Performance Modeling of ASP.Net Web Service Applications: an industrial case study

Thijmen de Gooijer

ABB Corporate Research
Software Architecture and Usability

Thesis supervisor: Anton Jansen
Advisor: Heiko Koziolek

Mälardalens Högskola
Akademin för innovation,
design och teknik

A thesis in partial fulfillment of
the requirements for the degree
Master of Science
in Software Engineering

Thesis supervisor: Cristina Seceleanu
Examiner: Ivica Crnkovic

Vrije Universiteit
Faculteit der Exacte Wetenschappen

A thesis in partial fulfillment of
the requirements for the degree
Master of Science
in Computer Science

Examiner: Patricia Lago

Västerås, Sweden
July 13, 2011

The field resembled, more than anything else,
an old seafarer's chart with one big corner labeled
"Here there be monsters."

Only the brave of heart ventured forth ...

Paul Clements in his foreword to [SW02]

Abstract

During the last decade the gap between software modeling and performance modeling has been closing. For example, UML annotations have been developed to enable the transformation of UML software models to performance models, thereby making performance modeling more accessible. However, as of yet few of these tools are ready for industrial application. In this thesis we explorer the current state of performance modeling tooling, the selection of a performance modeling tool for industrial application is described and a performance modeling case study on one of ABB's remote diagnostics systems (RDS) is presented. The case study shows the search for the best architectural alternative during a multi-million dollar redesign project of the ASP.Net web services based RDS back-end. The performance model is integrated with a cost model to provide valuable decision support for the construction of an architectural roadmap. Despite our success we suggest that the stability of software performance modeling tooling and the semantic gap between performance modeling and software architecture concepts are major hurdles to widespread industrial adaptation. Future work may use the experiences recorded in this thesis to continue improvement of performance modeling processes and tools for industrial use.

Sammanfattning

Under det senaste decenniet har gapet mellan mjukvaru- och prestandamodellering minskat. Exempelvis har UML-annotationer tagits fram för att möjliggöra transformering av UML-modeller till prestandamodeller, vilket gör prestandamodellering mer allmänt tillgänglig. Än så länge är dock få av dessa verktyg redo för bredare användning inom industrin. I detta examensarbete har nuläget avseende verktyg för prestandamodellering utforskats, val av prestandamodelleringsverktyg för industriell tillämpning beskrivs, och en fallstudie inom ABB på ett fjärrdiagnostiksystem (RDS) presenteras. Fallstudien beskiver sökandet efter det optimala arkitektoniska alternativet för ett multimiljonprojekt avseende vidareutvecklingen av RDS-systemet, som är baserat på ASP.Net Web Services. Fallstudiens prestandamodell är integrerad med en kostnadsmodell för att ge värdefullt beslutsstöd vid utformandet av en handlingsplan för den framtida arkitekturen av RDS-systemet. Trots vår lyckade fallstudie anser vi att stabiliteten i modelleringsverktygen för prestandamodellering av mjukvarusystem samt den semantiska skillnaden mellan koncepten inom prestandamodellering och koncepten inom mjukvaruarkitektur framgent utgör stora hinder för utbredd industriell användning av prestandamodellering. De erfarenheter som är beskrivna i det här examensarbetet kan fritt användas till framtida förbättring av processer för prestandamodellering och verktyg för industriell användning.

Contents

Chapter 1

Introduction

In this thesis, we set out to create a performance model of a complex industrial system, which should help in selecting an architectural redesign that offers 10x more capacity. This Chapter first positions this thesis within the performance modeling research field in Section 1.1 and then explains ABB's interest in this thesis in Section 1.2. Once this foundation is in place, we identify the goal of the work and formulate tasks to reach this goal in Section 1.3. The latter will also indicate the contributions and findings of this thesis. We close this introduction with an outline for the remainder of the thesis in Section 1.4.

1.1 Research Motivation

Traditionally, there exists a gap between performance modeling concepts and software modeling concepts, making it difficult for architects to use performance models without consulting an expert. This gap is now closing, making performance modeling more accessible. For example, UML annotations have been developed to enable the transformation of UML software models to performance models, thereby removing the need for deep knowledge about performance modeling concepts such as queueing networks and Petri nets. In addition to these, strategies have been developed to deal with the state space explosion problem that limits the size of Markov chain based models. Finally, many techniques that underly software performance modeling are also used outside software engineering, where they are considered mature, and are frequently used. Yet, the industrial adaptation of software performance engineering seems to be low [SMF+07].

Sankarasetty et al. suggest that a poor toolset is one of the factors that is holding back industry acceptance of software performance modeling. Indeed many tools offer either maturity or familiar modeling concepts (e.g., annotated UML), but fail to offer both. Furthermore, the number of case studies on realistic, large industry software systems is limited. Industry might thus be right in its slow acceptance for it is not clear whether performance modeling can be easily integrated in existing software design and development.

1.2 Business Motivation

ABB's Corporate Research Center (CRC) is engaged in a project to improve the performance of a remote diagnostic solution (RDS) by architectural redesign. The RDS under review is owned by one of ABB's business units (RDSBU) and is used for service activities on deployed industrial equipment (devices). The RDS back-end is operating at its performance and scalability limits. Performance tuning or short term fixes (e.g., faster CPUs) will not sustainably solve the problems for several reasons. First, the architecture was conceived in a setting where time-to-market took priority over performance and scalability requirements. Second, the number of devices connected to the back-end is expected to grow by an order of magnitude within the coming years. Finally, the amount of data received from the devices, which has to be processed, is also expected to increase by an order of magnitude in the same period. Both dimensions of growth will significantly increase the demands on computational power and storage capacity and justify architectural redesign.

The main goal of the architectural redesign is to improve performance and scalability of the existing system, while controlling cost. It is not feasible to identify the best design option by prototyping or measurements. Changes to the existing system would be required to take measurements, but the cost and effort required to alter the system solely for performance tests are too high because of its complexity. Further, it is difficult to tell whether a measured performance improvement is the effect of a parameter change or the effect of a random change in the environment [Jai91]. Therefore performance modeling is considered a key tool to ensure that the right architectural decisions are taken.

1.3 Goal and Contributions

Based on the problems outlined in the previous Sections we formulate the following goals. Each goal is achieved by a set of tasks that is listed as a numbered list. Not all tasks fall within the scope of this thesis report. The thesis focusses on the actual performance model construction and validation. The other tasks have been carried out by the CRC project team or have been a joined effort between the author and the CRC project team. The goals and their tasks are listed below.

Goal 1. *Identify the best architectural alternative to achieve a scalable speed-up of at least one order of magnitude to counter the experienced performance problems and to enable the predicted growth. The desired speed-up takes into consideration the performance improvements of hardware in the prediction period, so the speed-up has to be achieved purely in software.*

Tasks towards Goal 1

1. Set performance and scalability requirements. (CRC project team)
2. Define architectural alternatives that improve performance and scalability. (joined effort, Chapter 6)
3. Select the most appropriate approach to model the RDS. (Chapter 5)

4. Create a performance model of the current implementation of the RDS. (Chapter 6)
5. Obtain performance measurements of the current implementation to validate the created model. (CRC project team)
6. Validate the created model using measurements on the current implementation obtained through experiments. (Chapter 6)
7. Predict the maximum capacity of the current system and alternative architectural designs. (Chapter 7)
8. Create an architectural roadmap. (joined effort, Section 7.3)
9. Prototype the critical parts of the envisioned architecture that could not be explored sufficiently with the performance model. (CRC project team)

Goal 2. *Evaluate the use of performance modeling for architectural redesign in industrial settings.*

Tasks towards Goal 2

1. Survey the available performance modeling tools. (Chapter 4)
2. Maintain a list of lessons learned while applying the selected performance modeling technique and tool (Chapter 8)

This thesis reviews the literature for software performance modeling tools that are useful in an industrial setting in two steps. First, we identify the requirements for a tool to be used in an industrial setting. Then, we assess the tools for use in a multi-million dollar industrial architectural redesign project of an ASP.Net web service application. In the literature review we analyze the industrial applicability of more than 10 performance modeling tools and find that only some use formalisms close to software modeling and also offer the required functionality.

Next, we report on a realistic industrial performance modeling case study in which we use the Palladio-Bench performance modeling tool. We selected Palladio-Bench based on the aforementioned analysis. The case study shows how performance modeling can support decisions in the architectural redesign of a complex system that uses modern technologies. For in the case study, we have successfully built a performance model for a 300 KLOC system with a 15-20% prediction accuracy. Subsequently, we have used the model to analyze more than more than 10 alternative architectures. Thereby, we enabled ABB to take an informed decision on an architectural roadmap. Finally, the thesis reflects on the use of Palladio-Bench in an industrial setting.

1.4 Thesis Outline

The remainder of this thesis is structured as follows. In Chapter 2 we introduce the reader to scalability and performance modeling, and describe the state of the art. Related work is discussed in Chapter 3. Next, in Chapter 4 we select a performance modeling tool for use in our case study based on the requirements

that are identified in the same Chapter. Chapter 5 then describes the process we followed to create the performance model. The performance modeling and the studied architectural redesigns are the subject of Chapter 6. The results of evaluating the performance models and the selected architectural roadmap are both discussed in Chapter 7. We report on our experiences with performance modeling in industry using Palladio-Bench in Chapter 8. Finally, Chapter 9 outlines plans and ideas for future work and we close the thesis with our conclusions in Chapter 10.

Chapter 2

Background & Foundations

In this Chapter we introduce the reader to the three dimensions of scalability in the AFK scale cube, which are used in the architectural alternatives of the case study. We also give an introduction to performance engineering and to software performance modeling. For the latter two subjects we discuss the most important techniques and give an overview of the state-of-the-art.

2.1 Scalability in Three Dimensions

The architectural alternatives considered in the case study in Chapter 6 are based on the theory of the AFK scale cube [AF09]. In this section, we give an overview of the principles of this scale cube, which was developed by Abbott and Fisher while working for AFK Partners.

The AFK scale cube explains scalability as three fundamental dimensions, the axes of the cube. The initial point (0,0,0) where the axes intersect defines the point at which the system is least scalable. A system at this point is a monolithic application that cannot be scaled-out and which capacity can only be increased by increasing the capacity of the hardware. Moving the application away from the initial point along any of the axes increases the scalability, enabling the increase of capacity by spreading demand over multiple hardware resources. We will now discuss the principles for each axis. The axes are named X, Y, and Z, and are illustrated in Figure 2.1.

Moving along the X-axis (applying an X-split) can be done by cloning services and data. The cloning should be done in such a way that work can be distributed without any bias. For example, we may install a second server to publish an exact copy of our website.

Scaling in the Y-direction (applying a Y-split) we divide the work into tasks based on the type of data, the type of work performed for a transaction, or a combination of both. Y-splits are sometimes referred to as service or resource oriented splits. A Y-direction scale-out introduces specialization, i.e., instead of having one worker doing all the work, we have multiple workers all performing a task within the process. For example, splitting our website into application logic and a database will allow us to deploy these functions on two different machines.

When we scale along the Z-axis (apply a Z-split) we separate work based on

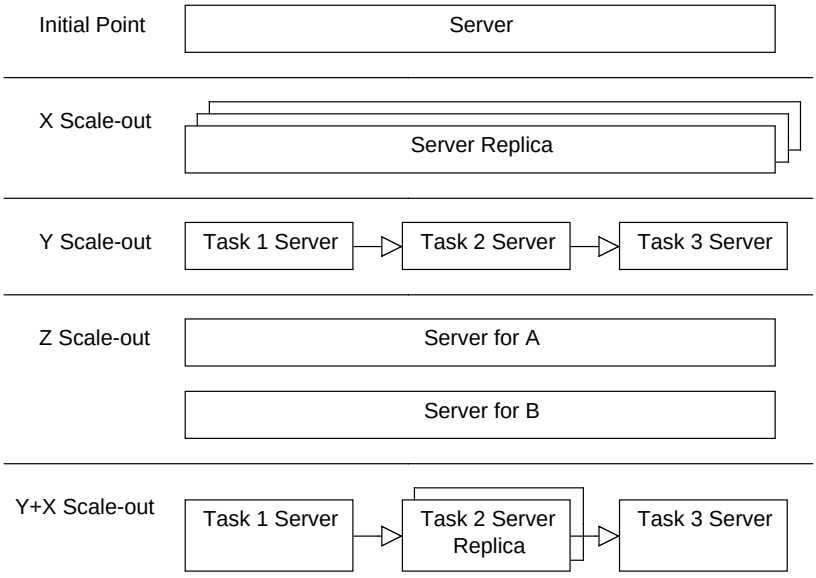

Figure 2.1: The three axes of the AFK scale cube

the requester (i.e., customer or client). This can be either by the data associated with a request, or the actions required by a request, or the person or system for which the request is being performed. For example, after replicating our server we may route customers from the US to a different server than customers from Europe, or clients who have account type A are served by a different server than clients who have account type B. When applying a Z-split we partition our data. That is, data is not mirrored between the replicas like when applying an X-split.

It is also possible to move along several axes. For example, once the process is cut into tasks via Y-splits, replication via X-splits can be applied on individual tasks. This example is illustrated in Figure 2.1 where in the 'Y+X Scale-out' pane a replica has been added for task 2.

2.2 Performance Engineering

According to Woodside the approaches to performance engineering can be divided into two categories: measurement-based and model-based [WFP07]. The former is most common and uses testing, diagnosis and tuning once a running system exists that can be measured. It can thus only be used towards the end of the software development cycle. The model-based approach focusses on the earlier development stages instead and pioneered with the Software Performance Engineering (SPE[1]) approach by Smith (e.g., [WS98]). Like the name suggests, in this approach models are key to make quantitative predictions on how well an architecture may meet its performance requirements.

[1]In this thesis performance engineering should usually be read as the general field, if we refer to the specific approach by Smith and Williams this will be pointed out or obvious from the context

Other taxonomies also exist, for example, Jain additionally considers simulation-based techniques as a separate category [Jai91]. On the other hand Koziolek dropped the distinction of categories entirely in the context of performance evaluation of component-based systems. He argues that most modeling approaches take some measurement input and most measurement methods feature some model [Koz10]. This fading of the boundary between measurement-based and model-based techniques is illustrated by the work of Thakkar et al., who present a framework to support the automatic execution of the huge amount of performance tests needed to build a model from measurement data [THHF08]. Thakkar et al. also suggest that academia should invest in unifying and automating the performance modeling process using measurement-based techniques, as these are more widely accepted in the industry. To structure the discussions in this Chapter, we will use Woodside's taxonomy of measurement-based versus model-based. In the following the measurement-based and model-based approaches are introduced with their strengths and weaknesses. Section 2.3 will then dive deeper into performance modeling.

2.2.1 Measurement-based performance engineering

Measurement-based approaches prevail in industry [SMF+07] and are typically used for verification (i.e., does the system meet its performance specification?) or to locate and fix hot-spots (i.e., what are the worst performing parts of the system?). Performance measurement dates back to the beginning of the computing era, which means there is a complete range of tools available, such as load generators, to create artificial system workloads, and monitors to do the actual measuring. Examples of commercial tools are Mercury LoadRunner, Neotys Neoload, Segue SilkPerformer, and dynaTrace.

An example of a state-of-the-art measurement-based tool is JEETuningExpert, which automatically detects performance problems in Java Enterprise Edition applications [CZMC09]. After detection it proposes ways to remove the problems using rule-based knowledge of performance anti-patterns. JEETuningExpert also helps us to understand why one would need a measurement-based system, for the JEE middleware behaviour is hard to predict and design time decisions might be changed during implementation. The JEE middleware hides the location of EJB components and takes care of much other non-functional behaviour. Unfortunately, JEETuningExpert knows only four anti-patterns at the moment, but recognizes these almost flawlessly.

Performance testing applies measurement-based techniques and is usually done only after functional and load testing. Load tests check the functioning of a system under heavy workload. Whereas performance tests are used to obtain quantitative figures on performance characteristics, like response time, throughput and hardware utilization, for a particular system configuration under a defined workload [THHF08].

We close our discussion of (pure) measurement-based techniques with two lists, enumerating the strengths, and the weaknesses of these approaches, respectively. Note that neither list is exhaustive, but together they enable making a trade-off with model-based approaches. Note that, in this thesis we aim to predict the performance of architectures that have not been implemented yet. Thus measurement techniques will only be instrumental in creating performance models and not used directly to find and 'remove' problems.

Strengths

- Tool support; the maturity of the field comes with a lot of industry ready tooling,
- Acceptance of industry; in part due to the credibility of real measurements [THHF08, SMF+07],
- Accuracy of results; observing the real system means that the problems and improvements measured are more accurate than those found in a model.

Weaknesses

- Not easily nor commonly applied in the early software development stages such as (architectural) design [WFP07],
- Performance improvement by code 'tuning' will likely compromise the original architecture [WS98],
- "Measurements lack standards; those that record the application and execution context (e.g. the class of user) require source-code access and instrumentation and interfere with system operation." [WFP07],
- Tools suffer from "a conflict between automation and adaptability in that systems which are highly automated but are difficult to change, and vice versa" [WFP07]. In the end, users have difficulty finding a tool that satisfies their needs and end-up inventing their own [WFP07],
- Every tool has its own output formats making interoperability a challenge [WFP07],
- It is difficult to correlate events in distributed systems. Determining causality within the distributed system is made even more difficult by the integration of sub-systems from various vendors, which is common practise [WFP07],
- Measurements are more sensitive to Murphy's law than other approaches, making the amount of time required to do them less predictable. [Jai91, pg. 31],
- "The setting-up of the benchmarking environment as well as repeatedly executing test cases can run for an extended period of time, and consume a large amount of computing and human resources. This can be expensive and time-consuming." [JTHL07],
- Test results obtained in a benchmarking environment may disagree with the performance of the production environment, because the former is often of a smaller scale than the latter. [JTHL07].

2.2.2 Performance engineering through modeling

The importance of performance modeling is motivated by the risk severe performance problems (e.g. [BDIS04]) and the increased complexity of modern systems, which makes it difficult to tackle performance problems at the code level. Considerable changes in design or even architecture may be required to mitigate performance problems. Therefore, the performance modeling research community tries to fight the 'fix-it-later' approach to performance in the development process. The popular application of software performance modeling is then to find performance issues in software design alternatives early in the development cycle, hence avoiding the cost and complexity of redesign or even requirement changes.

Performance modeling tools help to predict a system's behaviour before it is built or to evaluate the result of a change before implementing it. Performance modeling may be used as an early warning tool throughout the development cycle with increasing accuracy and increasingly detailed models throughout the process. Early in development a model can obviously not be validated with the real system, then the model represents the designer's uncertain knowledge. As a consequence the model makes assumptions that do not necessarily hold for the actual system, but which are useful for obtaining an abstraction of the system behaviour. In these phases validation is obtained by using of the model, and there is a risk of wrong conclusions because of the limited accuracy. Later the model can be validated against measurements on (parts of) the real system or prototypes and the accuracy of the model increases.

Jin et al. suggest that current methods have to overcome a number of challenges before they can be applied to existing systems that face architectural or requirement changes [JTHL07]. First, it must become clear how values for model parameters are to be obtained and how assumptions can be validated. Experience-based estimates for parameters are not sufficient and measurements on the existing system are required to make accurate predictions. Second, the characterization of system load in a production environment is troublesome due to resource-sharing (e.g., database sharing, shared hardware). Third, methods have to be developed to capture load dependent model parameters. For example, an increase in database size will likely increase the demands on the server CPU, memory, and disk.

Common modeling techniques include queueing networks, extensions to these such as layered queueing networks, and various types of Petri nets and stochastic process algebras [WFP07]. A recent and promising development is that of automatic performance model generation from annotated UML specifications; i.e., the architect or designer annotates his UML models and a performance model is automatically generated [WFP07]. We will discuss all of these in more detail in the next Section. We now close this Section with an overview of the strengths and weaknesses of performance modeling.

Strengths

- One can predict system performance properties before it is built [WFP07],
- The effect of a change can be predicted before it is carried out [WFP07],
- Automated model-building from specified scenarios with support of UML

profiles early in the life cycle [WFP07].

