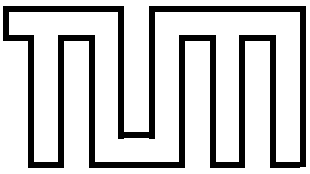
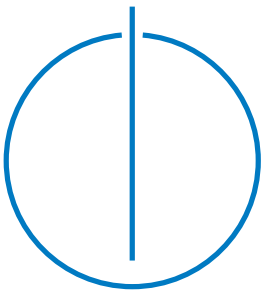


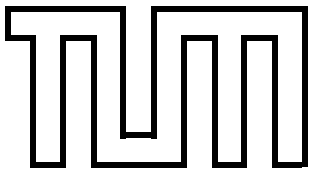
TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Wirtschaftsinformatik (I 17)
Univ.-Prof. Dr. Helmut Krcmar

# Continuous Performance Evaluation and Capacity Planning for Enterprise Applications

Andreas Brunnert, M.Sc.





TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Wirtschaftsinformatik (I 17)
Univ.-Prof. Dr. Helmut Krcmar

# Continuous Performance Evaluation and Capacity Planning for Enterprise Applications

Andreas Brunnert, M.Sc.

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

| | |
|---|---|
| Vorsitzender: | Univ.-Prof. Dr. Martin Bichler |
| Prüfer der Dissertation: | 1. Univ.-Prof. Dr. Helmut Krcmar |
| | 2. Univ.-Prof. Dr. Ralf Reussner (Karlsruher Institut für Technologie) |

Die Dissertation wurde am 15.07.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 07.10.2015 angenommen.

## Acknowledgement

Even though a dissertation is an effort that is undertaken by a single person, this person cannot succeed without the help of others. I would therefore like to take this opportunity to thank those that helped me during the dissertation journey.

First and foremost I would like to thank Prof. Helmut Krcmar who gave me the opportunity to pursue this topic under his supervision. In addition to the results of this dissertation, this opportunity led to the creation of a successful team of researchers that now carries on this topic. Due to the continuous support and encouragement of Prof. Krcmar the team was able to quickly establish itself within the international performance engineering community.

Furthermore, my thanks go to my colleagues in the performance team[1] at the fortiss institute. Without their ongoing support the research that led to this dissertation would not have been possible. Special thanks are due to Christian Vögele for helping me while conducting the first experiments using the performance model generator.

I would also like to thank all the students that contributed directly or indirectly to this dissertation as student workers or as part of their bachelor/master thesis. Some parts of this work would not exist without the help of Kilian Wischer, Markus Dlugi, Stefan Neubig and Simon Sprang.

This dissertation is also closely related to the work of the research group headed by Prof. Ralf Reussner at the Karlsruhe Institute of Technology[2]. His group has supported me with help and guidance whenever I was stuck while using their performance modeling and simulation tools. I am also very grateful that Ralf agreed to be the second referee for this dissertation.

Additional thanks are due to André van Hoorn with whom I have established the DevOps Performance working group within the research group of the Standard Performance Evaluation Corporation (SPEC)[3]. This working group provides an international platform for researchers and practitioners to tackle performance-related challenges within DevOps scenarios. It was a constant source for information and feedback.

Finally, I would like to thank my wife Christina and my parents for their unlimited support during the work on this dissertation. Without them, this work would not exist.

Munich, Germany, July 2015 Andreas Brunnert

[1] `http://pmw.fortiss.org/`
[2] `http://sdq.ipd.kit.edu/`
[3] `http://research.spec.org/working-groups/devops-performance-working-group/`

# Abstract

**Motivation and Goal** The need to continuously adapt enterprise applications (EA) to changes in the business environment led to several modifications in the software development process in recent years. These adjustments, often summarized by the term DevOps, include a tighter integration between development (Dev) and operation (Ops) teams as well as an increased frequency of releases of new EA versions. The goal of these modifications is to quickly deliver new features or bug fixes to the users, compared to traditional release cycles in which they are released in larger batches in a few major versions.

Performance characteristics (i.e., response time, resource utilization, and throughput) of an EA change whenever new features or fixes are introduced in new versions. Therefore, their performance needs to be continuously evaluated. Measurement-based performance evaluation approaches are often used for this purpose which require a test environment that is comparable to a production system. Maintaining such systems is labor intensive and expensive. If multiple deployments of the same EA exist it is often not feasible to maintain test instances for all these systems. Furthermore, not all deployments are known at the time of a release (e.g., for off-the-shelf products). The goal of this dissertation is to address these challenges by introducing model-based performance evaluation and capacity planning capabilities for EAs that avoid the need for performance test environments.

**Research Approach** This dissertation follows a design-oriented research strategy. It builds upon existing research in the area of software performance and open challenges in the domain of EA performance. New artifacts (i.e., concepts, approaches and software prototypes) are introduced as solution proposals and are continuously improved according to evaluation results of their utility. To evaluate their utility, evaluation methods such as controlled experiments and simulations are applied.

**Results** In a first step, this dissertation outlines conceptual and practical challenges for model-based performance evaluations. Afterwards, a solution to automatically generate models for performance evaluations is introduced. This solution is further improved and evaluated to allow for the construction of so called resource profiles. Resource profiles are models that describe the resource demand per transaction for each component of an EA. They are created during the software development process and allow for detecting performance changes in EA versions due to feature additions or bug fixes. Once a version is being released, resource profiles can be distributed by an EA vendor (EAV, i.e., software vendors or consulting companies). Using resource profiles, EA hosts (EAH, i.e., data center providers) and/or users (EAU, i.e., companies that source software from EAVs) can plan the required capacity for EAs for different workloads and hardware environments.

**Contribution to Research** Performance evaluation as part of the development and operation processes has been a research topic for several decades. However, current research is mostly focused on either evaluations in the development process (e.g., using software performance engineering (SPE) activities) or during operations (e.g., using application performance management (APM) activities). This work addresses the gap in between these two research areas. This gap has not been addressed so far as the closer integration of development and operation processes is a phenomenon that can only be observed in recent years. This dissertation outlines how resource profiles, as artifacts created and used during development, can be applied in the transition to and during operation in order to support performance evaluations throughout the life cycle of an EA.

**Contribution to Practice** As contribution to practice, the results of this dissertation ease the use of model-based performance evaluations. Nowadays, the effort required for manually creating a model for performance evaluations is often out of proportion compared to their benefits. This dissertation introduces a solution to generate performance models automatically. Furthermore, it outlines the use of such generated models throughout an EA life cycle. These capabilities avoid the need for maintaining expensive performance test environments and allow for performance evaluations that would not be feasible otherwise (e.g., for a wider variety of hardware environments). The knowledge about the resource demands of an EA and the ability to predict its performance for specific workloads and hardware environments help to estimate the required capacity for EAs more precisely and thus avoids unnecessary hardware purchases and associated costs in data centers.

**Limitations** The continuous performance evaluation and capacity planning capabilities introduced in this work can be applied for different types of EAs. However, this work evaluates these capabilities only for EAs built upon the Java Enterprise Edition (EE) standard. Additional limitations arise from the capabilities to simulate memory resource usage and the inability to measure hard disk drive demand on an appropriate granularity level on multiple platforms. Memory can only be simulated in a very simplistic way, the accuracy of prediction results for this resource is therefore questionable. That is why the resource profiles used in the evaluations in this work only depict the demand of software on central processing units (CPU) and network.

# Zusammenfassung

**Motivation und Ziel der Arbeit** Der Bedarf, Unternehmensanwendungen (UA) kontinuierlich an Änderungen in ihrem wirtschaftlichen Umfeld anzupassen, hat in den letzten Jahren zu vielen Modifikationen im Softwareentwicklungsprozess geführt. Diese Modifikationen werden häufig unter dem Begriff DevOps zusammengefasst und umfassen eine engere Integration zwischen Entwicklungs- (Dev) und Betriebsteams (Ops) sowie eine höhere Rate an Releases von neuen UA Versionen. Das Ziel dieser Anpassungen ist es, den Nutzern einer UA neue Funktionen oder Fehlerbehebungen schneller zugänglich zu machen als in traditionellen Releasezyklen, in welchen diese nur in großen Paketen in wenigen Hauptversionen zur Verfügung gestellt werden.

