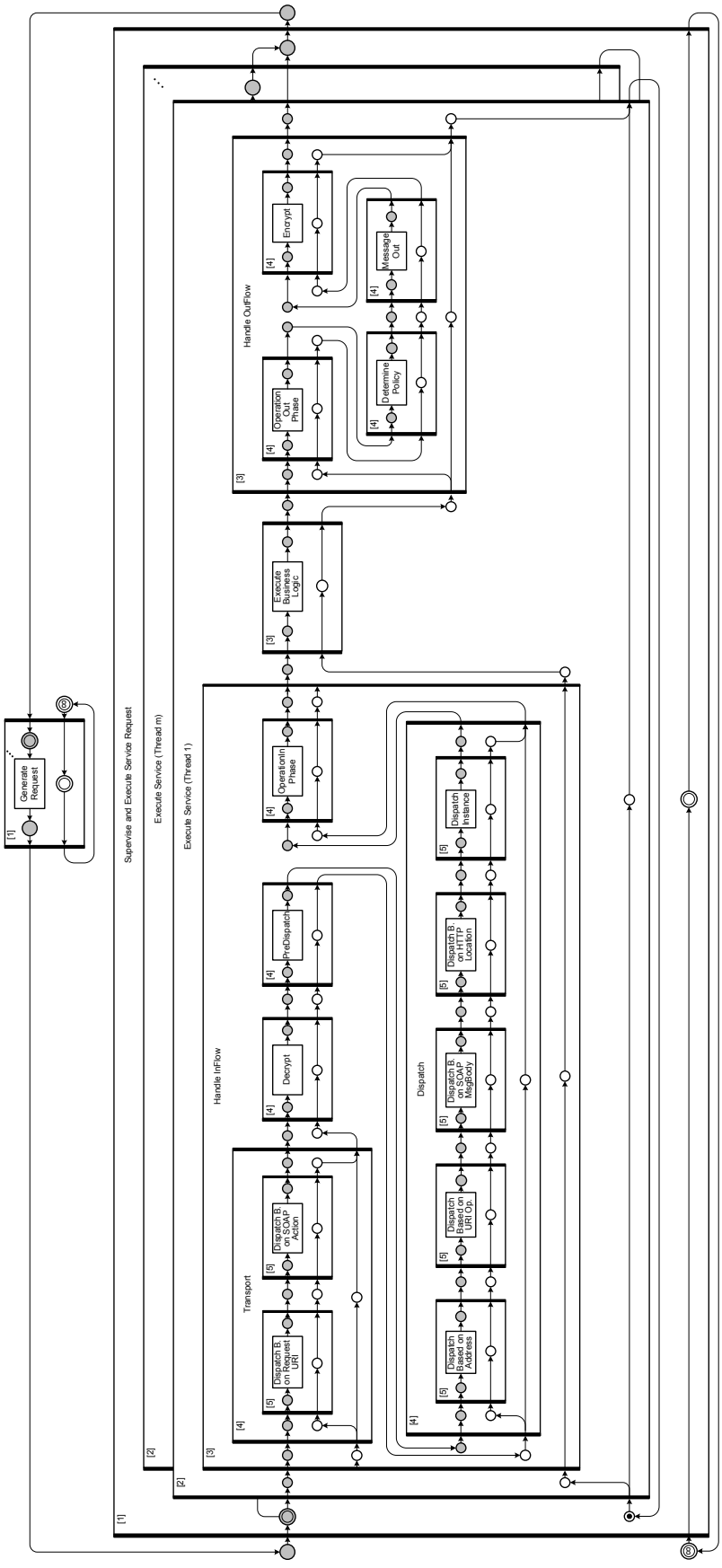


Figure C.4: Axis2 - Dynamic - All (Figure 5.22)