Weaknesses

- "There is a semantic gap between performance concerns and functional concerns, which prevents many developers from addressing performance at all. For the same reason many developers do not trust or understand performance models, even if such models are available." [WFP07],
- Performance modeling often has high cost [WFP07],
- Models are only an approximation of the system and may omit details that are important [WFP07],
- It is difficult to validate models [WFP07],
- Models always depend on various assumptions. The validity of these is unknown in advance, nor is the sensitivity of the predictions to these assumptions known [WFP07].
- A good understanding of the system is required for performance modeling, asking for the complete and accurate documentation of system behavior. But, in practise up-to-date and complete documentation rarely exists [THHF08].

2.3 Introduction to Performance Modeling

This Section introduces model-based performance engineering. We distinguish two categories of performance modeling approaches: analytical techniques and simulation techniques. Thakkar summarizes the difference between the two as: "Analytical techniques use theoretical models. Simulation techniques emulate the functionally of the application using a computer simulation whose performance can be probed" [THHF08]. However, no clear distinction can be made between simulation and analytics. Some constructed models may be evaluated both analytically and by using simulation.

Compared to simulation analytical techniques require more simplifications and assumptions, and this is especially true for models employing queues [Jai91, pg. 394]. Analysis techniques for performance models that are directly based on the states and transitions of a system generate a state space. Often these techniques cannot be used to model systems of realistic size due to the so-called state space explosion problem, in which the set of states to be analyzed grows exponentially with the size of the model [e.g., WFP07]. On the upside there are improvements to numerical solution methods for state spaces and the approximations they use [WFP07].

Simulation techniques allow for more detailed studies of systems than analytical modeling [Jai91, pg. 394]. However, building a simulation model requires both strong software development skills and comprehensive statistical knowledge. Also, simulation models often require (much) more time to develop than analytical models. Jain reports that this (sometimes unanticipated) complexity causes simulation efforts to be cancelled prematurely more often than that they are completed [Jai91, pg. 393]. Over 15 years later Woodside still has a similar

opinion: "simulation model building is still expensive, sometimes comparable to system development, and detailed simulation models can take nearly as long to run as the system." [WFP07].

Frank's thesis tells us that simulations for layered queueing networks can take two orders of magnitude longer than analysis (i.e., hours vs. seconds) [Fra99, pp. 224–227]. Woodside notes however, that cheap, increasingly powerful computing makes simulation runtimes more agreeable. Yet, the complexity of simulations remains high, which is illustrated by the fact that the runtimes listed in [MKK11] still are one to two orders of magnitude longer for simulations than for the layered queueing network analysis tool. Simulations take much longer due to the often substantial amount of runs that is required to gain statistical confidence in the results and the duration of model execution. Rolia et al. suggest therefore that an analytical model should always be preferred if it provides accurate predictions and does so consistently [RCK+09].

2.3.1 Simulation versus process algebras

As an example of the trade-off between analytical and simulation techniques we report on the findings of Balsamo et al. [BMDI04]. They compare Æmilia, an analytical technique based on stochastic process algebras, and UML-Ψ (UML Performance SImulator), a simulation-based technique. Their work gives us insight in the strengths and weaknesses of the approaches when assessing performance on the architectural level.

Using the Æmilia architectural description language Balsamo et al. observed the aforementioned state space explosion problem for some of their scenarios. In these cases they experimented with UML-Ψ, which derives simulation models from annotated UML diagrams. This is one of the reasons that Balsamo et al. suggest that combining techniques can overcome limitations of single techniques and provide more useful results.

In UML-Ψ there exists a near one-to-one mapping from UML elements to simulation processes making it easy to construct the simulation-based performance model. The Æmilia model is also quite easily derived from the architectural specification, but requires information on the internal behaviour of the components. In Æmilia it is also quite cumbersome to include fork/join systems, resource possession, and arbitrary scheduling policies. An advantage of UML-Ψ is then that it puts little constraints on the expressiveness of the software model.

UML-Ψ allows to predict a wide range of different performance metrics. In Æmilia knowledge of the tool's internal functioning is necessary to specify performance indices. Æmilia has the advantage however that it computes an exact numerical result whereas the simulation expresses results in confidence intervals and needs many samples (and potentially a lot of execution time) to compute means and remove bias.

Finally we should remember that at the moment the scalability of the Æmilia approach is limited due to the state space explosion problem. According to Balsamo et al. this problem limits the applicability of Æmilia in 'real' situations. Solutions to the state space explosion problem require manual tuning of the generated models, which obviously requires skills and expertise. However, the TwoTowers analytical tool associated with the Æmilia approach does have the added benefit that it can analyse system functionality in addition to non-functional aspects like performance.

2.4 Performance Modeling Techniques

In the previous Sections we presented several performance engineering approaches and evaluated their strengths and weaknesses. This Section and Chapter 4 discuss concrete techniques and tools. Here we will discuss popular techniques independent of their implementation and in Chapter 4 we compare several tools employing these and other techniques.

2.4.1 Queueing Network Models

Queueing Networks and Queueing Network Models (QNM) form a generic performance modeling technique that also has many applications outside software performance modeling and computer science. A thorough discussion of their application in computer science can be found in [LZGS84]. We based our short overview on that of Kounev et al. in [KB03]. First we define several parameters used in QNM:

interarrival time the time between the arrival of successive requests.

service time the amount of time the request spends at a server: the time it takes the server to handle the request.

queueing delay the time a request spends waiting for service in the queue.

response time = $queueingdelay + servicetime$: the time a request spends within the service station.

A queueing network is a collection of connected queues. Every queue is part of a service station and regulates the entry of requests or jobs into its service station. Requests are handled by the service station's servers. Upon arrival at the station a request may be served directly if a server is available or if it causes a lower priority request to be preempted. Otherwise the request queues until a server frees up.

The queues might employ different strategies to decide which request is the next to receive service when a server becomes available; these strategies are called scheduling strategies. Typical strategies are first come first served, last come first served, processor sharing (processing power is equally divided among all requests and all requests are processed simultaneously), and infinite server (no queue ever forms, the server only introduces a processing delay). A service center with the infinite server scheduling strategy is also called a delay center.

A QNM is built by connecting queues. If from one point multiple routes may be followed, the likelihood of each route is specified with a probability. Requests may also return to a service station multiple times, in this case the service time is defined as the total amount of processing time required at the service station. Requests with similar service demands may be grouped in a class.

Measures typically derived from the evaluation of QNM are queue lengths, response time, throughput and utilization (of service stations). Several efficient methods exist to evaluate QNM, for example mean-value analysis. While QNM are very suitable for modeling hardware contention (e.g., contention for disk access), QNM are not very good at describing software contention. This limitation gave rise to the development of several extensions. In the next subsection we describe one of the most popular extensions to queueing networks: layered queueing networks.

2.4.2 Layered Queueing Networks

A popular extension to Queueing Network Models (QNM) are Layered Queueing Networks (LQN). LQN for distributed systems are introduced in [FMN+96]. LQN add support for (distributed) software servers, which are often 'layered': in distributed systems processes can act both as client and server to other processes. High-layer servers make use of services/functionality offered by servers at lower layers, and the delays at higher-layer servers depend on those at lower-layer servers.

Distributed systems may suffer from a bottleneck not caused by saturation of a hardware resource, but caused by a software server waiting for replies from lower level services: a software bottleneck. LQN can be used to find these problems by calculating service times of a process for its own execution, its requests to other servers and the queueing delays it experiences during those requests. Other performance issues that may be studied using LQN are the impact of different algorithms, load balancing, replication or threading, and an increase in the number of users. Replication and threading are techniques that may remove software bottlenecks. LQN can also be used for sensitivity analysis, for example, to determine the limit of multi-threading performance gains before the overhead (extra memory requirements for the heap, and execution stack) kicks in, or to measure the sensitivity to cache hit ratios [FMN+96].

A LQN models a system as requests for service between actors or processes and the queueing of a messages at actors [WHSB01]. Actors are modeled as tasks and accept service request messages at an LQN entry. An LQN entry provides a description of a service provided by an actor, and models this service and its resource demands. Tasks can be compared to objects and entries to methods, this analogy makes LQN conceptually easier to understand for software developers than many other performance modeling techniques.

Each LQN entry may have its own resource demands or performance parameters, for not all are expected to behave the same (e.g., a database read operation may be served more quickly than a write operation). Performance parameters also help deal with various types of distributions. The type of distribution deeply affects delays, consider for instance distribution across nodes on a WAN versus distribution across several processors on a shared bus. Further, LQN models are able to represent software at various levels of detail. In its simplest form it can be an ordinary queueing model not showing any software detail, but LQN can also model every software module in the design, all interactions between modules, and the resource demands for each of these [WHSB01].

2.4.3 Queueing Petri Nets

An ordinary Petri Net is a bipartite directed graph consisting of places, shown as circles, and transitions, shown as bars. We give a formal definition from [KB03]:

Definition 1. An ordinary Petri Net (PN) is a 5-tuple $PN = (P, T, I^-, I^+, M_0)$, where:

1. P is a finite and non-empty set of **places**,
2. T is a finite and non-empty set of **transitions**,

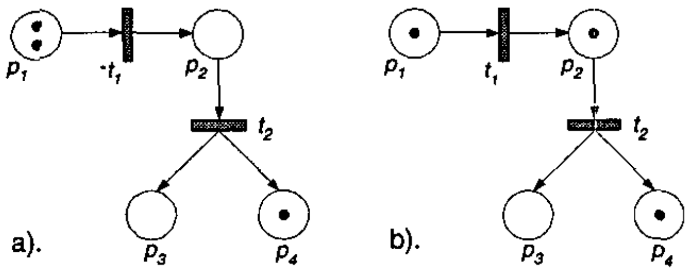

Figure 2.2: An Ordinary Petri Net before and after firing transition t_1 (taken from [KB03]

3. $P \cap T = \emptyset$,
4. $I^-, I^+ : P \times T \to \mathbb{N}_0$ are called backward and forward **incidence functions**, respectively,
5. $M_0 : P \to \mathbb{N}_0$ is called **initial marking**.

I^- and I^+, the incidence functions, specify the interconnection between places and transitions. Places may be input places or output places depending on whether there is an edge from the place to a transition (input place) or vice versa. Place p is an input place if $I^-(p,t) > 0$ and an output place if $I^+(p,t) > 0$. Each edge has a weight, which is assigned by the incidence functions. When a transition takes place it is said to fire. It then destroys N tokens from each of its input places, where N is equal to the weight of the edge between the input place and the transition. Transitions can only fire when all input places contain N tokens. Note that N is specific for each input place. Upon firing the transition creates tokens in output places, again the number of tokens created is dependent on the weight of the edge between the transition and the output place. An arrangement of tokens is called a marking and M_0 gives the initial arrangement of tokens in the Petri Net. An example of a simple Petri Net is shown in Figure 2.2. The example net has 4 places and 2 transitions, all edge weights are 1. The left of the figure shows the Petri Net before firing transition t_1, the right half shows the situation after firing.

Petri nets have been extended on several occasions either to increase convenience or expressiveness. A number of extension are combined in Colored, Generalized, Stochastic Petri Nets (CGSPNs), which add advanced timing control and typing of messages. None of the extensions are suitable to represent scheduling strategies, the speciality of queueing networks. To overcome this Queueing Petri Nets (QPN) include some ideas from queueing networks [KB03]. In QPNs the places of CGSPNs are extended by adding a queueing place, which consists of a QNM service station and a depository for the storage of tokens (requests in queueing networks) that have been serviced. While the readability of the models is reduced by the QPN extension, it does increase the expressiveness of the models [TM06].

One way to model a software system in QPNs is described by Tiwari and Mynampati in [TM06]. They map software processes onto ordinary places. The

number of process tokens in these places represent the number of available passive resource tokens (i.e., threads, connection pools, etc.). Transitions between places are defined so that they allow to a request token to move to the next queueing place when the right conditions are met.

Unfortunately, Petri nets are sensitive to the state space explosion problem inhibiting efficient analysis of large models. To alleviate this problem Falko Bause developed the HQPN (hierarchical QPN) formalism. In a HQPN model a queueing place might contain a whole QPN instead of a single queue. Queuing places are therefore known as subnet. This hierarchical nesting enables efficient numerical analysis and thereby alleviates the state space explosion problem [KB06]. However, when modeling systems of realistic size, the state space explosion can still be problematic [TM06].

LQN vs. QPN

Tiwari and Mynampati report on their experience applying both LQN and QPN to a J2EE application in [TM06]. We report some of their findings to compare the relative merits of the techniques. They succeeded applying both techniques and both models gave similar performance results, but it is interesting that they observe LQN to be less accurate in modeling software contention of a system than QPN. The remaining trade-offs are as follows (all quoted from [TM06]):

- QPN can be used to analyze both functional and performance aspects of system. Whereas LQN gives only the performance measures of a system.
- LQN can be analytically solved using the approximate MVA techniques with minimal resources; while QPN is analytically solved using Markov process thus requires resources that are exponential in the size of the model to produce exact results.
- LQN does not have any computational limitations, so can be modeled for any number of layers (tiers)/resources. Nevertheless the QPN computation model becomes exponentially complex with addition of each ordinary place and queueing place.
- The LQN models can be used for modeling any number of concurrent user requests. However the QPN model cannot be used for large number of concurrent requests due to state space explosion problem.
- LQN supports both the open (geometric distribution) and closed requests. While the QPN is restricted only to modeling the closed requests.
- LQN can be used to model synchronous, asynchronous and forward calls. So messaging systems can also be modeled with LQN. QPN supports only synchronous calls.
- In QPN memory size constraints for performance can be modeled more accurately than in the LQN.
- LQN model consists of convenient primitives notations which makes LQN construction simple, convey more semantic information and guarantee that these models are well-formed (i.e. stable and deadlock free) [DONA1995 (citation theirs)]. On the other hand, the low level notations used in QPN give them added expressive power with some readability complexity.

2.4.4 Palladio Component Model

The Palladio Component Model (PCM) is based on the roles and tasks within the component-based software engineering philosophy: component developer, architect, deployer, and domain expert. The component developer is responsible for developing software components. The architect combines these components to form a system, which is deployed by the deployer (i.e., system administrator and application manager). A domain expert specifies the usage of the system. We will now discuss the parts of the PCM in turn.

Component Repository

In the PCM, information about the available components and their interfaces is collected in the component repository. The component repository is created and maintained by the component developer(s). When a developer creates a component he should also specify its resource demands in a Service EFfect speciFication (SEFF). An important part of the SEFF is how the component uses its required interfaces. For example, a web page component SEFF may specify that the web page makes three calls to the database interface to read information. In addition, other resource requirements can be specified, for example, the amount of CPU cycles or time, or the number of threads required from a thread pool. Here a thread is an example of a software resource, which can be modeled in the PCM using passive resources (i.e., instead of processing resources such as a CPU).

System Model

Using the components from the component repository the architect can specify a system model. The architect selects the components he wishes to use and connects required interfaces to matching provided interfaces. Systems generally include at least one provided system interface to make its services available. Likewise systems may of course have required system interfaces to connect to other systems.

Allocation Model

The deployer of the system creates the allocation and resource environment diagrams. The resource environment diagram specifies the available hardware resources such as servers and their CPUs and memory. The allocation diagram then maps the system components to containers in the resource environment (i.e., typically servers).

Usage Model

Finally, the typical use of the system is specified by the domain expert in the usage model. The domain expert specifies usage scenarios and the amount of users or processing requests (in the case of batch jobs). The usage model may be modified to test system behaviour under various loads.

2.5 State-of-the-Art

In order to present a comprehensive state-of-the-art of software performance evaluation, that is, briefing the reader on techniques and tools beyond those used in this report, we first discuss a number of surveys, which can also be used as guide for further reading. After these, we discuss automated performance modeling, one of the field's frontiers. The contents of this Section should help the reader to judge the research field in which the thesis work was carried out.

In his survey of the state of the art and the future of software performance engineering, Woodside calls the state 'not very satisfactory' [WFP07]. Often heavy effort is required to carry out the performance engineering processes; there is a semantic gap between functional and performance concerns; and developers find it difficult to trust models: these are just three of the issues he identifies [WFP07]. He suggests that measurement and modeling based approaches have to be combined to ensure further progress.

Becker et al. and Koziolek surveyed performance evaluation for component-based systems in [BGMO06] and [Koz10]. They analyze the strengths and weaknesses of component-based software performance modeling tools and provide recommendations for future research. Koziolek blames immature component-based performance models, limited tool support for these and the absence of large-scale industrial case studies on component-based systems for the sparse industrial use of performance modeling for component-based systems [Koz10]. Out of the 13 tools he surveyed only four are rated to have the maturity required for industrial application, but the conclusion does note a significant advance in maturity over the last ten years of performance evaluation for component-based systems [Koz10].

The earlier findings of Balsamo et al. are similar. They note that despite of newly proposed approaches for early software performance analysis and some successful applications of these, performance analysis is still a long way from being integrated into ordinary software development [BDIS04]. And while no comprehensive methodology is available yet, they suggest that the field of model-based software performance prediction is sufficiently mature for approaches to be put into practice profitably.

2.5.1 Automatic Performance Modeling

Various approaches that automate (part of) the performance modeling process have been proposed over the years. Automation should make performance modeling faster and more accessible to software designers [WHSB01].

Model extraction from traces

A first category of approaches contains methods to extract performance models from execution traces. The work by Litoiu et al. describes one of the first trace-based model creation tools for 'modern' applications [LKQ+97]. They create layered queueing network (LQN) models using traces from EXTRA (a commercial tool then part of IBM's Visual Age suite). The EXTRA trace files may be imported into another IBM tool called DADT (Distributed Application Development Toolkit) to explore the impact of design and configuration changes on application performance.

In 2001 Woodside et al. presented a prototype tool to automatically construct layered queueing network models of real-time interactive software using execution traces of performance critical scenarios [WHSB01]. The extracted models may be maintained throughout the development cycle taking new traces gathered from the evolving system implementation. A limitation is that the traces have to be provided as angio-traces, a specific trace format.

Israr et al. propose the System Architecture and Model Extraction Technique (SAMEtech), which can use traces in standard formats to derive LQN models [IWF07]. SAMEtech can also combine information from multiple traces into one model and features a linearly scaling algorithm for large traces. Limitations include the inability to detect the joining of flows and synchronization delays associated with forking and asynchronous messages. Also workload parameters for the models are not automatically extracted.

Software design model transformation

In the second category we find several approaches that provide automatic transformation of software design models (e.g., UML models) to performance models, but to our knowledge, no robust, industry-ready tools are available yet. One of the problems is that it is difficult to translate incomplete or inconsistent design models into useful performance models.

Balsamo et al. transform UML use case, activity and deployment diagrams to multi-chain and multi-class queueing networks [BM05]. The UML diagrams have to be annotated with the UML profile for Schedulability, Performance and Time specification (SPT profile) to include the additional information required to specify a performance model. Unfortunately, the UML software model must be constrained to avoid the need for use of simulation or approximate numerical techniques. Petriu presents a similar automatic transformation of a UML specification into a layered queueing network [Pet05].

The work by Woodside et al. uses a common intermediate format, the Core Scenario Model (CSM), towards a framework for the translation of software design models to performance models, both in various notations [WPP+05]. The PUMA framework is more flexible than other model transformations: one can plug software design tools into the framework as sources and feed output to different performance modeling tools. In [WPP+05] input from UML 1.4 is used to create the CSM from which LQN, Petri Nets and QNM are built. The authors argue that a framework such as PUMA is a requirement for the practical adoption of performance evaluation techniques, as it does not tie a developer to "a performance model whose limitations he or she does not understand". However, PUMA is limited by the diversity of proprietary features vendors include in the XMI standard used to serialize UML software design models [WPP+05].

Zhu et al. present a model-driven capacity planning tool suite called Revel8or to ease cross platform capacity planning for component-based applications in [ZLBG07]. The Revel8or suite consists of three tools: MDAPref, which derives QNM from annotated UML diagrams; MDABench, which can generate custom benchmark suites from UML software architecture descriptions (also discussed in [ZBLG07]); and DSLBench, a tool similar to MDABench targeted to the Microsoft Visual Studio platform. The results of the benchmarks may be used as parameter values in performance models, therefore it is of great value having a benchmark that is representative of the application under development. Un-

fortunately, neither [ZBLG07] nor [ZLBG07] includes empirical data supporting the claimed productivity gains, nor are the tools publicly available.

Feedback into software design

The third and final category of automation approaches automatically 'solves' performance problems. For example, utilizing the automatic recognition of performance anti-patterns in software architectures, Cortellessa et al. introduce a framework to automatically propose architectural alternatives [CF07]. The framework employs LQN models to generated the architectural alternatives. The goal of the automation is to relief architects of interpreting the results of performance analysis and the selection of the right architectural alternative based on this analysis. The prototype tool used in [CF07], however, still requires human experience and manual intervention in some steps.