Performancecharakteristiken (wie Antwortzeit, Ressourcenauslastung und Durchsatz) von UA ändern sich sobald neue Funktionen oder Fehlerbehebungen in neuen Versionen eingeführt werden. Daher muss ihre Performance kontinuierlich evaluiert werden. Hierzu werden oft messbasierte Performanceevaluationsansätze verwendet, die eine Testumgebung benötigen, die vergleichbar mit einem Produktionssystem ist. Der Betrieb solcher Systeme ist arbeitsaufwendig und teuer. Sobald mehrere Installationen einer UA existieren, ist es oft nicht möglich, Testumgebungen für alle Installationen zu betreiben. Darüber hinaus sind nicht alle Installationen einer UA zum Zeitpunkt eines Releases bekannt (insbesondere bei Standardsoftware). Das Ziel dieser Dissertation ist es, diese Herausforderungen durch modellbasierte Performanceevaluations- und Kapazitätsplanungsansätze für UA zu adressieren, die keine Performancetestumgebungen benötigen.

**Forschungsansatz** Diese Dissertation folgt einer designorientierten Forschungsstrategie. Sie baut auf existierender Forschung im Bereich der Softwareperformance und ungelösten Problemen im Bereich der UA-Performance auf. Neue Artefakte (wie Konzepte, Ansätze und Softwareprototypen) werden in dieser Arbeit als Lösungsvorschläge eingeführt und kontinuierlich anhand von Evaluationsergebnissen ihrer Nützlichkeit verbessert. Um die Nützlichkeit der Artefakte zu evaluieren, werden Evaluationsmethoden wie kontrollierte Experimente und Simulationen eingesetzt.

**Ergebnisse** Im ersten Schritt zeigt diese Dissertation konzeptuelle und praktische Herausforderungen für modellbasierte Performanceevaluationen auf. Anschließend wird eine Lösung vorgestellt, die automatisch Performancemodelle für Performanceevaluationen generieren kann. Diese Lösung wird weiter verbessert und evaluiert um die Erstellung sogenannter Ressourcenprofile zu ermöglichen. Ressourcenprofile sind Modelle, die den Ressourcenbedarf pro Transaktion für jede Komponente einer UA beschreiben. Sie werden im Softwareentwicklungsprozess erstellt und ermöglichen es, Performanceänderungen in

einzelnen Versionen einer UA zu erkennen, die durch neue Funktionen oder Fehlerbehebungen verursacht werden. Sobald eine UA Version released wird, können Ressourcenprofile durch einen UA Anbieter (UAA, wie Softwarehersteller oder Beratungsunternehmen) mit der UA verteilt werden. Durch die Nutzung von Ressourcenprofilen können UA Betreiber (UAB, wie Rechenzentrumsdienstleister) und/oder Nutzer (UAN, also Unternehmen die Software von UAA beziehen) die erforderliche Kapazität für UAs für unterschiedliche Nutzerverhalten, Lastintensitäten und Hardwareumgebungen planen.

**Beitrag zur Forschung** Performanceevaluationen als Teil der Entwicklungs- und Betriebsprozesse sind seit vielen Jahrzenten ein Forschungsthema. Existierende Forschungsaktivitäten fokussieren sich jedoch eher entweder auf Performanceevaluationen im Entwicklungsprozess (zum Beispiel durch Software Performance Engineering (SPE) Aktivitäten) oder während des Betriebs (zum Beispiel durch Application Performance Management (APM) Aktivitäten). Diese Arbeit schließt die Lücke zwischen diesen Forschungsgebieten. Diese Lücke wurde bisher nicht adressiert, da die engere Integration von Entwicklungs- und Betriebsprozessen ein Phänomen ist, das erst in den letzten Jahren aufgekommen ist. Diese Dissertation zeigt, wie Ressourcenprofile als Artefakte, die in der Entwicklung erstellt und genutzt werden, während des Übergangs in den und im Betrieb eingesetzt werden können um Performanceevaluationen im gesamten Lebenszyklus einer UA zu unterstützen.

**Beitrag zur Praxis** Als Beitrag zur Praxis erleichtern die Ergebnisse dieser Dissertation die Anwendbarkeit von modellbasierten Performanceevaluationen. Heutzutage steht der Aufwand zur Erstellung von Modellen zur Performanceevaluation oft in keinem Verhältnis zum erwarteten Nutzen. Diese Dissertation führt eine Lösung ein, um Performancemodelle automatisch zu generieren. Weiterhin wird erläutert, wie solche generierten Performancemodelle im gesamten Lebenszyklus von UA genutzt werden können. Diese Fähigkeiten vermeiden den Bedarf an kostenintensiven Performancetestumgebungen und ermöglichen Performanceevaluationen die anders nicht möglich wären (zum Beispiel für eine größere Menge und Hardwareumgebungen). Darüber hinaus hilft das Wissen über den Ressourcenbedarf einer UA und die Fähigkeit Performance präziser zu prognostizieren, unnötige Hardwareanschaffungen und assoziierte Kosten in Rechenzentren zu vermeiden.

**Limitationen** Die kontinuierlichen Performanceevaluations- und Kapazitätsplanungsansätze, die in dieser Arbeit eingeführt werden, können für unterschiedliche UA Typen eingesetzt werden. In dieser Arbeit werden diese Fähigkeiten jedoch nur für UAs evaluiert, die auf Basis des Java Enterprise Edition (EE) Standards entwickelt wurden. Weitere Limitationen ergeben sich aus den Fähigkeiten den Hauptspeicherbedarf zu simulieren und den Festplattenbedarf auf allen Plattformen in der notwendigen Granularität zu messen. Der Hauptspeicher kann nur sehr vereinfacht simuliert werden, daher ist die Genauigkeit der Simulationsergebnisse eher fragwürdig. Aus den genannten Gründen repräsentieren die Ressourcenprofile in dieser Arbeit nur den Prozessor- und Netzwerkressourcenbedarf von Software.

# Contents

# List of Figures

# List of Tables

## List of Listings

## List of Abbreviations and Acronyms

| | |
|---|---|
| ACM | Association for Computer Machinery |
| API | Application Programming Interface |
| APM | Application Performance Management |
| AS | Application System (in chapters 6, 7, 8, 9: Application Server) |
| B | Browse |
| BIS | Business Information System |
| BISE | Business & Information Systems Engineering |
| BR | Branch |
| C | Component (in chapter 6: Clients, in chapter 7: Cores) |
| CD | Continuous Delivery |
| CF | Control Flow |
| CI | Continuous Integration |
| CPU | Central Processing Unit |
| CPUPE | CPU Prediction Error |
| CRM | Customer Relationship Management |
| CSV | Comma-separated Value |
| D | (Resource) Demand |
| Dev | Development |
| DML | Descartes Modeling Language |
| DU | Deployment Unit |
| EA | Enterprise Application |
| EAH | Enterprise Application Host |
| EASED | Workshop on Energy-Aware Software Engineering and Development |
| EAU | Enterprise Application User |
| EAV | Enterprise Application Vendor |
| EJB | Enterprise JavaBean |
| EMF | Eclipse Modeling Framework |
| EPC | Event-driven Process Chains |
| EPEW | European Workshop on Performance Engineering |
| ERP | Enterprise Resource Planning |
| ESB | Enterprise Service Bus |
| FCFS | First-come, First-serve |
| G | Generic |
| GB | Gigabyte |
| GC | Garbage Collector |
| HDD | Hard Disk Drive |
| HTTP | Hypertext Transfer Protocol |
| IC | Initial Cost |
| ICPE | International Conference on Performance Engineering |
| IO | Input/Output |

IPMI ..... Intelligent Platform Management Interface
IQR ..... Interquartile Range
IS ..... Information System
IT ..... Information Technology
Java EE ..... Java Enterprise Edition
JMX ..... Java Management Extension
JPA ..... Java Persistence API
JSF ..... JavaServer Faces
JSP ..... JavaServer Pages
JSS ..... Journal of Systems and Software
JVM ..... Java Virtual Machine
LQN ..... Layered Queuing Network
LT ..... International Workshop on Large-Scale Testing
M ..... Manage
MB ..... Megabyte
MCPU ..... Measured CPU Utilization
MMCPU ..... Measured Mean CPU Utilization
MMPC ..... Measured Mean Power Consumption
MMRT ..... Measured Mean Response Time
MOO ..... Multi-Objective Optimization
MRT ..... Measured Response Time
ms ..... Milliseconds
MT ..... Measured Throughput
OP ..... (Component) Operation
Ops ..... Operations
p ..... Probability
P/PU ..... Purchase
PCI ..... Performance Curve Integration
PCM ..... Palladio Component Model
PMW ..... Performance Management Work
PS ..... Processor Sharing
Q ..... Quartile
QN ..... Queuing Network
QoSA ..... International Conference on the Quality of Software Architectures
QPN ..... Queuing Petri-net
QUDOS ..... International Workshop on Quality-Aware DevOps
RAM ..... Random-Access Memory
RC ..... Relative Change
RDSEFF ..... Resource Demanding Service Effect Specification
RP ..... Resource Profile
RPC ..... Relative Power Consumption
RQ ..... Research Question
RT ..... Response Time
RTE ..... Response Time Error
RU ..... Rack Unit
S ..... Service Time
s ..... Seconds
SC ..... Server Count
SCM ..... Supply Chain Management

SCPU ..... Simulated CPU Utilization
SIPEW ..... SPEC International Performance Evaluation Workshop
SLA ..... Service-level Agreement
SMCPU ..... Simulated Mean CPU Utilization
SMPC ..... Simulated Mean Power Consumption
SMRT ..... Simulated Mean Response Time
SOA ..... Service-oriented Architecture
SOAP ..... Simple Object Access Protocol
SOSP ..... Symposium on Software Performance
SPE ..... Software Performance Engineering
SPEC ..... Standard Performance Evaluation Corporation
SRT ..... Simulated Response Time
SUT ..... System Under Test
T ..... Transaction
t ..... Time
TCO ..... Total Cost of Ownership
TPM ..... Transactions per Minute
U ..... User
UI ..... User Interface
UML ..... Unified Modeling Language
V ..... Version
ValueTools ..... International Conference on Performance Evaluation Methodologies and Tools
VM ..... Virtual Machine
W ..... Watt
w ..... Waiting Time
WG ..... Working Group
WI ..... Wirtschaftsinformatik
WOSP ..... Workshop on Software Performance
XML ..... eXtensible Markup Language
Y ..... Year

# Part A

# Chapter 1

# Introduction

## 1.1 Motivation and Background

Software engineering is a discipline concerned with the whole life cycle of software systems. It covers activities during software development and operation to ensure that such systems can be created and maintained with a high quality of service. An important quality factor of a software system is called performance. Performance is defined by the metrics response time, throughput and resource utilization (Jain, 1991). Guaranteeing performance requires a lot of activities throughout the software system life cycle. Activities to ensure that performance goals can be met during software development are summarized by the term software performance engineering (SPE) (Smith, 1981). Corresponding activities during operations are referred to as application performance management (APM) (Menascé, 2002b).

Software systems that support business processes are called enterprise applications (EA) (Grinshpan, 2012). The need to continuously adapt EAs to changes in the business environment led to several modifications in their development process in recent years (Humble/Molesky, 2011). These adjustments, often summarized by the term DevOps, include a tighter integration between development (Dev) and operation (Ops) teams as well as an increased frequency of releases of new EA versions (Erich/Amrit/Daneva, 2014; Humble/Molesky, 2011). The goal of these changes is to quickly deliver new features or bug fixes to the users, compared to traditional release cycles in which they are released in larger batches in a few major versions.

Nowadays, research in the fields of SPE and APM often tackles the concerns of development and operation teams independently from each other. Therefore, an integration between SPE and APM activities to address these newly emerging DevOps concepts from a performance perspective is missing (Perez/Wang/Casale, 2015). This dissertation aims to improve the integration of SPE and APM by supporting the transition of EAs from development to operation.

To ensure a smooth transition of new EA versions from development to operation from a performance standpoint, the perspectives of all parties involved in such a release process need to be considered. Development teams have to be aware of the performance impact of changes (i.e., feature additions, bug fixes or configuration changes) to address any

unforeseen performance regressions before a new EA version is released. Operation teams need to be able to ensure that sufficient capacity (i.e., soft- and hardware resources) is available to host an EA version. For these purposes, the teams need the input of EA users regarding the expected workload and their performance requirements. The goal of this dissertation is to introduce performance evaluation capabilities that support all parties involved in this process in an integrated way.

## 1.2 Research Objective and Guiding Questions

In order to support the software release activities outlined in the introduction, this dissertation proposes the use of model-based performance evaluation techniques. These techniques allow to predict the performance of EAs based on abstract descriptions of their performance-relevant aspects, hereafter called performance models. The effort of creating such models nowadays often outweighs their benefits (Brosig/Huber/Kounev, 2014; Kounev, 2005; Woodside/Franks/Petriu, 2007). Therefore, these techniques are rarely applied in practice (Koziolek, 2010; Mayer et al., 2011).

More commonly used approaches for performance evaluations are measurement-based. Measurement-based performance evaluations require test environments that are comparable to the final production systems in order to derive meaningful performance metrics (Jain, 1991). Due to the associated costs, it is often not feasible to maintain test environments for all deployments of an EA. Furthermore, measurements cannot be collected in scenarios in which new deployments need to be planned and test environments are not available. Model-based performance evaluations can address these shortcomings of measurement-based approaches.

To improve the applicability of model-based performance evaluations, this dissertation proposes a software solution to derive performance models automatically and provides systematic guidance on how such models can be used to evaluate the performance of an EA during the software release process. The Palladio Component Model (PCM) is used as performance modeling approach and the analysis focuses on EAs built upon the Java Enterprise Edition (EE) specification. Even though the analysis is focused on one technology, the model-based performance evaluation guidelines and concepts outlined in this dissertation are applicable for other technologies as well.

To achieve these objectives, this work is structured along the following four research questions:

**Research Question 1:** What are the challenges in applying model-based performance evaluations for enterprise applications?

The first research question focuses on achieving an understanding about the challenges of using model-based performance evaluations for EAs. To answer this question, existing literature regarding the application of performance models for EAs in different life cycle phases is reviewed and the challenges of applying performance models in an industrial project are presented.

**Research Question 2:** How can enterprise applications built upon the Java Enterprise Edition (EE) standard be automatically represented in performance models?

One of the key challenges for applying model-based performance evaluations is the effort required for creating performance models (Balsamo et al., 2004; Brosig/Huber/Kounev, 2014; Chen et al., 2005; Woodside/Franks/Petriu, 2007). This research question is therefore concerned with the automatic representation of Java EE applications in performance models. In order to create a performance model it is necessary to define which EA components and component relationships need to be represented. It is also important to characterize the data that needs to be collected to parametrize the model and how the required data can be collected. Afterwards, the data must be processed to transform it into a performance model. To simplify this process, a solution is proposed that can collect the required data and generate performance models for Java EE applications. These capabilities are built in a way so that they can be used for all EAs that comply with the Java EE standard.

**Research Question 3:** How can model-based performance evaluations support capacity planning and management processes for enterprise applications?

One of the main activities during the transition from development to operation from a performance perspective is the estimation of the required capacity for an EA deployment (Grinshpan, 2012). This research question is therefore concerned with finding a way of supporting this activity (hereafter called capacity planning or management) using performance models. Even though there exists a lot of research on capacity planning using performance models (Menascé et al., 2004), the organizational perspective is often ignored. However, one of the key challenges for capacity planning is the distribution of knowledge across different parties. Users of EAs need to specify their requirements in terms of performance and their expected workloads. Vendors of EAs know the internal details of these applications and can specify their performance-relevant aspects. Hosts that operate a data center know the hardware environment on which an EA can be deployed. Nowadays, there is a lack of a communication medium between them. This work addresses this lack by proposing the use of specifically formed performance models. These models can be distributed along with the EA binaries as soon as a new version is released and allow EA users and hosts to specify the workload and hardware environment for a deployment independently from each other.

**Research Question 4:** How can performance be continuously evaluated during the software development process using model-based techniques while an application is constantly modified?