The work by Xu automates the entire performance diagnosis and improvement process with a rule-based system that generates improved models [Xu10]. Using the approach a large number of different design alternatives can be explored quickly. Through the PUMA approach described earlier in [WPP+05] a LQN model is obtained. The models are evaluated with the LQNS tool and rules then generate improved performance models, which can in turn be (manually) translated back to improved software design models. A prototype tool called Performance Booster is evaluated on multiple small (of which one industrial) case studies. Eventually Performance Booster is aimed to be integrated in the PUMA toolset. Current limitations include the disregard of memory size effects, overhead costs of multi-threading, multiple classes of users, and the inability to suggest introduction of parallelism.

Chapter 3

Related work

In this Chapter, we describe the relevant work that has been done by fellow researchers and is related to the topic of this thesis. The discussion consists of three parts. First, we focus on publications that are related to our survey of tools and techniques and our tool selection. Second, we treat studies that discuss systems with architectures that are in some way similar to the RDS. Finally, we discuss case studies that are related for other reasons.

3.1 Survey and Tool Selection

Several recent surveys review performance modeling tools, with a focus on model-based prediction [BDIS04], evaluation of component-based systems [BGMO06, Koz10], and analysing software architectures [Koz04]. An outlook into the future of, and directions for future research in software performance engineering are given in [WFP07]. These surveys differ from this thesis by not studying the applicability of the tools to industrial systems. Jain's 1991 landmark book still provides relevant guidance for selecting and applying performance modeling techniques [Jai91]. The book, however, does not include recent tooling.

Other work evaluates a limited number of tools and mainly focusses on accuracy and speed of these. Balsamo et al. study the benefits of integrating two techniques based on stochastic process algebras and simulation towards performance analysis of software architectures and also discuss the relative merits of both approaches in their own right [BMDI04]. Similarly, Tiwari and Myanampati compare LQN and QPN using the LQNS and HiQPN tools to model the SPECjAppServer2001 benchmark application [TM06]. Meier's thesis compares the Palladio and QPN model solvers after presenting a translation from PCM to QPN. The evaluation is done in several (realistic) case studies and finds that the QPN solver is up to 20 times more efficient [Mei10] [subsequent publication in MKK11].

In an empirical study the effort and industrial applicability of the SPE-ED and Palladio modeling tools is investigated using student participants. The first report on this study finds that creating reusable Palladio models requires more effort, but is already justified if the models are reused just once [MBKR08b]. During later analysis of the empirical data the authors find that the quality of the created models is good and the usability of the tools by non-experts is

promising [MBKR08a]. The thesis of Koziolek includes a study that is similar to that of Martens et al., but is practically superseded by the latter [Koz04].

All above works show trade-off analysis similar to ours, but are often solely of academic nature. In this thesis, we additionally have industrial requirements for selection, e.g., tool stability, user-friendliness, and licensing.

3.2 Web Service Architecture Modeling Studies

A performance modeling tool intended for use early in the development cycle of SOAs is presented in [BOG08]. Among the suggested applications are evaluation of architectural alternatives and capacity planning. The tool produces SOA performance models from existing architectural artifacts (e.g. UML sequence and deployment diagrams), which may significantly reduce the cost of model creation. Brebner et al. later (presumably) extend the same tool and provide examples of the application of the approach [BOG09]. Unfortunately, little details are provided on the tool's capabilities, and to our knowledge the tool is not publicly available. Recently, the authors presented a case study modelling the upgrade of an enterprise service bus (ESB) in a large-scale production SOA [Bre11]. Based on this experience they found that: models do not have to be complex to be a powerful tool in real projects; models can be developed incrementally, adding detail as required; a lack of information on the system can be overcome by building multiple alternative models and selecting the best fitting model; and if not enough information exists to build a comprehensive model covering the entire system, multiple specific models may be build to explore different parts of the system.

Bogardi et al. published a series of papers on performance modeling of the ASP.NET thread pool [BMLC07, BMLCH07, BMLS09, BMLC09] cumulating to [BML11]. In this final publication they present an enhanced version of the mean-value analysis evaluation (MVA) algorithm for queueing networks, which models the ASP.NET thread pool more accurately than the original MVA algorithm. The current practical implications of the work remain unclear. In part, because the modeling of the global and application queue size limits has not been completed yet, while these limits are identified as performance factors [BMLC07].

A model to assess the scalability of a secure file-upload web service is has been constructed in the PEPA process algebra by Gilmore and Tribastone. They can predict the scalability easily for large population sizes without having to construct the state space and thus avoid the state space explosion problem [GT06].

Urgaonkar et al. present a QNM that predicts the performance of multi-tier internet services for various configurations and workloads. The average predicted response times are within the 95% confidence interval of the measured average response times. They also find that their model generalizes to any number of heterogeneous tiers; is designed for session-based workloads; and includes application properties like caching effects, multiple session classes and concurrency limits [UPS+05].

A case study on the SPECjAppServer2001 by Kounev and Buchmann demonstrates that QPN models are well suited for the performance analysis of e-business systems. One of the advantages they note is that QPNs integrate the

modeling of hardware and software aspects [KB03]. Later the study is repeated on the new version of the benchmark application, SPECjAppServer2004, using improved analysis techniques. Then the system is modeled entirely and in more detail, improving the accuracy and representativeness of the results [Kou06b, Kou06a].

Smith and Williams demonstrate their SPE approach and the associated SPE-ED tool in several case studies in their book [SW02], which also provides general practical guidance for the performance engineering process. One of the case studies investigates the performance impact of new functionality on an (imaginary) airline website [SW02, Chapter 13].

3.3 General Performance Modeling Studies

While many case studies in papers presenting tools and approaches are limited to 'toy' problems [e.g., SBC09], some are interesting and close to our work. For example, a three-tier web store modeled in Palladio and then implemented in .Net shows that the Palladio approach might be appropriate for our system [BKR07]. Gambi et al. also model a three-tier web architecture, but do so in LQNs while presenting a model-driven approach to performance modeling [GTC10]. Liu et al. present an approach to model the architecture of component-based systems in QNM and validate their work on an EJB application. They find that performance predictions were often within 10 percent of measurements on implementations (for example systems). What makes their work relevant is that its authors believe the approach is general and could extend to the .Net component technology [LGF05]. Unfortunately, to the best of our knowledge, the work is not supported by tools or extended in later work. Next, Xu et al. developed a template-driven approach for the LQN modeling of EJBs, which they use to correctly predict saturation of CPU and thread-pool resources [XOWM06]. It is suggested that an approach similar to theirs could be applied to .Net, but we have not found any work in this direction. The work by Koziolek et al. shows that model-driven performance evaluation also works for large and complex industrial process control systems [KSB+11]. The thesis of Ufimtsev proposes a framework to model the performance of middleware-based systems, such as J2EE and Com+/.Net, and includes a case study applying this technique to an industry-benchmark J2EE application, ECPerf [Ufi06]. The approach in Ufimtsev's thesis is interesting because it specifically addresses the fact that in web systems as little as 3% of execution time may be spend in 'user' code, while the rest of the time is spend in the application framework's code.

However, all these works have in common that they are limited in detail and chiefly used to demonstrate or showcase a technique. Whereas this thesis investigates *how* the techniques may be used in an *industrial* setting.

Other case studies model industrial systems, which differ significantly from ours: Gaonkar et al. create a simplified model of a Lustre-like file system in Möbius [GKL+09]. Huber's thesis models a storage virtualization system for the IBM System/z experimenting with Palladio outside the targeted modeling domain of business information systems [Hub09] [published in HBR+10]. Rolia et al. show that LQM are more suitable for modeling complex systems than QNM by means of an elaborate industrial case study of the SAP ERP system [RCK+09]. Franks et al. find that analytical results of a LQN evaluation are

within a few percent of simulations in a case study of an air traffic control system of realistic complexity and size [FAOW+09]. Jin et al. also use LQN and predict the scalability of a legacy information system, while presenting their BMM approach to performance prediction for legacy systems that need to adapt to new performance requirements [JTHL07].

Finally, [SMF+07], and [Mon10] provide experience reports based on long term experience [SMF+07], and introduce performance modeling [Mon10]. Unlike this thesis these reports do not show the case studies that led to this experience.

Chapter 4

Performance Modeling Tools

One of the tasks that we have identified in the introduction is to survey the available performance modeling tools. This Chapter reports on the results of this survey, and then discusses our selection of a performance modeling tool to model the RDS. To carry out these tasks we have tried to answer the following questions.

1. What are the common performance engineering tools and their pros and cons?

2. What performance modeling tools exist that are applicable to web service systems?

Various tools to construct performance models exist. During our survey we looked for tools based on several criteria. First, the tool should show applicability to performance modeling of software architectures, this excludes some of the more mature low-level tools. Second, it must be released and available, this excludes many academic tools described in published research. Finally, it should be mature and somewhat stable, again a requirement often not met by tools presented in research publications. We will now briefly motivate why we excluded some of the well known performance modeling tools to illustrate our selection process.

GreatSPN[1] and ORIS[2] are low-level Petri Net modeling tools and do not offer the expressive power of QPNs like QPME. Further, while state space reduction mechanisms are in place, the tools are still sensitive to state space explosion.

ArgoSPE[3] can generate Stochastic Petri Nets in GreatSPN format, but aims to ease the modeling of software systems by offering a UML 'interface'. Unfortunately, it is based on an obsolete version of ArgoUML. Furthermore, the advantage of the use of UML is limited by the sparse support for UML constructs. For example, there is no support for component diagrams. Finally, judging by the project home page, the tool doesn't seem to be actively maintained.

[1] GreatSPN: http://www.di.unito.it/~greatspn/index.html
[2] ORIS: http://www.stlab.dsi.unifi.it/oris/
[3] ArgoSPE: http://argospe.tigris.org/

TwoTowers[4] isn't actively updated either. The last new version dates five years back. The specification of models is cumbersome because of the textual notation used. Finally, the approach suffers from the state space explosion problem, limiting model scalability [BMDI04].

Hyperformix[5] promises to be an all-round scalability solution including load testing, instrumentation, measurement, and model tooling. However, the tool is very expensive and the exact features and supported use cases are not clear. Furthermore, specific tooling for load testing, and instrumentation and measurement was already acquired by ABB when the thesis work commenced.

PEPA tools[6] offers a powerful process algebra. Comprehensibility of process algebras and mapping to software architecture concepts is, however, problematic. In addition PEPA tools uses a textual notation and a solver that is less efficient than LQNS, which also has a textual notation but offers more natural mapping of concepts.

SHARPE[7] has not been updated in 10 years. And does not provide architectural support, but only low-level modeling.

Finally, PRISM[8] only has support for Markov chains, which makes it difficult to model queueing effects.

In the end we selected six tools that seemed mature enough and promised to fit our architectural modeling problem. These tools will be described in more detail. The discussion in this Section will be rather neutral, but it forms the foundation for our tool selection in Section 4.7. For information on other tools, recall from Section 2.5 that surveys of the state of the art may be found in [BDIS04, BGMO06, Koz10] and [Koz04, Appendix A].

A quick overview of the considered tools is included in Table 4.1 on page 26. Each tool is discussed in more detail in the following sections. The discussions start with a short introduction and then contain the following parts:

1. Advantages – positive reasons for selecting this tool.
2. Inputs – specific input requirements for the model and modeling process. Of course, all tools will require an architecture to model, performance goals, and performance measurements to instantiate model parameters on top of these.
3. Time required – qualitative assessment of the required effort for modeling.
4. Accuracy – qualitative assessment of the quality of the predictions made by the tool.
5. Assumptions – important assumptions made by the tool, e.g., limiting its applicability.
6. Limitations – reasons for not selecting this tool; besides timeliness we also wish to model memory and storage requirements of the system, support for this is also indicated here if present.

[4]TwoTowers: http://www.sti.uniurb.it/bernardo/twotowers/
[5]Hyperformix: http://www.hyperformix.com/
[6]PEPA tools: http://www.dcs.ed.ac.uk/pepa/tools/
[7]SHARPE: http://people.ee.duke.edu/~kst/software_packages.html
[8]PRISM: http://www.prismmodelchecker.org/

Name	License	Software Model	Performance Model	Analysis/ Simulation	Studies	Case Studies	Status
Java Modeling Tools	GNU GPL	-	QNM	both	[BC09, SBC09, CSM11]	[SBC09, RCK+09] (QNM related: [UPS+05]	stable
Layered Queueing Network Solver	Evaluation only	(text)	LQN	both	[Woo02, FMN+96]	[TM06, JTHL07, GTC10] (related: [Xu10])	stable
Palladio-Bench	Free download	Palladio Component Model (PCM)	PCM or LQN	both	[BKR07, MBKR08a, MBKR08b, BKR09, BKK09]	[Hub09, HBR+10, KSB+11]	stable / mature
Möbius	Commercial for non-academic use	SAN, Buckets & Balls, PEPA-nets	stochastic extensions to Petri Nets, Markov chains (+extensions), and stochastic process algebras	simulation, analysis for some	[CGM+07, CGK+09, San10, GKL+09]	???	mature
SPE-ED	Commercial	MSC, Execution graphs	QNM	both	[Smi86, WS98, SW02, MBKR08b, MBKR08a]	[WS98, SW02]	stable / mature
QPME	Open source	-	QPN	simulation	[KSM10, KDB06, KB06]	[KB06, Kou06a, Kou06b, KB03]	academic / stable (?)

Table 4.1: Overview of considered tools.

The amount of information on each of these subjects varies for each tool and is dependent on what we have found in literature. As the amount of time available for literature study was limited, some items will contain only educated guesses: the next Sections show precisely the information we based our tool selection on.

We suggest that the sometimes poor detail, that we were able to provide within limited time, is also an important indicator of the state of research. It is difficult for industry to quickly take an off-the-shelve tool to solve their problems: expertise is required, which potentially reduces the adoption of performance modeling in industry.

4.1 Java Modeling Tools

The Java Modeling Tools (JMT) is a suite of Java programs supporting the modeling and analysis of QNM. It may be downloaded under the GNU GPL at: `http://jmt.sourceforge.net`

4.1.1 Advantages

- Obtaining solutions to QNM is computationally cheap [RCK+09].
- JMT has an extensive GUI abstracting technical details, including wizards to guide the users through the process, which makes the applications accessible to inexperienced users and lowers the learning curve [SBC09].
- The robustness of predictions may be validated using multiple tools from the suite, since they offer overlapping functionality [SBC09].
- JMT offers "pre-processing of log traces, [and] clustering algorithms for selection of the most significant workloads to be modeled" [BC09].

4.1.2 Inputs

Knowledge about queueing/delay aspects of the system.

4.1.3 Time required

The modeling effort seems to be low, because of the claimed quality of the user interface [SBC09] and the relative simplicity of the QNM concept. We did not find concrete information on the time needed for model execution. Serazzi et al. do confirm that the mean-value analysis is faster than simulations in most cases [SBC09]. We estimate that the analytical algorithms to take no more than a couple of seconds and that simulations execute in maximal two orders of magnitude more time.

4.1.4 Accuracy

The work by Rolia et al. suggests LQN are more appropriate for modeling software systems than QNM [RCK+09]. They find that QNM do not offer the features necessary to predict mean response times within 15% of measured values in their case study.

4.1.5 Assumptions

- Response times are highly dependent on the queueing behaviour in the system.
- The software architecture can 'easily' be represented as a flat network of queues (i.e., mixing abstraction levels does not complicate the model too much).

4.1.6 Limitations

Rolia et al. note that it is challenging to accurately model industrial software systems using the simple QNM formalism [RCK+09]. More specifically, they note that software threading levels, asynchronous database calls, priority scheduling, multiple phases of processing, and the parallelism offered by multi-core processors cannot be captured in QNM, while they all have a significant impact on performance [RCK+09].

4.2 Layered Queueing Network Solver

The Layered Queueing Network Solver (LQNS) is a LQN tool, and has the most complete feature set of several common LQN tools [FAOW+09, Table 5]. It is available from the authors under an evaluation license from:
`http://www.sce.carleton.ca/rads/lqns/`

4.2.1 Advantages

- LQNs offer an easy way to express resource usage and to find bottlenecks in software servers [Xu10].
- The analysis algorithms for LQN scale up to large systems [Xu10] and can be used to model a large number of concurrent requests [TM06].
- LQN support both open (e.g., batch workload) and closed requests (e.g., interactive users). [TM06]
- LQN can model all of synchronous, asynchronous and forward calls. Enabling the modeling of messaging systems using LQN. [TM06]
- "LQN model consists of convenient primitives notations which makes LQN construction simple, convey more semantic information and guarantee that these models are well-formed (i.e. stable and deadlock free) [DONA1995 (citation theirs)]." [TM06]
- A 'second' execution phase may be modeled in LQN, this part of the execution is carried out autonomously by the server after replying to a incoming request (e.g., post-processing) [Woo02]. [Woo02]

4.2.2 Inputs

To construct a LQN the information itemized below is required. Each item maps almost one-to-one to a part of the LQN model.

- list or model of software components,
- service time at each component,
- resources (e.g., threads, cpu, disks),
- multiplicity of resources (i.e., how many of each),
- a mapping of components to these resources,
- scheduling policies for all components,
- tasks and their priorities,
- call paths (i.e., interaction between components),
- number of calls to a service (i.e., probability a call path is taken).

4.2.3 Time required

There is some evidence that LQN models are relatively easy to construct. Jin et al. report that they spent more time to identify what parts of their application to monitor and model than to construct the model and run the experiments [JTHL07]. In a direct comparison between the LQNS and HiQPN tools the former is mentioned as quicker to use and the evaluation of its models as less resource intensive [TM06].

The LQNS contains an analytic solver and a simulator. Obviously, the analytic solver is quicker. Franks reports on a case in which the analytical solver outperforms the simulator 15 to 40 times when the simulator is configured to provide results with a 95% confidence interval [FAOW+09].

4.2.4 Accuracy

While discussing the JMT we already quoted [RCK+09] suggesting that many of the LQN features are needed to achieve good accuracy. This study used the MOL tool (*not* part of LQNS) to find errors of 15% and 24% depending on the case, also noting a slight pessimism of the predictions in some situations. Best accuracy was achieved for CPU utilizations of less than 76%, which is considered a high utilization for enterprise application servers [RCK+09].

The aforementioned pessimism is also noted by Xu et al., who find a 6% difference between a LQN model solution and a (supposedly more detailed) simulation of an EJB application [XOWM06].

In their study comparing LQN and QPN Tiwari et al. find that the prediction errors are "within the acceptable range" except for one case where they find an error of 15%. They do note that the accuracy of the software contention results of their LQN model are lower than those of obtained with the QPN model [TM06].

Modeling a complex legacy system Jin et al. report that the solver results closely match measurements after model calibration [JTHL07]. For example,

they were able to predict throughput performance degradation within 8% of benchmarks.

Finally, Franks et al. were able to obtain results with an error of less than 2% in a case study on a complex air traffic control system. In some situations they reported errors of up to 17%, but these numbers are very acceptable for early evaluations [FAOW+09].

4.2.5 Assumptions

- The system being modeled is a parallel/distributed system that cannot easily be captured in an ordinary queueing network.
- Parallel/distributed system software can be abstracted into layers, where the performance of higher layers depends on the performance of the lower layers.

4.2.6 Limitations

- According to Tiwari et al. LQN are less suited to model memory size constraints than QPN [TM06].
- The user manual lists several known defects, but little details are provided. Some examples [FMW+05]:
  - Chain construction for models with multi- and infinite-servers.
  - No algorithm for phased multiservers OPEN class.
  - Need to implement queue lengths for open classes.
- The graphical model editor is not documented [Woo02].

4.3 Palladio-Bench

The Palladio-Bench implements the Palladio Component Model (PCM) described in Section 2.4.4. The software is released as an Eclipse plug-in and can be obtained via the tool's website: `http://www.palladio-simulator.com/`

4.3.1 Advantages

- Potential time-saving when modeled components are re-used in new models [MBKR08b].
- It is possible to specify input dependent resource demands, branch probabilities and loop iterations counts [BKR07]. For example, the execution time of a component can be made dependent on the size of a file it has to process.
- Integration with supplementary tools. Notably the design space exploration tool PerOpteryx.
- Code generation for performance prototypes.

4.3.2 Inputs

Similar to those of the LQNS (see page 29).

4.3.3 Time required

The effort required to construct a PCM and the accuracy of these efforts have been empirically studied using student participants by Martens et al. [MBKR08b, MBKR08a]. The work seems to suggest that once very basic familiarity with the tools and the underlying techniques has been acquired, a simple component based system may be modeled within 7 hours. After modeling a complex storage virtualization system, Huber et al. estimate that if one is unfamiliar with the PCM and the system, four person months are required to create, calibrate, and validate a PCM performance model of such a system [HBR+10]. Here it should be noted that Huber et al. used the PCM to model a system that is not a component-based business application, which is what the PCM was designed to model.

We return to the studies of Martens et al. in which they compare Palladio-Bench to the SPE-ED tool (discussed later in Section 4.5). It was found that participants who used Palladio-Bench spent more time searching for errors (i.e., fixing wrong or missing parameters) in their model than those who used SPE-ED (20% vs. 2-6% of total time) [MBKR08b]. Also, less (usability) problems were experienced by the SPE-ED participants, on average only 0.24 problems were reported per participant, versus 2.27 problems on average for Palladio-Bench participants [MBKR08a]. Not surprisingly then, predictions were done much faster using SPE-ED. The Palladio-Bench participants needed a factor 1.17 to 2.05 more time to do predictions, varying with the type of system and modeling task. However, in a more realistic setting where several alternatives had to be compared, predictions using Palladio were done only 1.17 to 1.32 times slower [MBKR08b].

4.3.4 Accuracy

The prototypical version of Palladio-Bench was already able to select the better architectural alternative [BKR07]. In Huber's case study on a storage virtualization system prediction errors stayed below 21% during model validation. This, despite the system being akin from the business information systems targeted by PCM [HBR+10]. Further, Koziolek et al. report that the prediction error of SimuCom (Palladio's simulator) is similar to that of the LQNS simulator [KSB+11].

The empirical study of Martens et al. finds that the mean response time predicted by the student participant models deviates on average 6.9% from their reference model, which lies within their limit of 10%. For some models the deviation was higher, but 85% of the participants was able to identify the correct design decisions. Participants in this study who used SPE-ED were more successful in identifying design decisions with 97% selecting the right decision. However, SPE-ED participants had 44% of the errors they made still in their final model, whereas Palladio-Bench participants only had 11% of their errors in their final model [MBKR08b]. The consistency checks Palladio-Bench makes

on the constructed model seem to make its predictions more reliable than those of SPE-ED.

4.3.5 Assumptions

- The system being modeled is a component-based, business information system [HBR+10].
- A user cannot execute more than one task at a time [BKR07].
- The architecture and component deployment is static. That is, the architecture or deployment of components does not (automatically) change at runtime [BKR07].
- The behaviour of the system is independent of the state of components or the runtime environment, i.e., components and their execution environment do not change QoS properties at runtime [BKR07].
- "It is assumed that the necessary model information like service effect specifications and parametric dependencies are available and have been specified by the component developer, system architect, system deployer and domain expert. We also assume that different component developers are able to agree on common parameter abstractions." ([BKR07])
- PCM makes some mathematical assumptions to reduce model complexity and thereby simulation run length and memory consumption. For example, the resource demands are assumed to be stochastically independent. [BKR09]

4.3.6 Limitations

Storage capacity requirements can probably not be modeled using the PCM. This functionality is not mentioned in [BKR07] when resource types are discussed. Later, Becker et al. report that support is limited to CPU demands for the moment [BKR09]. Huber et al. experience other modeling limitations while modeling a storage virtualization system. They had to implement workarounds for queue blocking and throughput constraints [HBR+10]. Becker et al. also provide lists of limitations in [BKR07, BKR09], we extracted the following points:

- Support for concurrency is limited. Predictions can especially be off for multi-core processors, for example because of the effects of CPU caching behavior in these system not being captured [BKR07].
- Only the processing rate can be modeled for processing resources, while more factors are important in modern CPUs, for example the pipelining and caching strategies [BKR07].
- The support for modelling the runtime environment is limited. For example, the middleware platform and its configuration can have a significant influence on performance [BKR07].
- The effect of memory bus contention and memory allocation and deallocation are not captured [BKR09].

4.4 Möbius

Möbius is a tool framework that implements multiple modeling languages and even allows to combine them to exploit their expressiveness or to combine multiple aspects of a system (e.g., performance and reliability) into one model [CGK+09]. Of the formalisms offered, the SAN, Buckets and Balls and PEPA modeling languages can be used for performance modeling. It is claimed that there are 30 industrial users of Möbius [GKL+09], which has its homepage at: `https://www.mobius.illinois.edu/`

4.4.1 Advantages

- Möbius allows to compose models to ease the modeling of complex systems: "Often, a modeler will create several atomic models that represent sub-systems of the total system. The Möbius composition formalism provides the flexibility and simplicity for the modeler to mix and match the atomic and other composed models together to build larger and more sophisticated models" [GKL+09].
- Simulations can be run in parallel distributed across multiple local (i.e., additional CPUs or cores) or remote processors (i.e., networked machines) [GKL+09].
- "Möbius provides extensive support for the analysis of results, which are either integrated into the Möbius framework work or as external tools. Möbius has integrated database support to add results from solutions generated from experiments solved using numerical or simulation techniques into an external SQL database. The result include the model parameters, experiment parameters such as batch sizes, and time of execution, and other related information." [CGK+09]

4.4.2 Inputs

Depends on the modeling formalism used, see [San10] for details.

4.4.3 Time required

We did not find any concrete indication of the amount of work required to construct models in Möbius, and did not study all supported modeling formalisms in sufficient detail to make an estimate. We found some evidence that the user has to write a segment of C++ code to define how performance variables should be collected [GKL+09], which seems laborious.

4.4.4 Accuracy

No reports on the accuracy of predictions made through Möbius are included in this study.

4.4.5 Assumptions

No information found.

4.4.6 Limitations

Multiple formalisms supported by Möbius rely on Markov chains and hence suffer from the state space explosion problem. There might be a reduction of the number of states by one to two orders of magnitude using lumping techniques [CGK+09], but what this means in practise is unclear.

4.5 SPE-ED

The SPE-ED tool supports the SPE approach documented by Smith and Williams [Smi86, WS98, SW02]. The software may be purchased through the website: `http://www.spe-ed.com/sped.htm`

4.5.1 Advantages

- Adapted for use by software architects in [WS98].
- Simple models, that are easy to construct and solve [WS98] (empirical support in [MBKR08b]).
- One of the most mature tools suitable for early performance predictions [MBKR08b].

4.5.2 Inputs

The use case architectural view is central in the SPE approach [WS98, pg. 6] and [SW02]. Scenarios (use case instances) describe the system components, their interactions, and the interactions the system has with the environment, all of which are needed for model construction. The SPE approach also uses information from the other architectural views that are part of Kruchten's 4+1 model: the logical, process and physical view [WS98]. The logical view helps to derive the resource requirements. Resource requirements are an estimate of the service needed from a particular resource in each step. The process view shows which components reside in which processes or threads and how these components communicate (e.g., synchronous or asynchronous). The physical view gives information about the hardware that executes the software processes.

4.5.3 Time required

The work by Martens et al. suggests that a simple component based system may be modeled within 7 hours in SPE-ED by their student participants with little experience, and that it is 1.17 to 2.05 quicker to model the system in SPE-ED than using Palladio-Bench. The participants of their study experienced little problems using the tool, but the resulting models were more error prone than PCM models [MBKR08b, MBKR08a]. Unfortunately, we do not have further information on the required modeling effort.

4.5.4 Accuracy

Of the literature that we studied, only the work by Martens et al. reports on the accuracy of predictions made using the SPE-ED. Their findings are included in Section 4.3.

4.5.5 Assumptions

SPE-ED uses QNM as its underlying performance model representation ([WS98]) thus it inherits the assumptions for QNM. No other assumptions were made explicit in the studied literature.

4.5.6 Limitations

We cannot find clear evidence of support for modeling memory and storage requirements in [SW02]. Timeliness of both memory and storage devices (e.g., the amount of seconds a disk request needs to be serviced) can however be modeled.

Martens et al. give us two other limitations. First, SPE-ED supports only M/M/n queues [MBKR08a], which means that inter-arrival and service times can only be modeled as having an exponential distribution. Second, SPE-ED only reports mean values [MBKR08a], which is an important limitation, for one could wish to base decisions on extreme values. For example, if the worst predicted performance value is good enough the modeled alternative can be accepted without doubt.

4.6 QPME

According to the 'Petri Nets World' website[9] QPME is one of the few Petri Net tools that supports advanced performance analysis, and it is the only current tool that supports hierarchical QPN. The QPN modeling formalism was shown to be well-suited to represent distributed e-business systems in [KB03], but lacking good tooling, which motivated the development of QPME. QPME is available through: `http://descartes.ipd.kit.edu/projects/qpme`

4.6.1 Advantages

- QPN show both the hard- and software behaviour of a system [KB03].
- QPN can represent simultaneous resource possession, blocking and contention for software resources (i.e., threads, connections and processes) [KB03].
- QPME takes care to reduce the state space explosion problem [KB06].
- In QPME multiple queueing places may share the same 'physical' queue. "This allows to use queuing places to represent software entities, e.g., software components, which can then be mapped to different hardware resources modeled as queues." [KSM10]

[9] `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html`

- QPNs are better able to model synchronization than LQN [KSM10].
- The modeling of departure disciplines ensures that tokens are discriminated based on the order they arrive in the depository [KSM10].
- QPME features integrated data analysis and visualization [KSM10].

4.6.2 Inputs

The following inputs were explicitly identified:

- list of software components,
- component interactions
- list of system components (e.g., cpu, disks),
- description of workload classes
- processing steps
- description of resources used by (software) components

4.6.3 Time required

We did not find concrete information about the time required to construct a QPN model with QPME in the studied literature, but Tiwari et al. find that the construction of a QPN model in another QPN tool, HiQPN, is more laborious than the construction of a LQN model [TM06].

For evaluation of the models, [KB06] provides a concrete indication of the running time of QPME simulations of a rather complex model of the SPECjAppServer2001 benchmark application. The study finds that in a moderate-load scenario with six application servers the average runtime on a 2GHz CPU was about 5 minutes (over 500 simulation runs). Running the same simulation for a heavy-load scenario resulted in an average runtime of 12 min per run. Simpler models are solved faster with simulation runs taking under a minute [KB06].

4.6.4 Accuracy

We only have accuracy figures from two publications in 2006 by the authors of QPME. We expect that the results in both publications come from the same experiment. They found similar results validating several configurations, one concrete example follows:

> The maximum modeling error for throughput is 9.3 percent, for utilization, 9.1 percent, and, for response time, 12.9 percent. Varying the transaction mix and workload intensity led to predictions of similar accuracy. Since these results are reasonable, the model is considered valid. ([Kou06b] and [Kou06a])

4.6.5 Assumptions

When modeling the SPECjAppServer2004 Kounev et al. made several assumptions and simplifications they claim are typical [Kou06a]. We suspect that they may also be required to limit the model complexity of this realistically sized benchmark system. The assumptions are as follows:

- Servers are assumed to be visited only once for each sub-transaction and during this visit sub-transactions receive all of its service demands.
- It is assumed that when a sub-transaction is serviced it visits all servers resources only once.
- The total service demand of a transaction at a given system resource is assumed to be spread evenly over its sub-transactions.

4.6.6 Limitations

- QPN are (at least to a certain extend) able to model memory constraints [Kou06b].
- QPNs are limited to simulation, i.e. no analytical techniques are available [KSM10].
- Simulations cannot be distributed across systems or cores [KSM10].
- It is not possible to specify load-dependent service demands [KSM10].
- QPN are restricted to modeling closed requests (e.g., interactive users) [TM06]
- QPN are limited to modeling synchronous calls and are thus less suitable for modeling message passing systems [TM06].
- At the moment QPME does not support timed transitions, but they may be approximated using a serial network of an immediate transition, a queueing place and a second immediate transition [KSM10].

4.7 Selecting a Modeling Technique and Tool

We claim that selecting a modeling technique and selecting a modeling tool cannot be seen as disconnected problems. Tool limitations and features influence the expressive power of the modeling technique, and the tool is thereby of influence on the applicability of a modeling approach. In the end, selecting the wrong tool will result in more problems when applying it [MBKR08a].

We already started the selection process at the beginning of this Chapter. Up to now the main criterion was applicability to performance modeling of software architectures. In this Section, we go deeper and we describe the selection of the Palladio-Bench modeling tool. First, we list our requirements with respect to the tool. Then, we describe the elimination process that we used to select among the six tools, which were discussed in the previous Sections.

4.7.1 Requirements

A number requirements were used to evaluate the surveyed tools and make our final decision. While no formal process was followed to establish and record the requirements to the performance modeling approach, some can be made explicit and are listed below. Another important factor in the selection process was the expert opinion of Heiko Koziolek (e.g., [Koz10]), who had a consulting role in the CRC project and thesis work.

- Modeling of software architectures must be supported by a straightforward mapping of software architecture concepts onto the modeling concepts provided.
- Support for the modeling of large and complex systems.
- Preferably the architectural trade-off process is explicitly supported.
- Scale-out scenarios to many core CPUs should be easy to model, without significantly reducing the accuracy of the results.
- It should be possible to model asynchronous web service calls.
- It must be possible to create models with sufficient prediction accuracy to make an architecture trade-off.
- It should be possible to analyze or simulate workloads with thousands of concurrent requests.
- Evaluation of the model must provide metrics indicating for each architecture the maximum number of devices, internal users, and the amount of diagnostic data that can be collected.
- Proven stability and preferably prior industrial case studies are available.
- Some form of support and regular bug fixes are provided by the tool developer.
- Easy to learn: a user should be able to use the tool with only basic knowledge of performance modeling.
- Easy to use: multiple versions of the model can be maintained in parallel and minor architectural changes can be modeled and tested within several minutes.

4.7.2 Rationale for selection

With the requirements laid out, we now first give our reasons for selecting the Palladio-Bench modeling tool. Second, we explain why we eliminated the other five tools that were discussed in the previous Sections.

Palladio-Bench supports the simulation of architectural models and the component-based philosophy that it employs makes that models are easily constructed and easy to understand. The Palladio-Bench GUI supports this philosophy by offering a 'UML-like' interface. The ability to re-use models and components is another very useful feature for our study of multiple architectural alternatives. Moreover, Palladio-Bench is at version 3.2 and has been used

in industrial case studies before, thus we assume that it is mature and sufficiently stable. Finally, Palladio-Bench is free and open source software, which makes it easy to obtain. We will now describe where other tools failed to meet our needs.

Initially, we had a strong preference for an analytical and non-Markov chain-based tools, because even recent publications [RCK+09, e.g.] still suggest that Markov chain approaches are not suitable for modeling large systems. This was an argument against the use of many formalisms supported by Möbius. We were further discouraged to use Möbius by that fact that none of the example applications for Möbius in [CGK+09] resemble our case. It seems that Möbius is not targeted at software architecture performance modeling.

The SPE-ED tool part of Smith and William's software performance engineering (SPE) method has been in widespread use and specifically targeted at early design and architecture modeling. It does, however, not seem to receive many major updates anymore.

Both Möbius and SPE-ED were ruled out based on the above arguments, and the fact that both are subject to license fees for commercial use. SPE-ED claims to be 'the software performance engineering tool at a remarkably low price'[10], but no price information is published. Regardless of cost, the acquisition of commercial software within ABB would have considerably delayed the thesis work, which was not deemed acceptable.

Ruling out SPE-ED was made easier by the availability of Java Modeling Tools (JMT), because it also implements QNM. While we were doubtful whether QNM could model software architectures in sufficient detail to be useful for a trade-off analysis of the alternatives, JMT offers the advantage of load dependent servers. We hoped to use these to model premature thread termination behaviour we had observed during our load tests. Further, the QNM formalism is relatively easy to understand and simple to use.

With all four remaining tools all having their advantages and disadvantages, we started a modeling effort using JMT because of its simplicity and because it could be downloaded directly. Unfortunately, it quickly proved to be not expressive enough. For example, asynchronous behaviour could not be directly expressed. Also the semantic gap between our software design of components interacting by web services calls and the QNM formalism concepts formed an obstacle.

In the mean time we had contacted the authors of QPME to obtain their tool, but at the time only QPME 1.5 was released. This meant that we could not use the improvements made in version 2.0, which we had assumed to be available when reviewing QPME in Section 4.6. Leaving us with just two tools to choose from: LQNS and Palladio-Bench.

Both LQNS and Palladio-Bench offer modeling concepts that are easily mapped onto software modeling concepts. The alert reader will notice that our expert consultant, Heiko Koziolek, is one of the authors of the Palladio tools. While we acknowledge this might have introduced bias, we would like to point out that dr. Koziolek also has expertise with many other tools including the LQNS, and that we plan to use LQNS in our future work.

The main disadvantage of Palladio-Bench is the limited model analysis tool, and thus being forced to run potentially long simulations. But LQNS had a

[10] source: http://www.perfeng.com/sped.htm

stronger disadvantage. LQNS is only available under a license, and getting a license agreement would have taken too much time in a situation were we already lost time because we switched tools.

Chapter 5

Approach

In this Chapter, we describe our approach to the performance modeling problem, thereby adding some detail to the task outline of goal one in Chapter 1. First, we will describe two common performance engineering processes and discuss the scope of the thesis in terms of these processes. Second, we outline the plan for our performance modeling case study.

The thesis assignment provided a clear separation of concerns. The CRC project would study RDS for bottlenecks, establish requirements, develop architectural alternatives and perform measurements. In the thesis, a modeling approach would be selected and then applied in a case study. In other words, the CRC project is the larger performance engineering (or performance evaluation) study and the thesis work takes care of the modeling within this study. So the next Section on performance engineering processes will show the bigger picture and how performance modeling fits in, whereas Section 5.2 describes our approach to creating the performance model. Finally, we would like to note that while the CRC project tasks are not described in detail in this thesis, obviously the thesis work did influence the CRC project tasks and vice-versa.

5.1 Performance Engineering Processes

In Chapters 2 and 4, we focussed on tools and techniques, but these are clearly only part of the performance engineering effort. The process that is followed to conduct a performance engineering study is just as important. We referenced two works during our endeavour. Smith and Williams describe a process as part of their software performance engineering (SPE) approach [SW02], and Jain provides steps for performance evaluations and for capacity planning [Jai91]. We will now introduce these processes and describe how the work described in this thesis fits in these processes.

The steps Jain identifies for a performance evaluation study are listed in Figure 5.1. We feel that steps 3–5 need some clarification. First, the difference between metrics (step 3) and parameters (step 4). Metrics describe what performance aspects of the system you will measure during the performance study (e.g., response time). Whereas parameters tell what things influence system performance, for example the mix of user requests. Jain calls a parameter a factor, if you wish to vary its value during the study (step 5). Because in practise

1. *State the goals of the study and define the system boundaries.* (Chapter 1)
2. *List system services and possible outcomes.* (Section 6.1)
3. Select performance metrics. (Chapter 6+7)
4. *List system and workload parameters.* (Section 6.2)
5. *Select factors and their values.* (Chapter 6)
6. Select evaluation techniques. (Section 4.7)
7. *Select the workload.* (Section 6.2+6.3)
8. Design the experiments. (Chapter 7)
9. Analyze and interpret the data. (Chapter 7)
10. Present the results. Start over, if necessary. (Chapter 7)

Figure 5.1: Steps for a performance evaluation study [Jai91, Box 2.2]

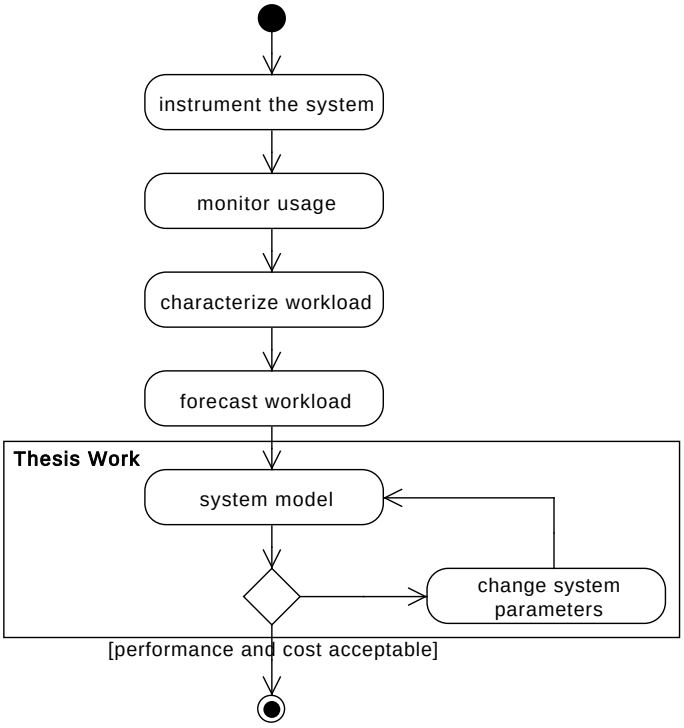

Figure 5.2: Steps in capacity planning process [Jai91, Figure 9.1]

generally not enough resources are available to study the effect of all parameters, only parameters with high impact on performance should be selected as factors.