As outlined in the introduction, the rate of changes that occur in the markets leads to a continuous need for companies to adapt their processes to such changes. The EAs that support these processes also need to be adapted rapidly whenever a change occurs. To accompany these requirements, modern software development processes are designed in a way that EAs are released very often with few feature additions or bug fixes instead of combining a lot of changes into few major releases. Performance characteristics of EAs can change for each feature addition or bug fix. Because it is not feasible to evaluate the performance of each EA version using appropriate test environments, a model-based approach is introduced to evaluate the performance impact of changes in new EA versions.

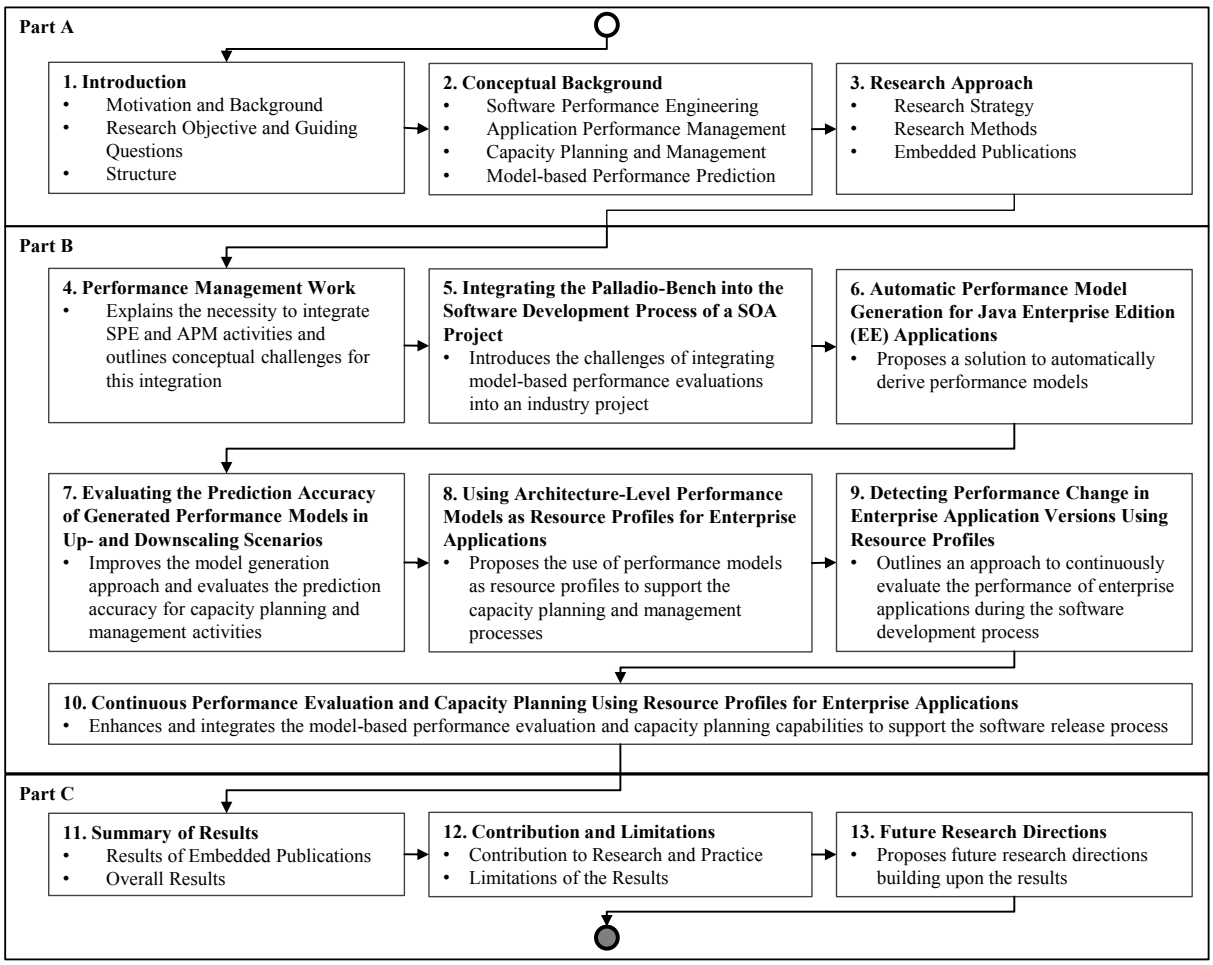



**Figure 1.1:** *Structure of this dissertation*

## 1.3 Structure

This dissertation consists of three parts: Part A, B and C. Figure 1.1 gives an overview of this structure. The arrows between the boxes depict the flow of argumentation.

Part A consists of three chapters, starting with chapter 1: Introduction which motivates the work, outlines the research objectives and the structure of this dissertation (this chapter). Chapter 2: Conceptual Background of part A introduces basic terms in the area of software performance and research directions in which this dissertation is embedded. Finally, chapter 3: Research Approach outlines the research strategy including research methods and resulting publications.

Part B is composed of seven publications (chapters 4 to 10) resulting from research done by the author as part of this dissertation. The first publication in chapter 4 explains the necessity to integrate SPE and APM activities as outlined in the introduction. The following chapter 5 introduces the challenges of integrating model-based performance evaluations into the software development process of an industry project. The next two publications in chapters 6 and 7 propose a way to automatically derive performance models from measurement (and, thus, APM) data and, therefore, address one of the main reasons why model-based SPE activities are not yet applied in practice. Afterwards, publication

8 proposes the use of these performance models as so called resource profiles for EAs to support the capacity planning and management processes. The following paper 9 builds upon the idea of resource profiles and proposes an approach to continuously evaluate the performance of EAs during software development. This approach allows to create resource profiles for each EA version that is being built and to use them to detect performance changes before an EA gets released. Finally, the publication in chapter 10 enhances and integrates the previously introduced performance evaluation and capacity planning capabilities to support the software release process.

Part C concludes this dissertation and is divided into three chapters. Chapter 11 summarizes the individual results of the publications included in part B and the overall results. Afterwards, chapter 12 outlines the contribution to research and practice and limitations of the results. Finally, chapter 13 proposes possible future research directions.

# Chapter 2

# Conceptual Background

This work is mainly influenced by two research areas that are closely interlinked, namely: SPE and APM. Figure 2.1 depicts the relationship of SPE and APM in the context of the life cycle of a software system. SPE describes the activities required to ensure that performance goals can be met when a system is being developed. APM on the other hand describes the activities required to achieve performance goals once a system is in production. SPE and APM activities are necessary throughout the entire life cycle of a software system; SPE activities are always required when a system is being changed (e.g., due to bug fixes or feature enhancements) and APM activities take over once these changes need to go live.

This dissertation focuses on SPE and APM activities for a specific type of software systems which are part of business information systems (BIS). These software systems are called EA or application systems (AS). BIS are defined as the information processing part of a corporation, whereas AS automate a specific task within an overall BIS (e.g., automating processes) (Ferstl/Sinz, 2008; Krcmar, 2010). EAs are defined similarly to AS as information backbones of corporations that support specific business functions (Grinshpan, 2012). EA and AS are therefore terms for the same type of software systems and are used interchangeably within this dissertation. Mostly, the term EA is used, the term AS is only used in the publication included in chapter 4.


**Figure 2.1:** *SPE and APM in the software system life cycle (Brunnert/van Hoorn, 2015)*

The performance of an EA is characterized by the metrics response time, throughput and resource utilization. Response times describe the time specific transactions need to complete, whereas throughput characterizes the amount of transactions that can be processed in a certain time frame (e.g., per second or minute). Resource utilization on the other hand describes how much of the available processing capacity of a resource has been used (e.g., central processing unit (CPU) utilization).

EAs have several characteristics that make it necessary to continuously evaluate and improve their performance. The most important reasons for this necessity are outlined by Grinshpan (2012) as follows:

1. *Enterprise applications support vital corporate business functions, and their performance is critical for successful execution of business tasks. [...]*
2. *Corporations inherently tend to grow by expanding their customer base, opening new divisions, releasing new products, as well as engaging in restructuring, mergers, and acquisitions. Business dynamics directly affects a number of application users, as well as the volume and structure of data loaded into databases. That means that tuning and sizing must be organic and indispensable components of the application life cycle, ensuring its adaptation to an ever-changing environment.*
3. *Each company is unique in terms of operational practice, customer base, product nomenclature, cost structure, and other aspects of business logistics; as such, enterprise applications cannot be deployed right after being purchased as they must undergo broad customization and be tested and tuned for performance before being released in production.*
4. *The typical enterprise application architecture represents server farms with users connected to the system from geographically distributed offices over corporate and virtual private networks.*
5. *Enterprise applications deal with much larger and complex data per a user's request as opposed to Internet applications [...]*
6. *The number of enterprise application users is significantly lower than that of Internet application users since their user communities are limited to corporation business departments. That number can still be quite large, reaching thousands of users [...]*
7. *End users work with enterprise applications not only through their browsers, as with Internet applications, but also through a variety of front - end programs (for example, Excel or Power-Point, as well as interface programs specifically designed for different business tasks). [...]*
8. *A signficant factor influencing the workload of enterprise applications is the rate of the requests submitted by the users – a number of requests per given time interval, usually per one work hour. Pacing defines an intensity of requests from the users and by the same token utilization of system resources.*

| Phase of Enterprise Application Life Cycle | SPE and APM Activities |
|---|---|
| (1) Sales | Capacity planning to determine hardware architecture to host an application |
| (2) Application deployment | Setting up hardware infrastructure according to capacity planning recommendations, application customization, and population with business data |
| (3) Performance testing | Performance tuning based on application performance under an emulated workload |
| (4) Application live in production mode | Monitoring application performance, tuning application to avoid bottlenecks due to real workload fluctuations |
| (5) Scaling production application | Capacity planning to accommodate an increase in the number of users and data volume[1] |

**Table 2.1:** *SPE and APM activities in the EA life cycle according to Grinshpan (2012)*

These reasons support the necessity to continuously adapt enterprise applications to changes in the business environment as outlined in the introduction and imply a continuous iteration from development (SPE) to operation (APM).

According to Grinshpan (2012), five EA life cycle phases shown in table 2.1 exist. The author only considers phases starting from the time an EA is sold and during operation (APM). Therefore, development (SPE) is not considered. However, this summary includes two important transitioning phases (1 and 2) and corresponding activities to achieve performance goals that are often not considered in SPE and APM definitions.

A key activity in the transitioning phases (1 and 2) is capacity planning. Capacity planning deals with the question of when capacity is adequate with regard to performance requirements and further constraints (e.g., costs) for a specific EA deployment. This activity is sometimes associated with SPE (Woodside/Franks/Petriu, 2007) and sometimes with APM (Menascé, 2002b) because it is a key activity as soon as a new EA is being moved from development to operation. Furthermore, it is used to evaluate the influence of seasonal patterns (e.g., Christmas shopping) or major feature changes on the required capacity once an EA is in production (see phase 5 in table 2.1). In this case, it is often also referred to as capacity management.

Even though the general idea of the phases and activities outlined in the work of Grinshpan (2012) is in line with the thinking in this dissertation, a more precise definition of SPE and APM activities can be found in other works. These works are outlined in the following sections.

[1] This dissertation refers to this activity as capacity management.

## 2.1 Software Performance Engineering

SPE is defined by Smith (2007) as:

> *"[...] a systematic, quantitative approach to constructing software systems that meet performance requirements."*

According to her definition it is a set of activities during system development. A more detailed list of SPE activities can be found in the work of Woodside/Franks/Petriu (2007), namely:

1. *Identify concerns*
2. *Define and analyze requirements*
3. *Predict performance from scenarios, architecture, and detailed design*
4. *Performance testing*
5. *Maintenance and evolution: predict the effect of potential changes and additions [...].*
6. *Total system analysis: consider the planned software in the complete and final deployed system. [...]*

The first activity, *identifying concerns* is necessary to identify the resources and software elements that need to be dealt with in order to achieve the desired performance. Afterwards, the *performance requirements* for specific workloads need to be defined. The workload can be defined for EAs by the amount of users interacting with a system simultaneously and their behavior. The performance requirements can be specified by defining upper and/or lower limits for the performance metrics response time, throughput and resource utilization.

To ensure that performance requirements can be met, a key element of the SPE definition (Smith, 1981; Smith, 2007; Woodside/Franks/Petriu, 2007) is the use of performance models. Performance models allow for performance predictions for situations that cannot be tested on a real system. One step in the SPE process according to Woodside/Franks/Petriu (2007) is therefore to *predict the performance*. The authors explicitly note that this should be done early from scenario descriptions or design documents. This allows to estimate performance at a time when no implementation is available. The authors also suggest that performance modeling can be used later in the development process, when an implementation is available. However, most of the work on performance modeling as part of SPE is still focused on early design time performance models (Danciu et al., 2015b; Koziolek, 2010). Different performance model types are introduced in section 2.4.

Once an implementation of a software system is available, *performance tests* can be performed. These tests evaluate if the current state of a software system can handle the expected workload while achieving the performance requirements. A key requirement for such performance tests is the availability of a test environment that is comparable to the production system. If this precondition is not satisfied, it is hard to derive meaningful results due to varying performance characteristics of the underlying environment configurations (e.g., CPU core count and speed).

The next two activities, namely the *maintenance and evolution* and *total system analysis*, are not considered to be part of SPE in this dissertation (this interpretation is in line with the initial work about SPE by Smith (1981)). The main reason for this interpretation is that Smith (2007) considers SPE as activities during development, whereas Woodside/ Franks/Petriu (2007) partially include operations. This work distinguishes SPE and APM by the life cycle phase, therefore, *maintenance and evolution* activities are considered part of APM. The *total system analysis* outlined by Woodside/Franks/Petriu (2007) can be interpreted twofold: by calling it capacity planning as they consider a planned software deployment or as performance debugging activity during operation, when a problem occurs. Therefore, the *total system analysis* for a planned software deployment is called capacity planning and the debugging steps for an existing deployment are considered part of APM as outlined in the next section.

## 2.2 Application Performance Management

APM is defined by Menascé (2002b) as a set of activities used by organizations to ensure that their EAs reach their performance goals. In contrast to SPE, the focus of APM is on the operation phase. This phase has different challenges compared to the development phase. One of the main performance goals in this phase is to ensure that service-level agreements (SLA) are met. Examples for such SLAs are upper limits for response time values for specific business transactions or throughput goals for predefined time frames.

Menascé (2002b) outlines two basic approaches to APM: a reactive and a proactive approach. As the names suggest, the reactive approach fixes problems only as they occur whereas the proactive approach tries to avoid problems from the beginning. Both of these APM approaches rely on monitoring systems to continuously collect measurements from the production systems. These measurements need to reflect the target metrics required to evaluate SLAs.

As soon as the measurements collected by monitoring systems indicate a possible SLA violation, APM tools should be able to trigger alarms (Menascé, 2002b). These alarms can either be send to technical staff within an organization to trigger manual action or to solutions that can automatically address performance problems. One type of such solutions that gained a lot of attention in recent years is the dynamic resource allocation in virtualized environments as it can add new (soft- and/or hardware) resources to a system on demand. Unfortunately, most EA architectures are not able to scale-up and -down as dynamic as the resources can be added or removed from a system (Vaquero/ Rodero-Merino/Buyya, 2011). An even more problematic attribute of such systems is that resources are shared between multiple EAs and performance characteristics cannot be guaranteed over an extended period of time. It is therefore critical to plan ahead for each EA deployment to provision the appropriate capacity for the defined performance goals whether virtualized solutions are used or not. The next section explains the main tasks during capacity planning and management.

## 2.3 Capacity Planning and Management

As outlined in the previous two sections, SPE and APM focus on activities in different life cycle phases of an EA. A key activity as soon as an EA is being moved from development to operations is capacity planning. According to Grinshpan (2012), who uses "application sizing" synonymously to "capacity planning", its goal can be defined as follows:

> *"Sizing delivers the estimates of hardware architecture that will be capable of providing the requested service quality for the anticipated workload."*

The author continues by describing capacity planning as an activity to eliminate boundaries (Grinshpan, 2012). These boundaries are defined by hardware resources (e.g., number of CPU cores or network throughput) or software settings (e.g., number of possible connections to a database) (Grinshpan, 2012). Therefore, a key element during capacity planning is to identify which boundaries need to be eliminated to achieve the desired level of performance.