The references between parentheses behind the steps in Figure 5.1 refer to the part(s) of the thesis that describe the result of that step. Steps printed in *italics* are not part of this thesis. These steps were taken by the CRC project team or by the CRC project team in collaboration with the author. Step 1 was given in the thesis assignment. Step 2 resulted in an architecture reconstruction effort by document and source code analysis and tests. Both these tasks were carried out by the CRC project team. Step 4 and 5 were a shared effort. We provide limited rationale for the selected factors and not all parameters are discussed in detail, for the system is simply to complex to do so in this thesis. Step 7 was defined in the goal of the CRC project (obtain a one order of magnitude speed-up). However, to find the maximum capacity for each of the alternatives, higher and lower workloads than the target capacity workload were used.

Jain also specifies a process for capacity planning studies, which is shown in Figure 5.2. The structure of the CRC project surrounding this thesis to some extend follows this process, for it took all of the steps in it. The flow in Jain's process also shows the place of the thesis, both in the process and in the project: several things were already done before the thesis work was started. System instrumentation was done using the Dynatrace performance monitor tool ([dyn11]) and offered a wealth of information on the behavior of the RDS. Usage monitoring and workload characterization was done by log file analysis and enabled our performance modeling study. And finally, workload forecasts were provided as part of the CRC project plan to motivate the need for architectural redesign.

The SPE process of Smith and Williams is illustrated in Figure 5.3 and again the scope of the thesis work is indicated. For the steps that are part of the thesis: the entire process of construction and validation of the performance models is described in Chapter 6, and the evaluation of the models is the subject of Chapter 7. Below we briefly discuss the out-of-scope steps.

The performance risk is identified and even experienced in the production environment. This motivated the CRC project and the performance modeling effort documented in this thesis. The critical use cases and key performance scenarios were identified by the CRC team and are discussed in Section 6.2.1. The performance objectives are specified in the growth scenario provided by the ABB business unit. The main performance objective is summarized in goal 1 in the introduction (page 2). The remaining three steps (those in the rightmost 'column' of Figure 5.3) are not directly applicable; if for a certain alternative the maximum capacity had been determined either the process stopped or the architecture was changed to increase performance, if the changes were substantial the resulting architecture was included as a new alternative.

5.2 Performance Modeling Plan

In the processes we described in the last Section, neither Smith and Williams, nor Jain provide detail on how to construct performance models or how to use performance modeling in architectural redesign. In their respective books both Smith and Williams and Jain, of course, provide guidance on how to use the performance modeling technique of their choice, but the PCM is slightly dif-

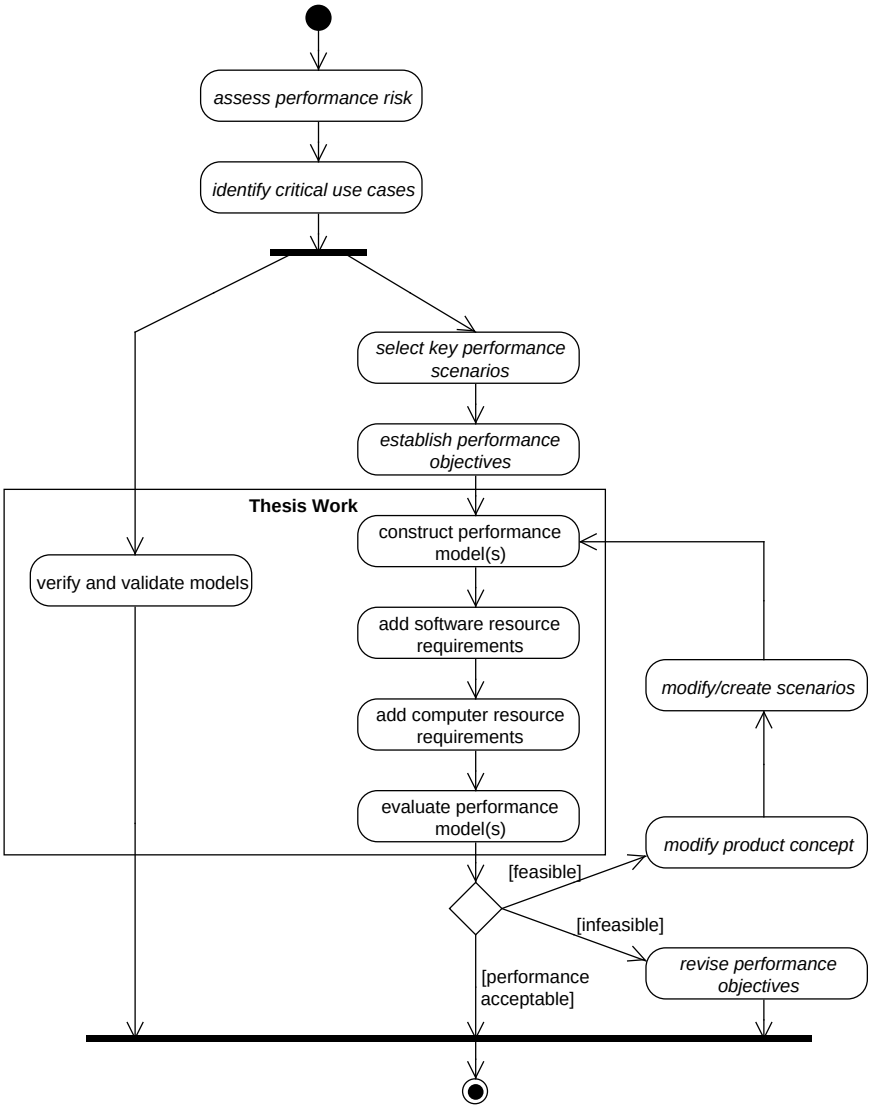

Figure 5.3: The work described in this thesis as part of the SPE process for object-oriented systems [SW02, Figure 2-1]

ferent from these. Therefore we will now describe how we planned to create the performance model(s), and how to use performance modeling in our architectural redesign project. First we recall some of the tasks we outlined in the introduction and use them to structure our discussion:

4. Create a performance model of the current implementation of the RDS.
5. Obtain performance measurements of the current implementation to validate the created model. (CRC project team)
6. Validate the created model using measurements on the current implementation obtained through experiments.
7. Predict the maximum capacity of the current system and alternative architectural designs.

Task 4: create a performance model The first step was to capture the existing system in a performance model, so that we could modify this model to capture the alternative architectures. We call the existing system the baseline (system) and its model the baseline performance model. To model the baseline architecture, we first studied its behavior by analyzing the available documentation and the load tests carried out by the CRC team. A major challenge was that the existing architectural descriptions were limited, but the load tests helped here because the Dynatrace performance analysis tool can create sequence diagrams of transactions and offers method-level performance information for each transaction in so-called purepaths as illustrated in Figure 5.4. Based on this we constructed an initial model using the order, modeling approach, and advise given in the Palladio tutorial videos [Bec], as follows:

1. Create a component repository of components and interfaces.
2. Specify the behavior and the service demands of the components.
3. Using the components from the repository create a system model.
4. Model the resource environment consisting of servers, CPUs, network resources, etc.
5. Specify the deployment of the system in the resource environment in a deployment diagram.
6. Create a usage model specifying the workload.

Task 6: validate the performance model The reason that we first mimicked the baseline is that this allowed us to calibrate the performance model against it by comparing the predictions made by the model for a certain workload to the measurement data from Dynatrace. During calibration the initial model was iteratively improved by detailing or adjusting the parts which showed errors of more than 20 % compared to measurement results. The model was calibrated against three different workloads. These workloads differ in intensity and are discussed further in Section 6.3.1. In retrospect the calibration effort has proved to be a very useful experience, because it helped us to learn more about the RDS architecture, the system behavior, and bottlenecks in the system.

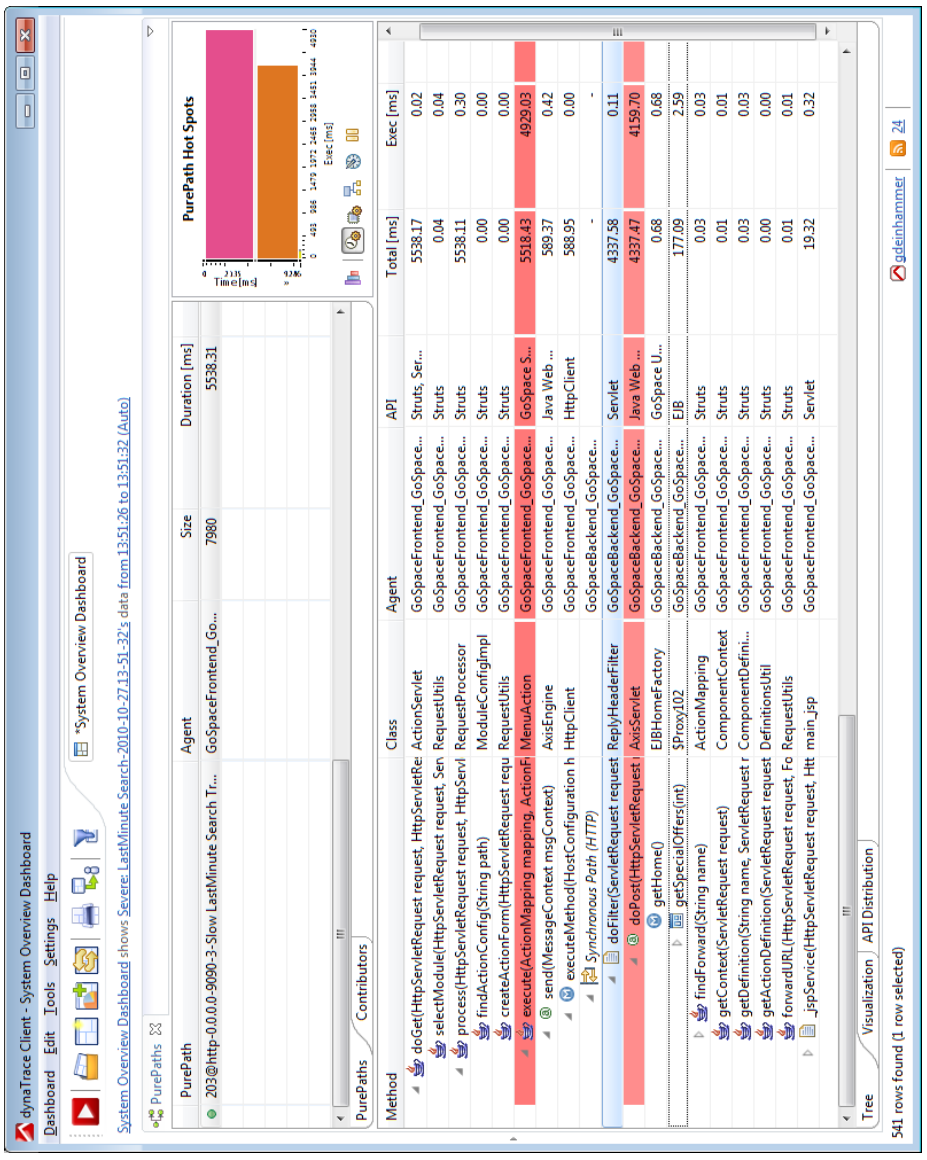

Figure 5.4: Dynatrace showing a generated Purepath for a distributed Java application (taken from [dyn11])

Task 7: make capacity predictions Once we had a calibrated model, we had components that matched those in the RDS and we could change the system architecture and the deployment to explore the alternative architectures. During a workshop four initial alternative architectures were developed (number 2 through 5 in Chapter 6). Later new insights resulted in more alternatives and we went through several iterations of alternatives. Our redesign exploration can be summarized as follows.

1. Model the alternative architectures
2. Find the capacity limits for the alternative architectures
3. If needed, tune the alternatives (e.g., by balancing the CPU configuration of the servers) or develop further alternatives, and go back to step 1.

Chapter 6

Architectures and Performance Models

In this Chapter, we present the architectural alternatives and we discuss how we constructed their performance model counterparts. We briefly motivate the alternatives and indicate how the performance modeling results steered the development of further alternatives. At the same time however, we remind the reader that architecture design is largely outside the scope of this thesis. Also, please refer back to Section 2.1, for we will refer to the three axes of scale in this Chapter. We will now first give an introduction to the RDS architecture, and next describe the scope of the models (i.e., what system parts/aspects are ignored), before we discuss the studied alternatives in four Sections: one Section on the baseline model and then one for each modeling iteration.

6.1 RDS Architecture Overview

The Remote Diagnostics System (RDS) is used by ABB to offer pro-active service to customers of certain devices. The RDS records device status information, failures and other data. Normally the system follows a push model: the device takes the initiative to connect and upload new data, but a pull model is also supported. In the following discussion, parts of the system are intentionally omitted (and not shown in the referenced figures) either to keep our case study understandable or to protect ABB's intellectual property.

The simplified use case diagram in Figure 6.1 shows the most common use cases for each actor in the system. Customers can track the status of their devices through a web interface and can also generate reports, for example showing device failures over the last year. After receiving a failure notification, service engineers troubleshoot problems on the devices either on-site or remotely. Service engineers can send commands to the device and request it to upload status information (i.e., the pull model). Service engineers also inherit the shown customer use cases. During normal operation devices periodically contact the RDS to upload diagnostic status information (i.e., the push model). Finally, we have the system developers/administrators who perform maintenance on the system and its database.

The structure of the RDS is illustrated in Figure 6.2. The devices run device

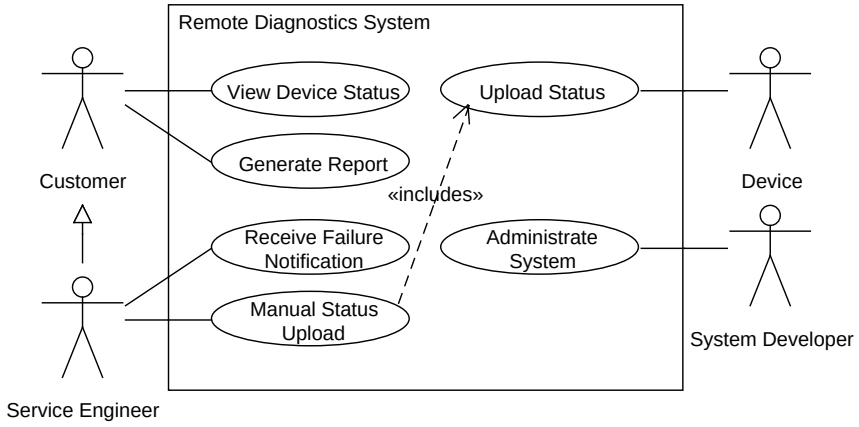

Figure 6.1: UML use case diagram showing the primary actors and use cases for the RDS.

specific software shown as the 'Device Software' component. This software offers an interface to issue commands to the device (e.g., to support the pull model upload requests by the service engineers). The device software connects to the 'RDS Connection Point', a software package running in ABB's DMZ. The connection point is the gate to ABB's internal network. The device data is received by the 'Device Connector', and a check takes place to verify the device is under a service contract. The device data is in a compressed and device specific format, which is unpacked and parsed by the 'Device File Parser'. The parsing component also sends the data onward to the core web service package 'RDS Web Services'.

The system core package 'RDS Web Services' handles both the processing and storing of the uploaded data ('Diagnostic Data Processing + Saving'), as well as the publishing of data ('Website Services' and 'Data Access Service') and interaction with external systems (e.g. the 'device VPN management' interface). The customer website is hosted outside the RDS back-end and gets data from the RDS web services via a proxy, 'Customer Proxy'. The website for service engineers is hosted within the same ASP.Net container as the RDS web services. Both websites offer access to reports that a created by a separate reporting component 'Report Generator'. Received data is processed and then stored in the database (i.e., the 'Database' component). Uploads of diagnostic data are mined, for example to predict the wear of parts, in the 'Data Mining and Prediction Computation' component.

The system is also connected to various other systems. Two examples are shown in the diagram in Figure 6.2: the 'device VPN management' and 'ABB customer and device database' interfaces. The former is used for the management of the VPN connection between the devices and the connection point in ABB's DMZ. The latter represents a Microsoft SQL Server (MSSQL) plug-in that synchronizes the RDS database against a central ABB database recording information on which customers have what service contracts for which devices.

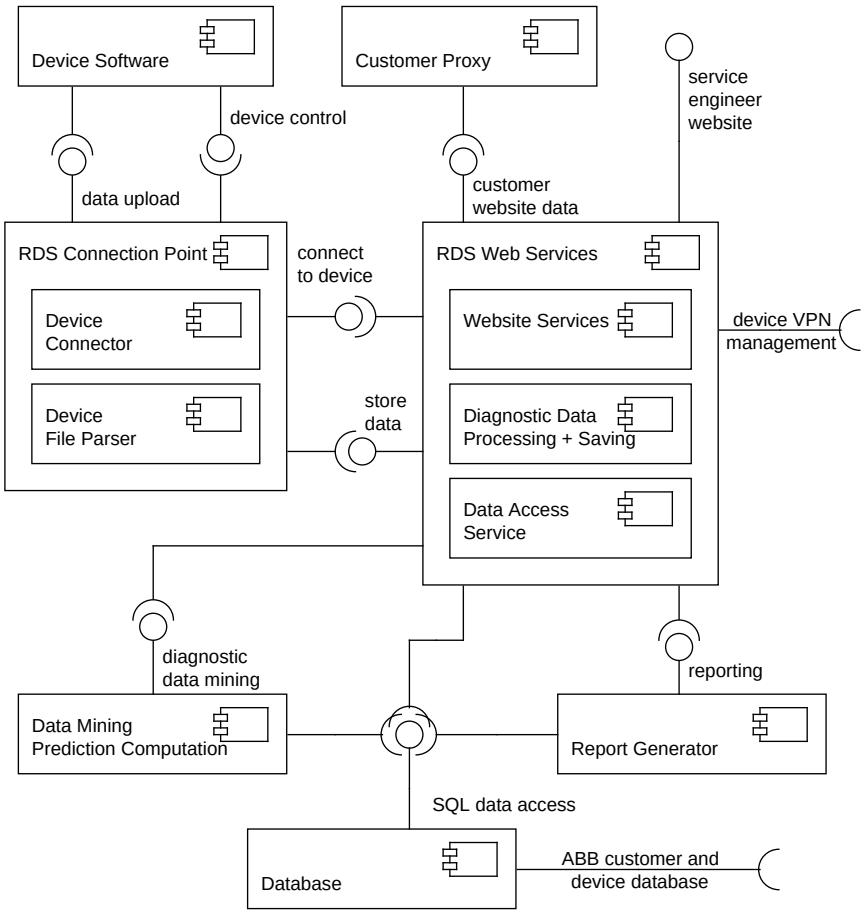

Figure 6.2: UML component diagram showing the most important aspects of the RDS.

This synchronization scheme reduces the latency for look-up of this information when a human user or device connects to the system.

We chose not to discuss the developer and administrator use cases in terms of the logical view in Figure 6.2, for this would increase the complexity of the picture, without offering additional clarity during our discussion of the performance models or alternative architectures considered.

6.2 Modeling Scope

Some parts of the system are considered to have little influence on the performance and are not included in the model to reduce model complexity. Similarly, certain aspects are too laborious to include and therefore are also excluded. In the following, we describe the explicit choices made on the model scope and the rationale for these choices. These choices are summarized in a number of assumptions, which we will later use to justify decisions that we make during modeling or prediction.

6.2.1 Use cases

One way in which we limited the scope was by limiting the considered use cases. For all actors log analysis was carried out to identify the most frequent use cases. It showed that two actors are responsible for the majority of the load: the devices uploading data to the RDS, and the service engineers (SE) visiting the internal website. Of the uploaded data more than ninety percent is either system, diagnostic, or failure information, so these three use cases were modeled to represent the upload behaviour. For the SE, four scenarios were constructed for the most frequently visited pages.

The log analysis also revealed that the uploads are not perfectly spread over time. That is, some peaks exist where the number of uploads per minute goes up by more than an order of magnitude. In our capacity planning, we assume that the upload rate will be flattened, but we reserve capacity for (limited) peaks. Since ABB can control the upload schedule to a great extend this is a reasonable assumption.

The customer actors were not considered because log analysis showed they connect to the system much less frequently than SE. Their performance impact is also lower, because customers only interact with the data of their devices, whereas SE access data on country level. Further, the customer front-end is being moved to another technology, thereby changing the way it interacts with the back-end system.

The reporting functionality was not included despite its significant performance impact, because concrete plans exist to migrate this functionality to a dedicated server.

The developers and system administrators may also interact with RDS via special tools, but the log analysis again showed that these use cases are very infrequent and therefore can be ignored. We summarize our assumption about the importance of the use cases in Assumption 1.

Assumption 1. *The uploading of diagnostic, system, and failure data and the requests SE make to the RDS represent the most important and demanding use cases.*

Assumption 2. *The RDS reporting functionality (i.e., report generation) will be off-loaded to a separate server which also contains a read-only copy of the database.*

6.2.2 Run-time behaviour

The system behaviour for the selected upload and SE use cases was analyzed in load test experiments. Using the Dynatrace monitoring tool we could select a reasonably level of detail for the operations within each use case. In Dynatrace, we can get data on the CPU time and wall clock time spent in each API/component/method. Based on this we included the most important parts of the system.

Assumption 3. *The system behavior that we observed using Dynatrace during our load tests is representative for the average system behavior.*

The RDS struggles with instability in the ASP.Net container. RDS is a highly asynchronous application which opposed to typical web applications has a strong focus on data uploads instead of data downloads. Reduced opportunities for caching and higher processing times result in request termination and other unstable behaviour, when the system is under high load. Our performance model ignores this instability.

Assumption 4. *The stability of the software will be improved by code changes and ASP.Net container tuning.*

6.2.3 Infrastructure

Several aspects of the application infrastructure are not considered in the performance models. The first is 3rd party systems. The RDS interacts with ABB internal and external 3rd party systems. Our model assumes that the response times of the services offered by these systems do not increase as the load within RDS increases. We make this assumption, because we cannot easily obtain information on the scalability of these systems.

Assumption 5. *The performance of 3rd party systems is independent of the load on RDS.*

Second, the network capacity is assumed to always be sufficient and scaled up by the IT provider as required. This assumption is inherited from the CRC project. The first reason for this assumption is that we expect our IT provider to be able to provide the capacity and latency required. The second reason is the limited detail offered by Palladio's network simulator in combination with the extend of experimentation necessary if one wants to specify the network subsystem in detail. For example, it is rather challenging to specify for a complex system running in .Net how much latency network messages experience in each layer.

Assumption 6. *ABB's IT provider will provide network systems that offer the required latency and bandwidth.*

Third, the Microsoft SQL Server (MSSQL) database replication/duplication mechanisms are not modeled or not modeled in detail. Little is known on the

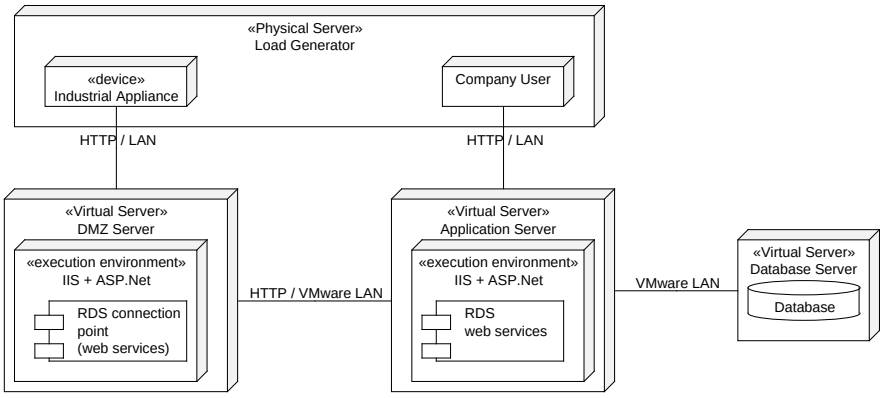

Figure 6.3: Experimental RDS deployment on three virtual machines

exact behaviour of the MSSQL in these cases, and it was not feasible to conduct experiments to prototype the behaviour. As a result the database scalability figures and resource requirements are expected to be slightly optimistic.

Assumption 7. *A simplified model of MSSQL provides us with sufficiently accurate capacity estimates for the system.*

6.3 Baseline Model (Alternative 1)

The December 2010 release of the RDS was taken as the baseline version to which all improvements are measured. This version is deployed in an isolated experimental setup, which is used to do performance measurements. These measurements were used to model the system behaviour, to find the resource demands for all entities in the model, and to calibrate the baseline performance model. In the next Section, we describe the experimental setup and the measurements done, then we explain how we constructed the initial performance model, and finally we discuss the calibration and improvement of this model.

6.3.1 Measurements

All measurements were conducted in an experimental environment at ABB Corporate Research. Three virtual machines were used to run the RDS application as illustrated Figure 6.3. A separate physical machine ran the Neoload load generator tool managing virtual devices and SEs to generate requests to the RDS [neo11]. The experimental setup was instrumented using Dynatrace, which instruments the code on the .Net CLR level and is able to trace a request through all the tiers in our distributed setup [dyn11]. Dynatrace data collection agents were installed on the DMZ and application servers. Performance parameters of windows performance monitor on the database server were also recorded by Dynatrace.

Rolia et al. suggest that measurements should run for at least forty minutes to get reliable average response times for use towards model validation

workload	uploads/min	SE
low	41.0 (34.2)	20.5
medium	78.6 (68.3)	39.3
high	187.9 (162.3)	93.9

Table 6.1: The model calibration workloads used. The number of uploads/min is given as the configured value and between parentheses we give the actual value according to Dynatrace.

[RCK+09]. During the first load tests on our system, we verified the consistency of the performance measurements and we gained sufficient confidence in Dynatrace's instrumentation to run all our measurements for just thirty minutes.

We ran load tests for our calibration on three different intensities: low, medium, and high. The medium workload approximates the current production load on RDS. The workloads are specified in Table[1]6.1 as the number of sustained uploads received from the devices per minute, and the number of concurrent service engineers requesting internal web pages from the system.

6.3.2 Model construction

Only limited architectural documentation was available to base the performance model on. We selected the components to include in the initial model based on a component level logical view and a deployment view, that were developed using information from the developers and source code analysis.

The behaviour was analyzed by creating single stimulus (i.e., one type of upload or one SE visit) measurements with Dynatrace. Dynatrace's ability to create sequence diagrams was very helpful in this endeavour. The Dynatrace instrumentation can be as low as method level, which is far too detailed for the performance model. Therefore we opted to start modeling at the web service level. For example, a web service call to upload a file to the DMZ Server results in several other web service calls to the application server to register and store the uploaded information. These web service calls obviously trigger a number of database calls to persist the information. So our focus was on capturing behaviour on the level of web service calls between the tiers. We added detail when differences in use cases leaded to specific behaviour. To give an impression of the complexity of the model at this time: we had defined seven components, up to 5 web services for each component, and modeled 4 use cases. In the worst case, we would need about $7 \times 5 \times 4 = 140$ measurement values to define the resource demands, which is a substantial amount.

In the baseline model, the deployment mimics the experimental setup because we wanted to be able to calibrate against measurements on this setup. The experimental setup is different from the production setup in that in production the database is co-located on the application server machine. No model was created to mimic the production environment. Alternative 6 ('one' server) described later has a similar deployment scheme, but uses more powerful hardware.

[1]To protect ABB's intellectual property, the data presented has been modified.

6.3.3 Calibration

During calibration we improved the accuracy of the initial model, where accuracy should be read as the difference between the predicted values for performance indicators obtained by model simulation and the values measured during our experiments. Calibration is important, because all performance models for the alternative architectures use the same resource demands as the baseline model, but change the allocation of components and occasionally the order or frequency of operations. One should not forget however that a model error obtained through calibration is of course only valid for that specific model [Kou06a]. After changing the model to capture an architectural alternative, we cannot determine the error, but only assume the error remains constant. Validation of the changed model is only possible by prototypes or full implementation.

After a simulation we would compare a set of metrics (e.g., response times for services of a particular component) and if a prediction was off by more than 20%, we looked for modeling errors or places to add detail to the model. For example, the data mining component needed to be modeled in some detail to get accurate predictions for each type of upload, because the component is quite complex and resource intensive. Occasionally, we also ran additional experiments and performed code reviews to get a better understanding of the system and why the prediction was off. This always led to useful insight, either to improve the model or for the CRC project in general. Every simulation run was recorded in an excel sheet during the calibration to track the model accuracy and the effect of changes made to the model.

After calibration the model gave values up to 30% too low for the DMZ server CPU utilization. The application server utilization figures were off by a maximum of 10% and the database server CPU utilization results were at most 30% too high. The response times for 75% of the validated (internal and external) calls was within 30% of the actual value. These error percentages are for the high load scenario described in Section 6.3.1. The errors for the other workloads are slightly lower, but of the three available workloads the high workload is most representative for the workloads that we used to find the capacity limits of the alternatives. The error percentages are reasonable, but not very low. However, both our measurements in the experimental setup and the model predictions showed that the application server, for which our model most accurately predicts utilization, would be the most likely bottleneck. The alternatives that we select to use in our architectural roadmap in Section 7.3, all have the same components deployed to the application server as in the baseline model, thus we feel sufficiently confident in the use of the model predictions towards the construction of a roadmap.

6.4 Initial Alternatives

The initial architectural alternatives were developed during a workshop with the software development team responsible for RDS and explore one axis of scale each. Both the initial and the second iteration alternatives, try to achieve a capacity increase by improving scalability and not by refactoring of code or other means. The reason for this is the following assumption:

Assumption 8. *The speed-up achieved by code tuning will become less signifi-*

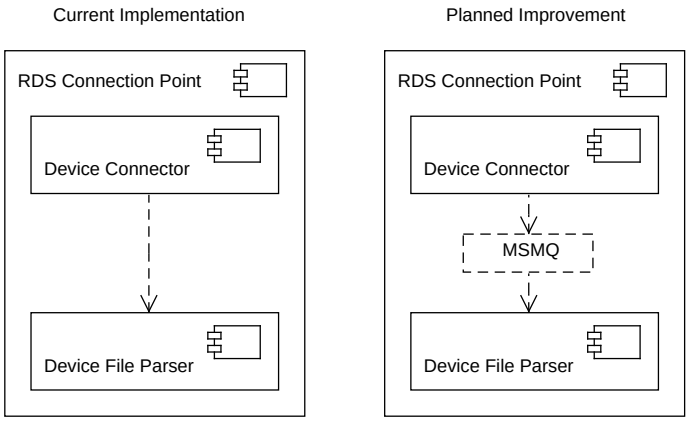

Figure 6.4: Example of the introduction of an explicit queueing mechanic using Microsoft Message Queue (MSMQ) in the RDS, which is currently in progress.

cant with each iteration of code refactoring and cannot be extended to a speed-up of one order of magnitude.

To improve stability and enable scalability some software changes are clearly required, and since the available capacity is already limited in production, work has already started on this. Some changes were even already put into production while this thesis was written. One of the key improvements is the introduction of an explicit queueing mechanism using Microsoft Message Queue (MSMQ) as illustrated in Figure 6.4. A queue will be introduced on the DMZ server to limit the amount of uploads concurrently processed by the system. Thereby this queue will make sure that the system remains responsive under peak loads. Another queue has been implemented to limit the amount of concurrently running data mining processes. The data mining is very computationally expensive and requires a lot of database access. Therefore it can easily take a number of seconds to process one computation. With a queueing mechanic in place this computation can be deferred to a quiet moment. Note that 'quiet' and 'busy' times in the application are defined by two workloads with very different schedules: devices uploading data and users requesting web pages. The effect of these queues was studied using the constructed performance model and found to be beneficial. This is just one example of the side-studies the performance modeling enabled.

The performance models for all alternatives (except for the baseline model) include the same software enhancements. This is relatively easy to do in PCM since these enhancements are represented in the component repository (see Section 2.4.4). Whereas the scale-outs scenarios that distinguish the alternatives are encoded in the system, and allocation models.

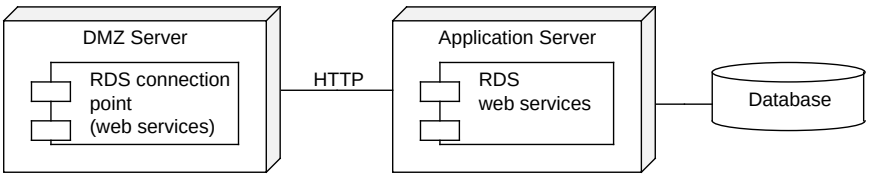

Figure 6.5: Architectural alternative 2: software queues

6.4.1 Alternative Architecture 2

Alternative 2 (software queues)[2] implements the discussed software changes, but does not move along any scale axis yet. It has a deployment similar to the of the experimental RDS deployment shown in Figure 6.3. The underlying hardware has changed though, in the experimental RDS deployment all virtual machines have the same hardware configuration. In alternative 2 (software queues) the resources are balanced towards the relative workload for each machine. Alternative 2 (software queues) is presented in Figure 6.5. Note, that we simplified the diagram. We use this simplified view to illustrate all alternatives, but alternative 3 (extreme Y-split).

The aforementioned queues were modeled in two different ways in Palladio. The queue on the DMZ server uses the event formalism. An enqueue action is modeled as the sending of an event. A dequeue action is done if an event handler is available. The maximum number of concurrent event handlers (i.e., queue consumers) is limited, because the handlers require a limited passive resource for execution. The model for the other queue is similar. However, instead of sending an event, an asynchronous call is made to the queue consumer, which again requires a limited passive resource for execution.

6.4.2 Alternative Architecture 3

Alternative 3 (extreme Y-split) explores the effect of an extreme Y-split. As shown in Figure 6.6, each component has been assigned its own server. To simplify the diagram the component boxes are not drawn. The main goal of alternative 3 (extreme Y-split) is to show that to achieve optimal scalability multiple axes of scale have to be traversed. This was needed because the current strategy for the system was to move 'heavy' components to other machines, see Assumption 2 (page 52) as an example.

To model alternative 3 (extreme Y-split) we have re-used the system model from alternative 2 (software queues), but obviously the additional hardware resources had to be specified and the deployment had to be altered.

6.4.3 Alternative Architecture 4

With alternative 4 (replicate) the X-axis of scale is explored: an identical copy of the all servers is set-up, and the data tier is partly shared by the introduction of

[2]All alternatives have been assigned short names, which will be included between parentheses.

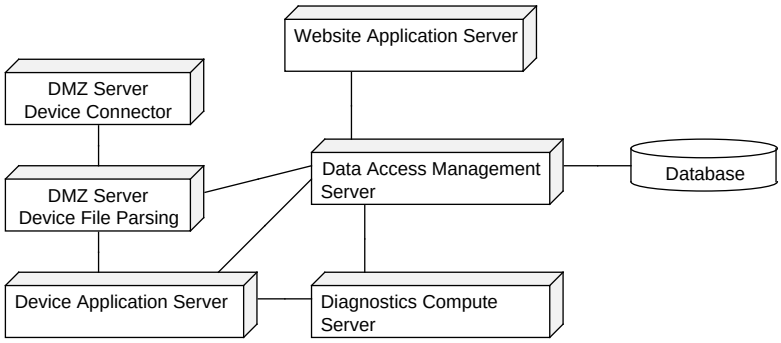

Figure 6.6: Architectural alternative 3: extreme y-split

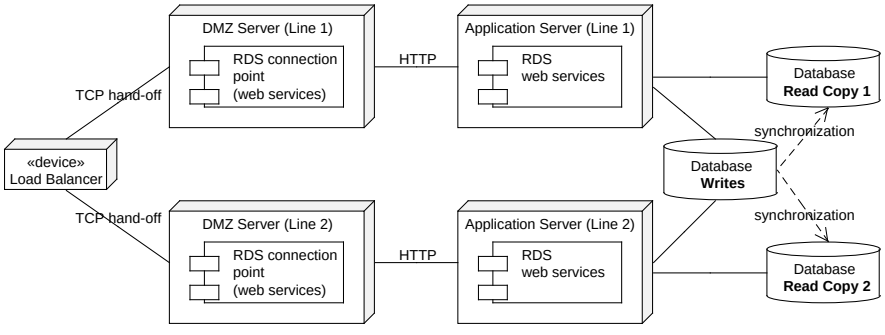

Figure 6.7: Architectural alternative 4: replicate

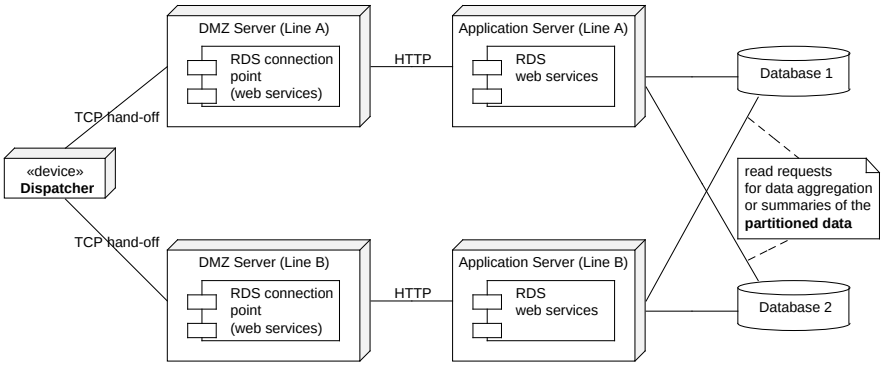

Figure 6.8: Architectural alternative 5: partition

a shared write-only database server as shown in Figure 6.7. Several alternatives were considered for the data tier: two mirrored database servers, one shared server, and the chosen option with three servers. The shared server option was ruled out, because it does not agree with the strategy of replication. The mirrored synchronized databases option does align, but it was feared that the reads would be disturbed by the frequent writes in the database. The main use case of the RDS is to record data, thus compared to traditional web applications the read:write ratio is lower (i.e., more writes for every read). In the selected design the new data can periodically be written to the read copies of the database in transactions optimized by MSSQL. The workload is spread round-robin over the two available pipelines by a software load balancer using a TCP hand-off.

Initially, only one 'pipeline' of the alternative was modeled. Later the other was added to study the contention for the write database. However, the synchronization between the write database and the read database copies was never modeled for two reasons. First, little information is available about MSSQL's behaviour in this situation or what optimizations it makes. Second, the simulations showed that the workload on the databases was limited and the application servers and communication links would form the bottleneck. The software load balancer is assumed to have minimal computational demands and is therefore not modeled.

Assumption 9. *The software load balancers used in the architectural designs have limited performance impact and their resource demand therefore does not have to be included in the performance model.*

6.4.4 Alternative Architecture 5

The final axis of scale, the Z-axis, is embodied in alternative 5 (partition) and pictured in Figure 6.8. The data is partitioned on the device identifier, which uniquely identifies a device and is submitted by the devices in the header to each upload. The load balancer from alternative 4 (replicate) now needs to be more intelligent and examine the device identifier before forwarding a request, hence it got the more intelligent name 'dispatcher'. As the split on device id

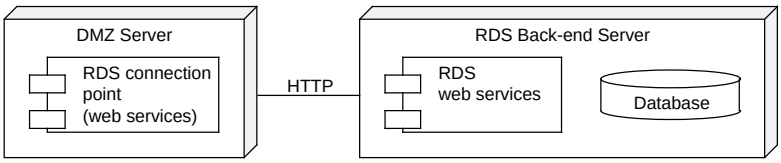

Figure 6.9: Architectural alternative 6: 'one' server

does not respect many logical splits (e.g., based on customer), read requests for aggregated data will cause requests from one pipeline's application server to both databases.