Menascé/Almeida (2002) describe the goal of capacity planning as to provide adequate capacity for an EA. According to their definition, an EA...

> *"[...] has adequate capacity if the service-level agreements are continuously met for a specified technology and standards, and if the services are provided within cost constraints."*

Even though both definitions of capacity planning describe the notion of providing sufficient hardware for the expected workload, the second one also outlines an important limitation. This limitation is that the provisioning of hardware and software is limited by constraints set by the business environment in which an EA needs to be introduced. Examples for such constraints are costs in terms of initial purchasing cost or maintenance cost (including labor cost) in the long term. All these costs are further constrained by existing contracts with vendors providing the software and hardware systems.

To accompany these costs, a cost model needs to be created that describes the dependency between specific hardware environment and software settings and the resulting costs (Menascé/Almeida, 2002). The input for such a cost model is therefore a technical description of the required hard- and software environment for the expected workload of an EA.

As data centers tend to grow continuously it is even important to consider whether it is still feasible to add a new EA within the available space during capacity planning. If this is not the case, existing servers need to be replaced with more powerful ones or additional space for the data center needs to be acquired. Another ever growing concern is the energy consumption which is steadily becoming one of the major cost drivers in data centers nowadays (Poess/Nambiar, 2008). Due to the complexity of the corresponding capacity provisioning tasks, hosting providers that provide the required capacity on demand are increasingly used in recent years. However, as mentioned before, even these providers need to plan their capacity accordingly. Furthermore, a customer of such providers also needs to calculate the cost for the expected level of service.

### Capacity Management: Continuous Capacity Planning

Although the terms capacity planning and capacity management are often used synonymously (van Hoorn, 2014), this work makes a distinction between both terms. The term "capacity planning" is used to describe the activity of initially estimating the required capacity whereas capacity management describes the continuous application of "capacity planning" as soon as an EA is in production.

In capacity management scenarios there are basically two ways of adding additional resources if additional capacity is required (van Hoorn, 2014): scaling horizontally (scaling out) and scaling vertically. Scaling horizontally means adding additional servers of the same type, whereas scaling vertically means adding additional resources to the existing servers or replacing existing resources with faster ones.

To estimate the required capacity of an EA for a specific workload without the need to test it on a real system, performance models need to be used. They are introduced in the next section.

## 2.4 Model-based Performance Prediction

As outlined in the previous sections, performance models play a key role in SPE activities during software development as well as during capacity planning and management activities. This section introduces three different types of performance models that are often used for these activities.

Figure 2.2 depicts a generic performance modeling and prediction process. A performance model of a software system needs to represent the workload, software architecture and hardware environment. The workload describes the user behavior and their count or arrival rate. The software architecture can be depicted in different granularity levels but at least the performance-relevant aspects (e.g., resource demands) need to be represented. The third layer that needs to be represented is the hardware environment. This includes for example the amount of CPU cores in a server that are available to process requests. All performance modeling approaches allow for a combined description of these three aspects. The way of their representation differs but they always need to be considered when modeling a software system.

A performance model that describes these three aspects can be used as input for an analytical solver or simulation engine to predict performance. The results of an analytical solution or a simulation of a model are usually the performance metrics response time, throughput and resource utilization. The advantage of analytical solvers is that they are very fast, whereas simulations are slow. However, the accuracy of analytical solutions is often lower because more simplifications are assumed to make them solvable (Menascé et al., 2004).

Performance models can be represented in a variety of different notations that all rely on different solution techniques (Balsamo et al., 2004; Cortellessa/Di Marco/Inverardi,

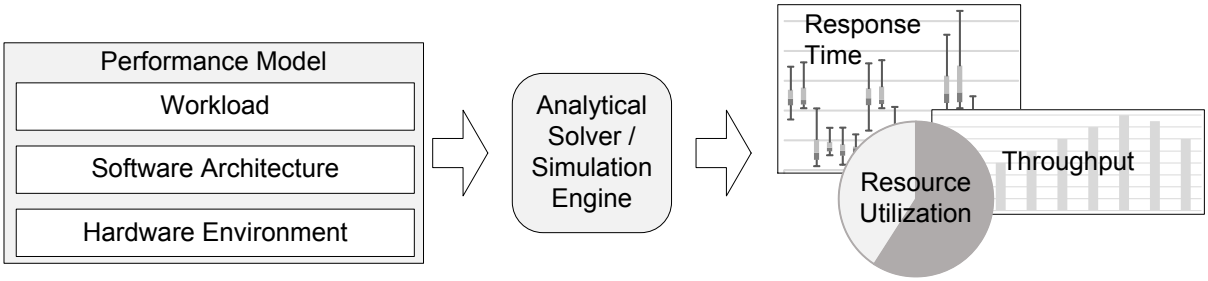



**Figure 2.2:** *Performance modeling and prediction*

2011; Koziolek, 2010). A basic classification can be made between classic notations such as (Layered) Queuing Networks ((L)QN), Stochastic Petri Nets and Stochastic Process Algebras and modern architecture-level notations such as PCM. A key difference between the classic and the modern performance modeling notations is that the modern notations allow for a separate specification of workload, software architecture and hardware environment whereas the classic notations depict these aspects in a single monolithic model.

Even though modern techniques allow for representing the workload, software architecture and hardware environment independently from each other, key ideas for their solution derive from existing notations. As this work intensively uses PCM which uses concepts of queuing theory to predict performance using simulations, the basic QN modeling notation and its extension LQN are introduced before the PCM meta-model and its solution techniques are explained.

**Queuing Networks** Resources of a hardware environment (e.g., CPUs or hard disk drives (HDD)), are specified in queuing networks using individual service stations with queues that are connected with each other. Each service station with queue in these networks is specified using a notation similar to the one shown in figure 2.3. This figure depicts, that the time ($t$) a transaction spends on a resource depends on the waiting time ($w$) in a queue and on the service time ($S$) on a service station. A service station could denote a CPU core or a HDD for example. The queue in front of a service station represents the scheduling algorithm that is used to allocate jobs on these resources. Figure 2.3 furthermore depicts the arrival rate of jobs ($\lambda$).

The content of figure 2.3 is also known as *G/G/1* queue. In this notation, the first *G* (generic) specifies the inter-arrival times of jobs that can be any generic distribution, the second *G* specifies a distribution of service times for the service station and *1* specifies that there is only one service station.

This model of a service station including its associated queue can be modified and extended in several ways. In a first step, the amount of service stations that is available to service jobs can be increased (e.g., to represent an increase of CPU cores). Furthermore, the type of queuing mechanism used to schedule jobs can be modified. It is also possible to increase the amount of queues for a specific type of service station, even though this possibility is not used in this dissertation.

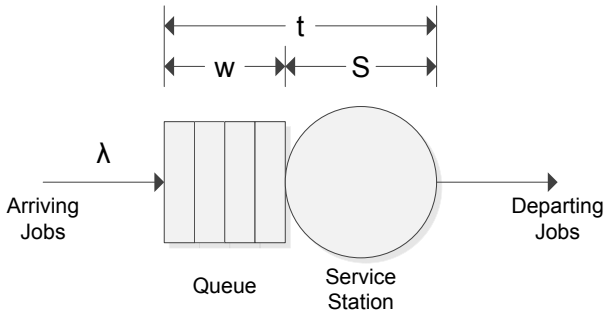



**Figure 2.3:** *Single service station with a single queue (adapted from Menascé* et al. (2004)*)*

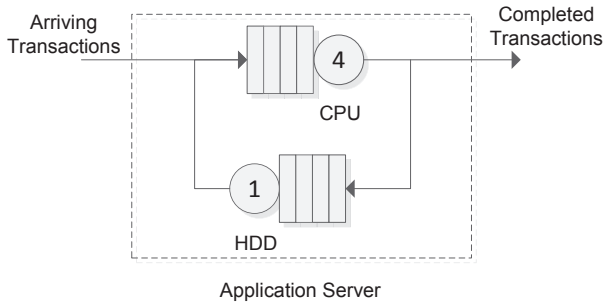



**Figure 2.4:** *Example for a queuing network of an application server machine (adapted from Menascé* et al. (2004)*)*

Common scheduling strategies for computer systems are (Bolch et al., 1998; Menascé et al., 2004): Fist-come, First-serve (FCFS) and Processor Sharing (PS). FCFS queues process jobs as they arrive. PS on the other hand, gives each job in a queue the same amount of processing time. This means, if $n$ processes are in a queue, each job receives $1/n$ of the available service time provided by the service stations. PS is a special case of the round robin scheduling discipline (Menascé et al., 2004), as the jobs are distributed equally to the available service stations, but return to the queue if $1/n$ of the available service time was not sufficient for their completion.