Alternative 5 (partition) has not been modeled explicitly. If the split is perfectly balanced, it should have twice the capacity of alternative 2 (software queues) for all machines have the same specifications and are duplicated. In practise, the workload might be uneven, because each device is tied to one of the pipelines and the requests for a certain pipeline might be clustered in time. The latter should be avoided however by choosing the right discrimination criterion and analyzing historical data. We also chose not to model and study the dispatcher, because the implementation was not decided upon, and because it is likely that a separate device or machine will be used if the task is computationally intensive.

6.5 Second Iteration Alternatives

The initial alternatives were modeled and evaluated, but all failed to provide the needed extra capacity, so more alternatives had to be developed. During the second architecture iteration we got more information on the hardware and software cost model, which was found to favour a low number of servers and a low number of databases in particular. Based on this new knowledge we developed the alternatives presented below.

6.5.1 Alternative Architecture 6

With alternative 6 ('one' server) we reversed our strategy and reduced the number of machines to the absolute minimum, because this seemed to be most economical in the cost model. For security purposes a separate DMZ Server still has to exist, but otherwise all components are deployed on one machine, the 'RDS Back-end Server'. The performance model for this alternative is a straightforward deployment according to the diagram in Figure 6.9.

6.5.2 Alternative Architecture 7

Despite the heavy configuration of the back-end server in alternative 6 ('one' server), the latter fails to provide sufficient capacity. This led to the development of alternative 7 (synchronized databases) (Figure 6.10), which consists of two back-end servers in an X-split. A database synchronization has to be set-up because

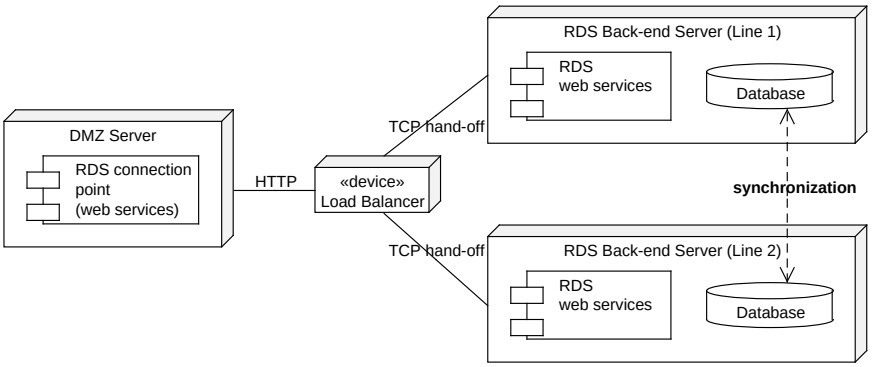

Figure 6.10: Architectural alternative 7: synchronized databases

of the X-split, which uses replication (remember, partitioning happens in a Z-split). The load balancing is handled by a software load balancer on the DMZ server directing the requests to either of the back-end servers.

The performance model for alternative 7 (synchronized databases) is similar to those of alternatives 4 and 5. However, for alternative 7 (synchronized databases) we also modeled a simple database synchronization mechanism. We modeled the two-phase commit protocol for two nodes [OV11]: upon receiving a write transaction the master first sends a prepare command to the replica; after receiving an acknowledgement from the replica; it sends a commit command; and commits the transaction locally. Both databases have to process all the writes in real-time (i.e., no lazy synchronization), this makes the performance of one line dependent on the database performance of the other line. The underlying assumption is that this simple model matches MSSQL's behaviour well enough to maintain model accuracy.

Assumption 10. *Our implementation of two-phase commit in the performance model matches the behaviour of MSSQL well enough to make sensible predictions on the performance of two synchronizing databases.*

6.5.3 Alternative Architecture 8

The database synchronization used in alternative 7 (synchronized databases) creates a heavy load on the network between the two databases and requires hardware capacity for the database on both back-end servers. A cheaper and simpler alternative is to have a shared database server between replicated application servers as shown in Figure 6.11. No new solutions or additional components were needed to construct the performance model of alternative 8 (shared database). We could re-use the model of alternative 7 (synchronized databases), and consolidate the databases on a new database machine.

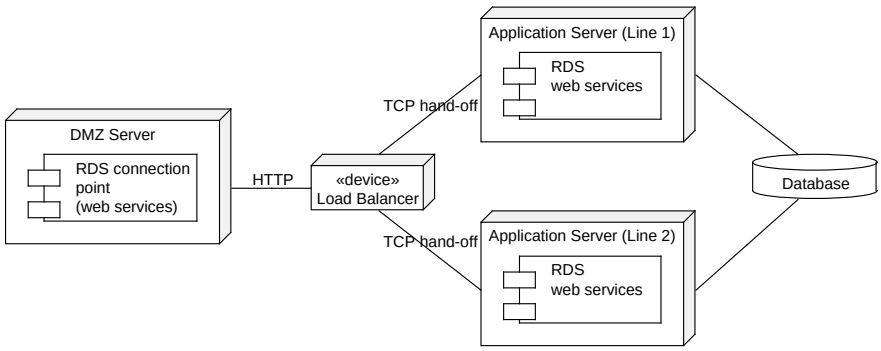

Figure 6.11: Architectural alternative 8: shared database

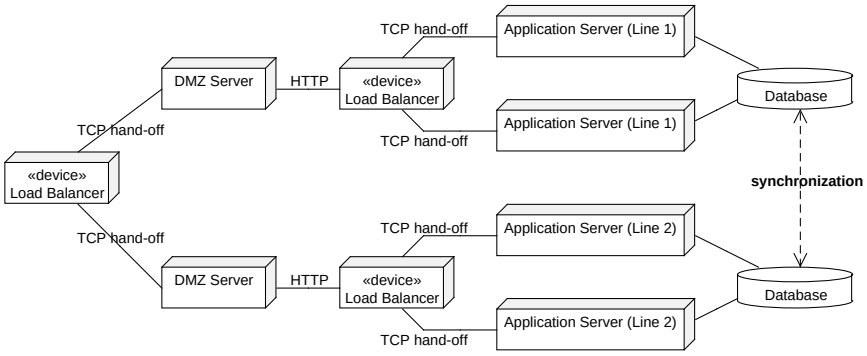

Figure 6.12: Architectural alternative 9: x-split shared database

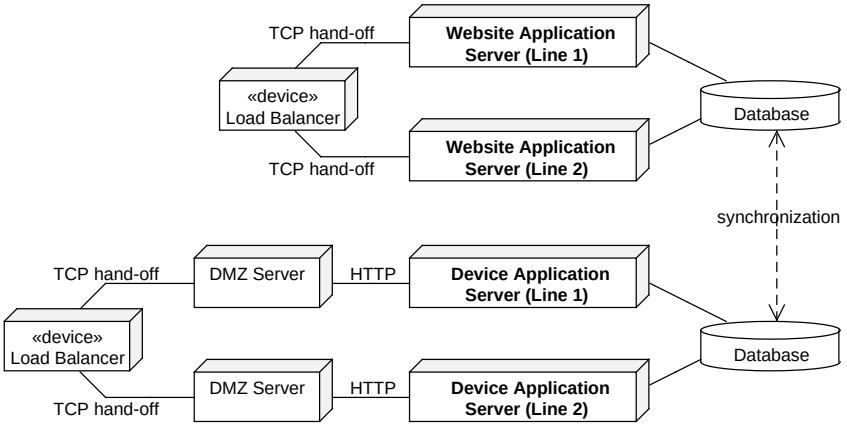

Figure 6.13: Architectural alternative 9b: z-split on actor

6.5.4 Alternative Architecture 9

Looking to increase capacity even further than in previous alternatives we introduced alternative 9 (X-split shared database) with four application servers, two connector servers and two levels of load balancing as shown in Figure 6.12. Essentially it is an X-split of alternative 8 (shared database) and is also modeled as such, but we re-use the database synchronization feature that we introduced in our performance model for alternative 7 (synchronized databases).

So far, all alternatives assume that the functionality offered by the components within the RDS will not change and the structure of the system will remain the same. However, there is one clear use case split that can be made: human users visiting the web pages versus devices uploading data to the RDS. Both these use cases are handled in the same application container in all alternatives, but in alternative 9b (Z-split on actor) we explore splitting these use cases. In this alternative, there is one line (with two replicas) to handle the website requests and one line (with again two replicas) to process the uploads. The architecture is illustrated in Figure 6.13. Again DB synchronization is needed, but now data mostly flows from the database connected to the 'upload' line to the 'website' line. The performance model for alternative 9b (Z-split on actor) is a copy of that of alternative 9 (X-split shared database) minus the DMZ server for the 'website' line.

6.6 Third Iteration Alternatives

The previous two iterations focussed on increasing scalability within the RDS and all indeed increased capacity compared to the baseline. However, we also used the performance models to study other software changes. In this Section, we present the two most prominent ones. Most of these changes were studied

after most of the alternatives discussed before had been modeled, hence we discuss them as the third iteration.

During the calibration of the initial model, a discrepancy was noted in the metrics of the data mining component. The component had already been modeled in quite some detail and its internal behaviour seemed correct. A code review then revealed the web service call that causes the data mining routines to run used a C# lock-statement to become mutual exclusive. In other words, only one client at a time could execute this web service, thereby reducing the capacity of the system. We modeled this lock in the performance model as a passive resource with only one available instance. With the lock included in the model, we could study the capacity difference between the model with and without the lock, and do so for each alternative. We later learned that the lock had been introduced to throttle incoming requests during peak loads as a temporary replacement for the software queues.

In the thesis introduction, two directions of growth are identified: an increase in the number of connected devices and an increase in the amount of data collected from each device. We measure the quality of an architecture for the first direction of growth by searching the maximum capacity it offers. To study the capacity reduction of an alternative under an increase in the collected information, we created a copy of the models and modeled the effect of the increase in collected information. To model the effect we changed the behavioural specifications and resource demands (i.e., changing the Palladio SEFF diagrams) by asking the below questions and reflecting the answers in the models.

- Is the behavioural specification or resource demand affected by an increase of information?
- If so, what kind of change is likely to occur? For example, a linear/exponential increase in computation time.

Chapter 7

Predictions for an Architectural Roadmap

In this Chapter, we will discuss how we configured Palladio to evaluate the performance model, what the results of the model evaluations are, and how ABB constructed an architectural roadmap based on these results.

7.1 Simulator Configuration

The Palladio-Bench includes several model evaluation methods: an analytical solver (PCMSolver), a model simulator (SimuCom), and a plug-in that translates Palladio models to a LQN and then starts LQNS (see Section 4.2). The LQN option was not available to us (recall the discussion in Section 4.7) and PCMSolver does not offer all required functionality, so the SimuCom PCM simulator was used for model evaluation.

During calibration the simulation runs were configured to simulated 30 minutes of wall clock time, and thereby match the duration of our measurements. Simulation runs to find the capacity limits were capped at 10 or 15 minutes simulated wall clock time. The actual runtime for simulations varies from under a minute to near real-time and depends on model complexity, workload intensity and resource saturation. We found the runtime to be quite sensitive to high workloads (regardless of resource utilization), hence we limited the simulated time when looking for the capacity limits.

SimuCom offers a basic network and middleware simulation, which we enabled during calibration and thus also during our capacity study. It was only during the study, however, that we learned that the simulation runtime is increased considerably when this feature is enabled for high workload simulations. Another disadvantage we discovered was the limited accuracy of the network simulation. There is no documentation describing how the network latency specified in the model is used by SimuCom, but after contact with some of its developers, it became apparent that the entire duration of the latency is spent in the model's network resource. Because our model assumed the latency to be measured from the entry of the network stack on one node, till the exit from the network stack on another note, the network resource utilization figures obtained were inaccurate. The utilization of some network resources also be-

Alternative	Capacity (NCD)	Capacity [info] (NCD)	Cost (US$)
1 (baseline)	900	–	8.55
2 (software queues)	1600	500	15.34
3 (extreme Y-split)	3500	900	38.17
4 (replicate)	3500	1100	62.88
5 (partition)	3200	1000	41.69
6 ('one' server)	1800	5000	17.92
7 (synchronized databases)	3500	1100	40.63
8 (shared database)	3500	1100	32.67
9 (X-split shared database)	6700	2200	67.92

Table 7.1: Capacity predictions and cost of the modeled architectures.

came so high that they formed a bottleneck. Forcing us to introduce additional network resources in the model acting according to Assumption 6 (page 52).

Further, all simulation results were stored to files for later reference. Other settings were kept to default and the option to simulate failures to get reliability estimates was not enabled.

7.2 Model Simulation Results

The goal of the evaluation of the performance model is to find the maximum capacity measured in the number of connected devices (NCD) for each alternative. To establish the maximum number of connected devices (NCD) each alternative supports, we used the following criteria:

CPU max 50% (utilization $\leq 50\%$) – There should be spare capacity available on the CPU of each server to deal with peak loads (e.g., because of a queue of uploads caused by maintenance downtime). 50% is an accepted goal utilization in industry.

LAN not saturated (utilization $\leq 80\%$) – We allowed for higher utilizations on the LAN for the reasons mentioned in the previous Section and because of Assumption 6.

The number of concurrent service engineer users (CSE) is given by the following assumption:

Assumption 11. *The ratio between the number of connected devices (NCD) and concurrent service engineers (CSE) visiting the RDS website is constant.*

In the RDS version that we modeled, there is a very high number of database calls for each request, leading to severely skewed response time figures for the web services under high load. The number of database calls per request was reduced significantly in a later release and a migration to another database abstraction layer is in progress. Therefore we ignored response times in our capacity study.

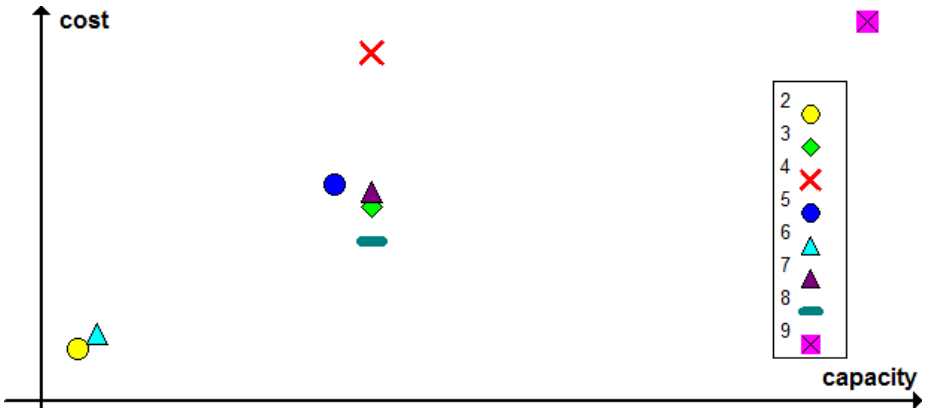

Figure 7.1: Cost for each of the modeled architectures versus their capacity

The capacity in NCD for each alternative is shown in Table[1]7.1. We list the capacity for each alternative as obtained after modeling iteration 2, and after modeling iteration 3 for the case where the amount of collected information is increased (i.e., only of of the cases discussed in Section 6.6). The operational cost for each alternative are also included in the table. These were calculated by the CRC project team using a cost model to enable a trade-off between cost and performance.

7.3 Architectural Roadmap

To move the RDS from its current state to a scaled-out version, an architectural roadmap was established. The capacity predictions and cost for each of the architectures as introduced in the previous Section, form the basis for this architectural roadmap. Figure[2]7.1 shows the relation between cost and capacity. A roadmap is necessary for two reasons. First, functional development and maintenance of RDS has to continue, so not all effort can be put into migrating the system to its final goal in one step. Multiple increments, each providing a capacity gain, are required to spread the work. Second, immediately deploying the alternative with the targeted capacity would initially provide an order of magnitude too much capacity, and thereby increase cost per connected device to an unacceptable level.

The performance models enabled ABB to find alternatives providing the required capacity and identify what architectures can form intermediate steps. The roadmap is as follows:

- The predictions made very clear that the first step should be to parallelize the data mining process, which was capped by a lock-statement in the code (recall Section 6.6), and later by having only one consumer on the queue for this process (queues discussed in Section 6.4).
- Second, the already planned improvements should take place, such as introducing a dedicated database server and a dedicated server for reporting

[1]To protect ABB's intellectual property, the data presented has been modified.
[2]To protect ABB's intellectual property, the data presented has been modified.

(Assumption 2).

- Now before the new architectures can be implemented, some changes that enable this implementation must be made. For example, a NAS device for shared file storage should be put in place.
- After this, architecture 8 should be implemented, because it offers an medium capacity at low cost.
- Finally, architecture 9 may be put into place to further increase the capacity.

For ABB one of the major benefits of the performance modeling done, is the fact that the capacity figures can be combined with the business growth scenario to get a time-line on the architectural roadmap. Without the capacity predictions by the performance model, only guesswork could have helped ABB to plan its migration. This guesswork would have increased the risk of being too late and experiencing capacity problems, or being too early and making unnecessary investments.

Chapter 8

Experiences with Palladio

The experiences we gained during the case study are summarized in this Chapter. First, we contemplate on the usefulness of performance modeling and the PCM in the RDS architectural redesign in Section 8.1. Second, we share our experiences with Palladio-Bench in Section 8.2.

8.1 Performance Modeling Using the PCM

Irrespective of Palladio-Bench or the obtained capacity information, the construction of a performance model in itself was already useful. It forced us to understand the system's (performance) behaviour and identify the critical parts: it helped us to ask the right questions about the system.

Similarly, model calibration helped us find oddities in the system behaviour. The model represented a polished version of the system that should match its average behaviour, but under higher load this might not be the case. For example, it was only when calibrating the system against a higher load than the load used to instantiated the model, that we learned about the imposed limitations (i.e., lock) around the data mining routines.

The mapping of architecture concepts onto the PCM concepts was easy and straightforward as expected. This greatly sped-up model construction, and enabled us to explore more alternatives and to quickly evaluate minor variations while tuning the alternatives.

One of the oddities we ran into during the very first modeling efforts, was that the user was required to specify linking resources (i.e., networks), even though the simulation of these was disabled. Further, it is unclear how the network parameters should be specified (recall our discussion on network latency in Section 7.1). Later in a conversation with the authors of Palladio, we learned that the network model lacks detail and is still under construction. For detailed network performance analysis they recommended to use a network performance tool. This limitation is acceptable, but not documented or apparent to the user from the GUI.

The separation of concerns into different models (e.g., component repository and system architecture model) is a strong feature. Unfortunately, the combination of different models was occasionally difficult in practise. For example, it is tricky to combine a usage model (i.e., workload) with several different sys-

```
java.lang.NullPointerException
  at java.util.Hashtable.get(Unknown Source)
  at de.uka.ipd.sdq.scheduler.processes.impl.ProcessRegistry...
  at de.uka.ipd.sdq.scheduler.resources.active.SimActiveRes...
  at de.uka.ipd.sdq.scheduler.resources.active.SimActiveRes....
  at de.uka.ipd.sdq.scheduler.resources.active.AbstractAct...
  at de.uka.ipd.sdq.simucomframework.resources.ScheduledRes...
  at de.uka.ipd.sdq.simucomframework.resources.Abstract...
  at defaultrepository.impl.Upload.iUpload_postFile...
  at defaultrepository.impl.ports.IUpload.postFile...
  at rsc.impl.ports.IUpload_RDS.postFile...
  at uploads.impl.Uploads.scenarioRunner...
  at de.uka.ipd.sdq.simucomframework.usage.OpenWorkloadUser...
  at de.uka.ipd.sdq.simucomframework.usage.OpenWorkloadUser...
  at de.uka.ipd.sdq.simucomframework.abstractSimEngine.SimProc...
  at de.uka.ipd.sdq.simucomframework.ssj.SSJSimProcess...
  at de.uka.ipd.sdq.simucomframework.ssj.SSJSimProcess$1...
  at java.lang.Thread.run(Unknown Source)
```

Figure 8.1: NullPointerException thrown by a model transformation in Palladio-Bench

tem architectures, because the usage model depends on a particular, version dependent, identifier of an interface.

8.2 Using the Palladio-Bench Tool

While we successfully used the Palladio-Bench and it generally worked well, the tool is the major weakness of the Palladio 'approach'. One of the problems is the lack of adequate documentation. The existing documentation is either very technical and detailed, or too high-level to help in constructing a complex model. Documentation that would be useful is provided in the form of technical reports and theses, but most are outdated by at least two years.

Besides the lack of up-to-date documentation, more usability problems exist. For example, there are vague constraints on entity names within models. We could not use "(Linq/ADO.Net)" in a component name or interface operations with a dot such as "Login.aspx". Unfortunately, both were accepted by the editor and only caused problems during model transformation. Similarly, the middleware simulation requires a CPU to exist on all resource containers, but no message pops-up telling this. Instead, the user has to analyze a Java exception thrown during the model transformation. We have to note that occasionally sensible and understandable exceptions were thrown, but we also faced NullPointerExceptions such as the one in Figure 8.1, which later appeared to tell us that we cannot use the operating system scheduling algorithms in our model (e.g., imitating the scheduler of Windows 2003).

Palladio-Bench also suffers from some stability problems. For example, it often happens that when a simulation is aborted the Eclipse framework freezes or crashes. Further, the simulation time can get near real-time for high workloads.

Long run-times are a known problems of simulations, but SimuCom additionally experiences memory problems. Palladio-Bench runs in Eclipse Galileo, which no longer comes in a 64-bit version for windows thereby limiting the allocatable memory to one gigabyte. Using an older Java version and an archived Eclipse build, we got a 64-bit environment up and running to work around this problem, but this shouldn't be necessary.

We already mentioned the separation of concerns as a strong point, and in the same category the ability to re-use entire models is convenient. But again, the Eclipse modeling framework makes this more tricky than it should be at times. For example, model files occasionally contain the full path when referencing other files instead of a model id or a relative path within an Eclipse project. Updating these references is cumbersome and cannot be done from within the Eclipse user interface.

Finally, Palladio-Bench could benefit from a more efficient way to store its results to handle large simulations well (i.e., simulations of a lot of uploads per second and CSE for more than 10-15 minutes). For example, the visualization mechanisms provided may take a long time to generate a diagram or fail to do so at all. It would be convenient to have a well functioning result visualization for quick use. While it is also possible to export the results and use an external program to create diagrams, the processing of the results before they can be exported also takes a while.

In the end, the PCM was very suited to our problem and easy to use. While some problems exist in Palladio-Bench, we could solve or work-around all of them. At the moment Palladio-Bench is one of the very few performance modeling tools that offers intuitive modeling and is actively maintained. For industrial applications the biggest question that remains is how well Palladio-Bench scales to even bigger models with more intense workloads.

Chapter 9

Future Work

The work presented in this thesis can be extended and several ways to do so are already planned to take place. We will discuss these plans first and then outline other possible directions in which the work can be continued.

The first task that has been planned to be carried out directly after the completion of the thesis, is a case study of the Peropteryx design space exploration tool. Peropteryx is an extension to Palladio-Bench under development by the same research team [KKR11]. After specifying a cost model and the degrees of freedom for the existing PCM, Peropteryx generates a set of initial architecture candidates. It then employs the Palladio-Bench model evaluation tools (e.g., SimuCom or LQNS) on each candidate and continues with a genetic search based on the most promising candidates. In the end, Peropteryx gives a Pareto curve of architectural candidates showing the trade-off between cost and performance. The results of prior case studies using Peropteryx are discussed in [MKBR10, KSB+11]. We look to apply Peropteryx for the obvious benefits of automatic design space exploration, but also to study the potential of the included cost model for making architectural decisions in an industrial setting.

Second, we plan to submit the results and experiences of both case studies for publication at ICPE2012. We will also share our knowledge and experience across the corporate research centers within ABB. In the internal communication we plan to show how we used the Dynatrace performance monitoring software and Neoload load generator to study and mitigate performance problems in production systems.

Potential future work is a verification of our predictions and validation of the suggested architecture roadmap after the alternatives have been implemented and put into production. Similarly, validation of our experiences and results could be obtained by applying another modeling language to the same case. This would also enable us to compare merits like is done in [RCK+09]. However, considering the time and effort required for this, we do not expect the case study to be repeated using another tool.

Chapter 10

Conclusions

Based on our literature review analyzing industrial applicability of more than ten performance modeling tools and the presented case study, we conclude that Palladio-Bench is one of the very few performance modeling tools that offers intuitive modeling and is actively maintained. We found that most of the performance modeling tools that are available either are no longer maintained, or use modeling concepts that are not intuitive to the software architect and require more performance modeling knowledge to obtain the right abstraction. Unfortunately, we forsee that Palladio-Bench's academic nature might limit industrial adoption. For industrial applications, the question remains how well Palladio-Bench scales to even bigger models with more intense workloads, and when the remaining usability and stability problems will be dealt with.

However, if we generalize our experience, performance modeling tools seem to begin to meet industry's needs. Without prior experience, we were able to build a performance model of a 300 KLOC system with a 10–20% prediction accuracy. The final model consisted of 13 components, 41 behaviour specifications, $41 \times N$ resource demands, and up to 9 servers/model. This model could be re-used relatively easily, enabling us to analyze more than ten alternative architectures that each explored various scalability options.

The work described in this thesis enabled ABB to take an informed decision on an architectural roadmap to increase the capacity of their RDS back-end. We showed how an iterative performance modeling approach was instrumental in creating this roadmap of architectural redesign. One of the main benefits of our performance modeling efforts is that the created capacity estimates could be used to put a sensible timeline on the roadmap. Finally, the insight in the RDS gained by constructing and calibrating the performance model, and the side studies the model enabled, also proved very valuable.

Bibliography

[AF09] Martin L. Abbott and Michael T. Fisher. *The art of scalability.* Addison–Wesley, 2009.

[BC09] Marco Bertoli and G Casale. JMT: performance engineering tools for system modeling. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):10–15, 2009.

[BDIS04] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and M Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

[Bec] Steffen Becker. Palladio-Bench Screencasts. `http://www.palladio-simulator.com/tools/screencasts/` – last accessed: 8th June 2011

[BGMO06] Steffen Becker, Lars Grunske, Raffaela Mirandola, and S. Overhage. Performance Prediction of Component-Based Systems - A Survey from an Engineering Perspective. *Architecting Systems with Trustworthy Components*, pages 169–192, 2006.

[BKK09] Fabian Brosig, Samuel Kounev, and Klaus Krogmann. Automated extraction of palladio component models from running enterprise Java applications. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 1–10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[BKR07] Steffen Becker, Heiko Koziolek, and Ralf Reussner. Model-Based performance prediction with the palladio component model. *Proceedings of the 6th international workshop on Software and performance - WOSP '07*, page 54, 2007.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, January 2009.

[BM05] Simonetta Balsamo and Moreno Marzolla. Performance evaluation of UML software architectures with multiclass Queueing Network models. In *Proceedings of the 5th international workshop on Software and performance - WOSP '05*, pages 37–42. ACM, 2005.

[BMDI04] Simonetta Balsamo, Moreno Marzolla, Antinisca Di Marco, and Paola Inverardi. Experimenting different software architectures performance techniques. *Proceedings of the fourth international workshop on Software and performance - WOSP '04*, pages 115–119, 2004.

[BML11] Ágnes Bogárdi-Mészöly and Tihamér Levendovszky. A novel algorithm for performance prediction of web-based software systems. *Performance Evaluation*, 68(1):45–57, January 2011.

[BMLC07] Ágnes Bogárdi-Mészöly, Tihamér Levendovszky, and Hassan Charaf. Models for predicting the performance of ASP.NET Web applications. *Periodica Polytechnica Electrical Engineering*, 51(3-4):111, 2007.

[BMLC09] Ágnes Bogárdi-Mészöly, Tihamér Levendovszky, and Hassan Charaf. Improved Performance Model for Web-Based Software Systems. *WSEAS TRANSACTIONS on COMPUTERS*, 8(10):1711–1720, 2009.

[BMLCH07] Ágnes Bogárdi-Mészöly, Tihamér Levendovszky, Hassan Charaf, and Takeshi Hashimoto. Improved Evaluation Algorithm for Performance Prediction with Error Analysis. *Intelligent Engineering Systems, 2007 International Conference on*, pages 301–306, June 2007.

[BMLS09] Ágnes Bogárdi-Mészöly, Tihamér Levendovszky, and A. Szeghegyi. Improved performance models of web-based software systems. In *Intelligent Engineering Systems, 2009. INES 2009. International Conference on*, pages 33–38. IEEE, 2009.

[BOG08] Paul Brebner, Liam O'Brien, and Jon Gray. Performance Modeling for Service Oriented Architectures. In *ICSE Companion '08 Companion of the 30th international conference on Software engineering*, pages 953–954, New York, NY, USA, 2008. ACM.

[BOG09] Paul Brebner, Liam O'Brien, and Jon Gray. Performance modeling evolving Enterprise Service Oriented Architectures. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, number 2, pages 71–80. IEEE, September 2009.

[Bre11] Paul Brebner. Real-world Performance Modelling of Enterprise Service Oriented Architectures : Delivering Business Value with Complexity and Constraints. In *International Conference on Performance Engineering 2011*. ACM, 2011.

[CF07] V Cortellessa and L Frittella. *A framework for automated generation of architectural feedback from software performance analysis*, volume 4748, pages 171–185. Springer Berlin / Heidelberg, 2007.

[CGK+09] Tod Courtney, Shravan Gaonkar, Ken Keefe, Eric W. D. Rozier, and William H. Sanders. Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 353–358, June 2009.

[CGM+07] Tod Courtney, Shravan Gaonkar, Michael G. McQuinn, Eric Rozier, William H. Sanders, and Patrick Webster. Design of Experiments within the Mobius Modeling Environment. In *Fourth International Conference on the Quantitative Evaluation of Systems (QEST 2007)*, pages 161–162. IEEE, September 2007.

[CSM11] Giuliano Casale, Giuseppe Serazzi, and Politecnico Milano. Quantitative System Evaluation with Java Modeling Tools (Tutorial Paper). In *International Conference on Performance Engineering 2011*, pages 3–8. ACM, 2011.

[CZMC09] Marco Crasso, Alejandro Zunino, Leonardo Moreno, and Marcelo Campo. JEETuningExpert: A software assistant for improving Java Enterprise Edition application performance. *Expert Systems with Applications*, 36(9):11718–11729, November 2009.

[dyn11] Dynatrace - Application Performance Management and Monitoring, 2011. `http://www.dynatrace.com` – last accessed: June 14th, 2011

[FAOW+09] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced Modeling and Solution of Layered Queueing Networks. *IEEE Transactions on Software Engineering*, 35(2):148–161, March 2009.

[FMN+96] Roy Gregory Franks, S. Majumdar, J.E. Neilson, Dorina C. Petriu, Jerry Rolia, and Murray Woodside. Performance analysis of distributed server systems. In *Proceedings of the sixth International Conference on Software Quality, Ottawa, Canada*, pages 15–26. Citeseer, 1996.

[FMW+05] Greg Franks, Peter Maly, Murray Woodside, Dorina C. Petriu, and Alex Hubbard. Layered Queueing Network Solver and Simulator User Manual. Technical report, Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada, 2005.

[Fra99] Roy Gregory Franks. *Performance Analysis of Distributed Server Systems.* PhD thesis, Carleton University, Ottawa, Ontario, Canada, 1999.

[GKL+09] Shravan Gaonkar, Ken Keefe, Ruth Lamprecht, Eric Rozier, Peter Kemper, and William H. Sanders. Performance and dependability modeling with Möbius. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):16, March 2009.

[GT06] Stephen Gilmore and Mirco Tribastone. Evaluating the scalability of a web service-based distributed e-learning and course management system. In *Web Services and Formal Methods Third International Workshop, WS-FM 2006 Vienna, Austria*, volume 4184, pages 214–226. Springer Berlin / Heidelberg, 2006.

[GTC10] Alessio Gambi, Giovanni Toffetti, and Sara Comai. Model-Driven Web Engineering Performance Prediction with Layered Queue Networks. In Florian Daniel and Federico Facca, editors, *Current Trends in Web Engineering*, volume 6385 of *Lecture Notes in Computer Science*, pages 25–36. Springer Berlin / Heidelberg, 2010.

[HBR+10] Nikolaus Huber, Steffen Becker, Christoph Rathfelder, J. Schweflinghaus, and Ralf Reussner. Performance modeling in industry: a case study on storage virtualization. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pages 1–10. ACM, 2010.

[Hub09] Nikolaus Huber. *Performance modeling of storage virtualization.* thesis, Universität Karlsruhe, 2009.

[IWF07] T Israr, Murray Woodside, and Greg Franks. Interaction tree algorithms to extract effective architecture and layered performance models from traces. *Journal of Systems and Software*, 80(4):474–492, April 2007.

[Jai91] Raj Jain. *The art of computer systems performance analysis.* John Wiley & Sons, 1991.

[JTHL07] Yan Jin, Antony Tang, Jun Han, and Yan Liu. Performance Evaluation and Prediction for Legacy Information Systems. *29th International Conference on Software Engineering (ICSE'07)*, pages 540–549, May 2007.

[KB03] Samuel Kounev and Alejandro Buchmann. Performance modelling of distributed e-business applications using Queuing Petri Nets. *IEEE International Symposium on Performance Analysis of Systems and Software ISPASS 2003*, pages 143–155, 2003.

[KB06] Samuel Kounev and Alejandro Buchmann. SimQPN – A tool and methodology for analyzing queueing Petri net models by means of simulation. *Performance Evaluation*, 63(4-5):364–394, May 2006.

[KDB06] Samuel Kounev, C. Dutz, and Alejandro Buchmann. QPME-queueing petri net modeling environment. In *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*, pages 115–116. IEEE Computer Society, 2006.

[KKR11] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. PerOpteryx: Automated Application of Tactics in Multi-Objective Software Architecture Optimization. In *(to be published) Proceedings International Conference on the Quality of Software Architectures (QoSA'11), Boulder, Colorado*, 2011.

[Kou06a] Samuel Kounev. J2EE Performance and Scalability - From Measuring to Predicting. In *Spec Benchmark Workshop*, page 12 pp., 2006.

[Kou06b] Samuel Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, 32(7):486–502, July 2006.

[Koz04] Heiko Koziolek. *Empirische Bewertung von Performance-Analyseverfahren für Software-Architekturen.* Diplomarbeit, Universität Oldenburg, 2004.

[Koz10] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, August 2010.

[KSB+11] Heiko Koziolek, Bastian Schlich, Carlos Bilich, Roland Weiss, Steffen Becker, Klaus Krogmann, Mircea Trifu, Raffaela Mirandola, and Anne Koziolek. An Industrial Case Study on Quality Impact Prediction for Evolving Service-Oriented Software Categories and Subject Descriptors. In *Evolution*, 2011.

[KSM10] Samuel Kounev, Simon Spinner, and Philipp Meier. *QPME 2.0 - A Tool for Stochastic Modeling and Analysis Using Queueing Petri Nets*, pages 293 – 311. Springer, Berlin / Heidelberg, 2010.

[LGF05] V. Liu, I. Gorton, and A. Fekete. Design-level performance prediction of component-based applications. *IEEE Transactions on Software Engineering*, 31(11):928–941, November 2005.

[LKQ+97] M Litoiu, Hamid Khafagy, Bin Qin, Anita Rass Wan, and Jerry Rolia. A Performance Engineering Tool and Method for Distributing Applications. In *CASCON '97 Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research.* IBM Press, 1997.

[LZGS84] 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., Upper Saddle River, NJ, USA, 1984.

[MBKR08a] Anne Martens, Steffen Becker, Heiko Koziolek, and Ralf Reussner. An empirical investigation of the applicability of a component-based performance prediction method. *Computer Performance Engineering, 5th European Performance Engineering Workshop, EPEW 2008, Palma de Mallorca, Spain*, pages 17–31, 2008.

[MBKR08b] Anne Martens, Steffen Becker, Heiko Koziolek, and Ralf Reussner. An Empirical Investigation of the Effort of Creating Reusable, Component-Based Models for Performance Prediction. *Component-Based Software Engineering*, pages 16–31, 2008.

[Mei10] Philipp Meier. *Transformation of Palladio-Model-Instances to Queueing-Petri-Nets*. PhD thesis, Karlsruhe Institute of Technology, 2010.

[MKBR10] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering - WOSP/SIPEW '10*, page 105, 2010.

[MKK11] Philipp Meier, Samuel Kounev, and Heiko Koziolek. Automated Transformation of Palladio Component Models to Queueing Petri Nets. In *(submitted to) 14th International ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE-2011)*, 2011.

[Mon10] Kevin Montagne. Tackling architectural complexity with modeling. *Communications of the ACM*, 53(10):46, October 2010.

[neo11] Neotys Neoload Load Testing Tool, 2011. `http://www.neotys.com/product/overview-neoload.html` – last accessed: June 14th, 2011

[OV11] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, 3rd edition, 2011.

[Pet05] Dorina C. Petriu. *Performance Analysis with the SPT Profile*, chapter 14, pages 205–224. Hermes Science Publishing Ltd., London, England, 2005.

[RCK+09] Jerry Rolia, Giuliano Casale, Diwakar Krishnamurthy, Stephen Dawson, and Stephan Kraft. Predictive modelling of SAP ERP applications: challenges and solutions. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 9:1—-9:9. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[San10] William H. Sanders. Möbius Manual. Technical report, University of Illinois, 2010.

[SBC09] Giuseppe Serazzi, Marco Bertoli, and Giuliano Casale. User-Friendly Approach to Capacity Planning Studies with Java Modelling Tools. *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, 2009.

[SMF+07] J. Sankarasetty, Kevin Mobley, L. Foster, T. Hammer, and T. Calderone. Software performance in the real world: personal lessons from the performance trauma team. In *Proceedings of the 6th international workshop on Software and performance*, pages 201–208. ACM, 2007.

[Smi86] Connie U Smith. The evolution of software performance engineering: a survey. In *Proceedings of 1986 ACM Fall joint computer conference*, pages 778–783, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[SW02] Connie U Smith and Lloyd G Williams. *Performance solutions: a practical guide to creating responsive, scalable software.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.

[THHF08] Dharmesh Thakkar, Ahmed E. Hassan, Gilbert Hamann, and Parminder Flora. A framework for measurement based performance modeling. *Proceedings of the 7th international workshop on Software and performance - WOSP '08*, page 55, 2008.

[TM06] Nidhi Tiwari and P. Mynampati. Experiences of using LQN and QPN tools for performance modeling of a J2EE application. In *CMG 2006 Conference*, volume 1, page 537. Computer Measurement Group, 2006.

[Ufi06] Alexander Ufimtsev. *Vertical Performance Modelling and Evaluation of Component-based Software Systems.* PhD thesis, University College Dublin, 2006.

[UPS+05] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):291—-302, June 2005.

[WFP07] Murray Woodside, Greg Franks, and Dorina C. Petriu. The Future of Software Performance Engineering. *Future of Software Engineering (FOSE '07)*, pages 171–187, May 2007.

[WHSB01] Murray Woodside, Curtis Hrischuka, Bran Selic, and Stefan Bayarov. Automated performance modeling of software generated by a design environment. *Performance Evaluation*, 45(2-3):107–123, July 2001.

[Woo02] Murray Woodside. Tutorial introduction to layered modeling of software performance. Technical report, RADS Lab at Carleton University, 2002.

[WPP+05] Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Hui Shen, Toqeer Israr, and Jose Merseguer. Performance by unified model analysis (PUMA). *Proceedings of the 5th international workshop on Software and performance - WOSP '05*, pages 1–12, 2005.

[WS98] Lloyd G Williams and Connie U Smith. Performance evaluation of software architectures. In *Proceedings of the first international workshop on Software and performance - WOSP '98*, pages 164–177, New York, New York, USA, 1998. ACM Press.

[XOWM06] Jing Xu, Alexandre Oufimtsev, Murray Woodside, and Liam Murphy. Performance modeling and prediction of enterprise JavaBeans with layered queuing network templates. *ACM SIGSOFT Software Engineering Notes*, 31(2):5, March 2006.

[Xu10] Jing Xu. Rule-based automatic software performance diagnosis and improvement. *Performance Evaluation*, 67(8):585–611, August 2010.

[ZBLG07] Liming Zhu, Ngoc Bao Bui, Yan Liu, and Ian Gorton. MDABench: Customized benchmark generation using MDA. *Journal of Systems and Software*, 80(2):265–282, February 2007.

[ZLBG07] Liming Zhu, Yan Liu, Ngoc Bao Bui, and Ian Gorton. Revel8or: Model Driven Capacity Planning Tool Suite. *29th International Conference on Software Engineering (ICSE'07)*, pages 797–800, May 2007.