In order to represent hardware environments using queues, the individual queue specifications are combined to a network of queues. An example for such a queuing network is shown in figure 2.4. This figure depicts the hardware resources CPU and HDD of an application server machine. In the case of this example, the machine contains four CPU cores and one HDD. Transactions processed by the application server are always processed by the CPU and sometimes need to load data from the HDD as well. The example in figure 2.4 represents an open queuing network as transactions arrive in a network and leave it once the transaction processing is completed. If the entry and exit arrows of this network would be connected and form a loop, such a network would be called closed network. In such cases, a transaction never leaves the network.

The decision between specifying a closed or open queuing network is also a decision regarding the specification of the workload on a system. In open queuing networks, workload intensities are specified by arrival rates, whereas workload intensities are specified by the population of transactions for closed networks. The number of transactions within a closed queuing network is therefore constant (Menascé et al., 2004).

In order to simulate different workloads using a queuing network, not only the arrival rate or the amount of transactions, but also different types of transactions need to be represented. Transactions are represented using so called customer classes (Menascé et al., 2004). For each of these classes of customers, their workload intensity and their service demands at each resource needs to be specified. Queuing networks with multiple customer classes are called multi-class networks. Once the customer classes, the available resources and their queuing behavior are specified, performance can be predicted using analytical or simulation techniques.

**Layered Queuing Networks** One of the key drawbacks of traditional queuing networks is that they can only represent transactions that use one resource at a time. An extension called LQNs has therefore been proposed by Woodside et al. (1995). A key feature that distinguishes LQNs from QNs is the fact that they allow for simultaneous use of resources and represent the resulting queuing behavior of such a nesting (Franks et al., 2009). The authors call this simultaneous resource possession. An example that is given by Franks et al. (2009) is a server that calls another server and waits for the results of this call. The simultaneous resource possession furthermore supports a better representation of software elements in performance models such as thread pools in addition to the more hardware-oriented specifications in traditional QNs.

An example for an LQN that depicts software as well as hardware elements can be found in figure 2.5. The example in this figure shows users interacting with a server-side application using a web interface over the hypertext transfer protocol (HTTP). The key model elements that are used in this figure are *Tasks, Entries, Demands, Calls* and *Host Processors* (Woodside, 2013). The interacting elements in LQNs are *Tasks*, they are specified using bold rectangles in figure 2.5. In this example, the *Tasks* are *Users*, *HTTP Server Threads* and *AppServer Threads*. Each of these *Tasks* can have one or multiple *Entries*, specified using the rectangles attached to the *Tasks*. *Entries* define what the interacting elements can do. In this example, *Users* can execute HTTP requests, *HTTP Server Threads* can service web requests by the *Users* and *AppServer Threads* can either execute *Browse* or *Purchase* operations. The relationship of *Entries* is specified by *Calls*, in this example, the *HTTPRequest* entry calls the *WebService* and this entry calls either the *Browse* or *Purchase* entry. To represent the probability of control flows of requests, each *Call* is associated with a probability. In this example, the probability that the *HTTPRequest* entry calls the *WebService* entry is one (*y=1*) as only one *Call* exists between both layers. As multiple *Calls* exist between the *WebService* entry on the HTTP server layer and the *Browse* and *Purchase* entries on the AppServer layer, their probability is specified individually (*y=0.6* for the *Call* from *WebService* to *Browse* and *y=0.4* for the *Call* from *WebService* to *Purchase*).

In order to derive performance metrics from such models, their service demands, available service stations and queues need to be defined. The service demand in LQNs is specified using *Demands* in *Entries*. For example, the *Browse* entry requires 20 milliseconds (ms) of service time (in this case interpreted as CPU demand) on the *AppServer Machine*. The service time is specified relative to a *standard processor*. The *AppServer Machine* is a *Host Processor* similar to the other two machines depicted as ellipses in the figure (*User Device* and *HTTP Server Machine*). A *Host Processor* has a queue and a specific way of scheduling its tasks, similar to queues in traditional QNs. A *Host Processor* furthermore specifies its speed ratio relative to the *standard processor*, so that the service demands can

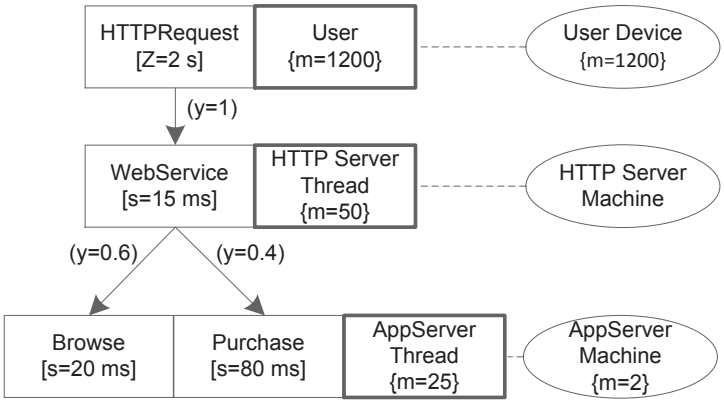



**Figure 2.5:** *Example for a layered queuing network of a multi-tier application (adapted from Woodside* (2013)*)*

be uniformly specified in a LQN. The amount of service stations in a *Host Processor* is given by the variable $m$ (multiplicity). In the case of this example, the *AppServer Machine* contains two service stations ($m=2$), whereas the *HTTP Server Machine* only contains one service station. In order to represent 1200 concurrent users, the multiplicity of *User Devices* is specified as 1200.

To specify the workload intensity and software contention, each *Task* also contains a variable $m$ that specifies its multiplicity. For example, figure 2.5 specifies the amount of *User* tasks to be 1200, translating directly into concurrent users of the system. Limitations introduced by software are represented by the multiplicity of the *HTTP Server Thread* and *AppServer Thread Tasks*. As only a limited amount of these threads is available, the request processing in this system is limited by these numbers. In order to further specify the behavior of users in a LQN, a variable $Z$ can be used in *Entries* to specify think times, that do not place load on the associated *Host Processors*. In the example in figure 2.5, each user waits 2 seconds (s) between two requests. Similar to QNs and other performance modeling notations, complete LQNs can be used as input for an analytic solver or a simulation engine to derive performance metrics.

**Palladio Component Model** Even though LQNs allow for a more realistic representation of software systems, their abilities to represent software independently from the workload and hardware environments are still limited. Whenever something needs to be changed in the workload (e.g., switching between different usage profiles), the software architecture (e.g., changing the control flows) or the hardware environment (e.g., by introducing new servers and changing the allocation of software components) an LQN needs to be completely restructured as it combines all these aspects in one monolithic model. In order to address this challenge, architecture-level performance models have been introduced (Koziolek, 2010). These models depict the performance-relevant aspects of a software architecture independently from the workload and hardware environment.

An example of an architecture-level performance model is PCM that is used throughout this dissertation as modeling notation. The PCM meta-model represents the performance-relevant aspects of a software system separately from the workload and the hardware environment (Becker/Koziolek/Reussner, 2009). Performance-relevant aspects of a software system are represented in a so called repository model. This model contains components of

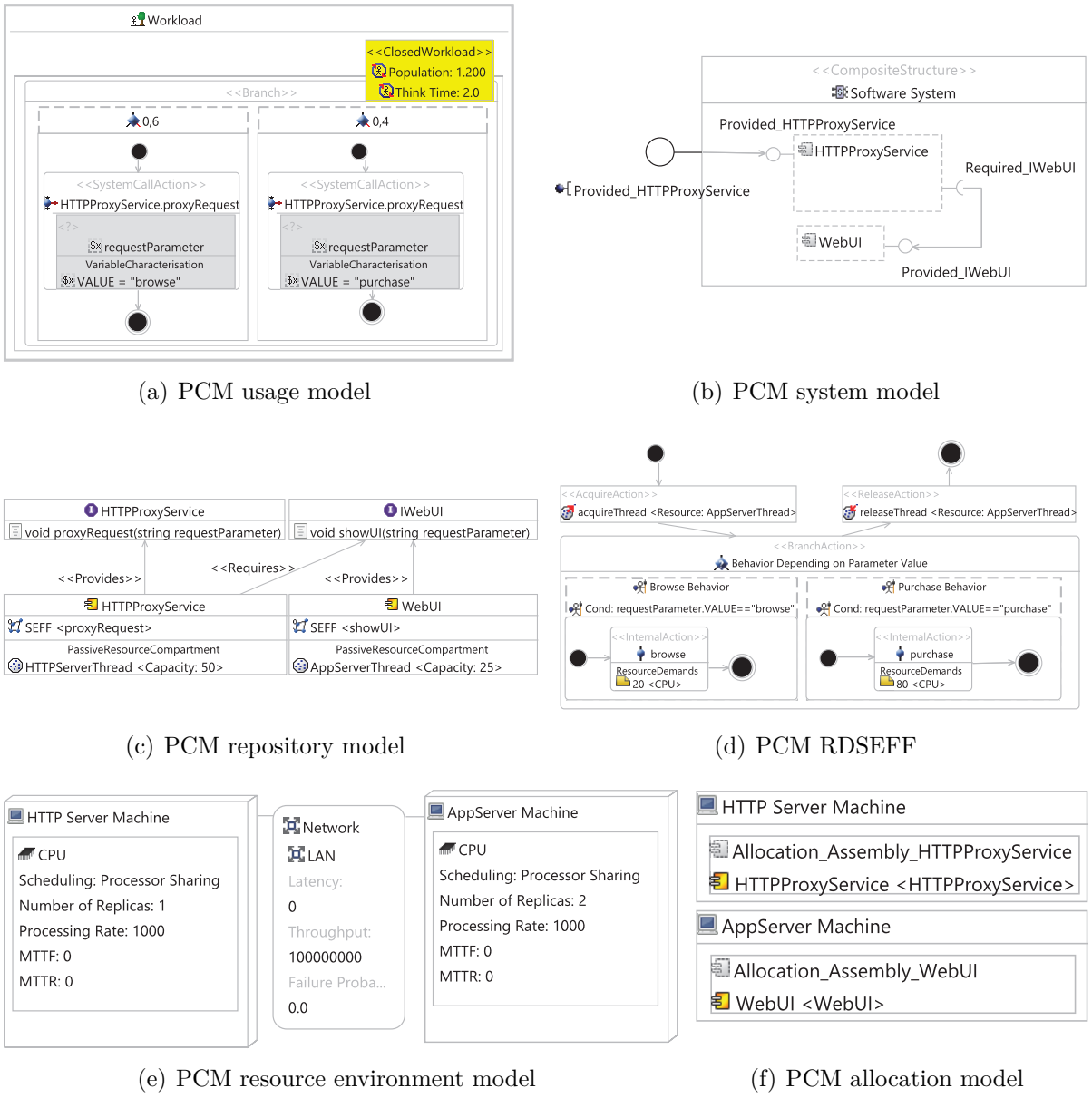
a software system and their relationships. The control flow of a component operation, its resource demand and parametric dependencies are also specified in this model. Components are assembled in a system model to represent an application. The workload on a system is described in a usage model. The remaining two model types in PCM describe the hardware environment: A resource environment model allows to specify available resource containers (i.e., servers) with their associated hardware resources (e.g., CPU or HDD). An allocation model specifies the mapping of system model elements on resource containers.

To showcase the difference between a LQN- and a PCM-based representation of the same software system, the LQN example from figure 2.5 is represented in an extended form as PCM model in the following. The PCM representation of the LQN example is shown in figure 2.6. This figure contains the representation of all model layers the PCM meta-model is composed of. Starting from the top left, the model represented in figure 2.6(a) is a PCM usage model and specifies the workload. It represents the behavior of a single user, a user population of 1200 and a think time of 2 seconds between each user interaction. A major difference to the LQN-based representation is that the probability of calling *browse* or *purchase* is now represented by specifying a parameter in the call to the *HTTPProxyServer* represented in this usage model. In the LQN example, this probability has been specified for *Calls* executed by the *WebService*, even though it is actually a decision of the user.

In order to call an operation from a usage model it is necessary to specify a PCM system model that defines a software system and thus the boundaries of a performance evaluation. The PCM system model for this example is shown in figure 2.6(b). This model contains two components that represent the system under evaluation. In the case of this example, these components are called *HTTPProxyService* and *WebUI*. The PCM system model only specifies the *HTTPProxyService* as externally accessible, thus, prohibiting that the *WebUI* component can be used directly.

The *HTTPProxyService* and *WebUI* components represent individual elements of a software system. Their control flow and resource demanding behavior is specified in a PCM repository model. The PCM repository model for the example is shown in figure 2.6(c) and contains the detailed component specifications of the *HTTPProxyService* and *WebUI* components. Both components specify their provided operations using interfaces (i.e., *HTTPProxyService* and *IWebUI*). The components themselves represent a specification of an implementation that complies with that interface. It is furthermore specified that the *HTTPProxyService* component requires a component that implements the *IWebUI* interface in order to function. This dependency is required to call one component from another. To also represent the software thread limits for each of these components, passive resources are used. Similar to the example in figure 2.5, the *HTTPProxyService* can process at most 50 threads at any point in time, whereas the *WebUI* can only process 25 threads concurrently.

To specify the behavior of specific component operations, so called Resource Demanding Service Effect Specifications (RDSEFFs) are used. The RDSEFF for the *showUI* operation of the *WebUI* component is shown in figure 2.6(d). When a request is processed by the system, it first tries to acquire a free thread. If no thread is available, it waits until one comes available in a FCFS manner. To specify the resource demand difference depending on the input parameter of the operation, a so called branch action is created. This branch action specifies that the operation requires only 20 ms CPU time when it is called using




(a) PCM usage model

(b) PCM system model

(c) PCM repository model

(d) PCM RDSEFF

(e) PCM resource environment model

(f) PCM allocation model

**Figure 2.6:** *Example for a PCM representation of the LQN example*

the parameter *browse*, whereas it consumes 80 ms CPU time when it is called using the parameter *purchase*. As soon as the request processing is finished, it returns the thread to the pool and finishes the request processing. The behavior of the *proxyRequest* operation of the *HTTPProxyService* is not shown as it only passes the parameter through to the *WebUI* component and handles the thread allocation similarly.

Before a PCM model can be used for performance evaluations, a hardware environment needs to be specified. This is done in a PCM resource environment model. A resource environment model for the example is shown in figure 2.6(e). It contains two machines from the previous example and specifies that the *HTTP Server Machine* has only one CPU core, whereas the *AppServer Machine* contains two CPU cores. It is important to note that for each resource in a server PCM allows to specify the processing rate. The resource demand values in the RDSEFFs are interpreted relative these values. In this work, a simulated time step is defined as one second and a processing rate of 1000 as units

of work per second and thus milliseconds. The network latency of the linking resource in figure 2.6(e) is specified in seconds, whereas its throughput is specified in bytes per second. The latency is set to zero to disable the network evaluation, as network has not been part of the previous example.

To map the system model components to the available servers in a resource environment, an allocation model needs to be specified, like the one in figure 2.6(f). It shows that the *HTTPProxyService* component is mapped to the *HTTP Server Machine*, whereas the *WebUI* component is mapped to the *AppServer Machine*. This complete set of models can now be used as input for simulation engines or analytical solvers to predict performance. It is also possible to transform PCM models into a LQN model in order to use existing LQN solvers.

As shown by translating the LQN example into a PCM-based representation, the modular approach of architecture-level performance models provides a better separation of concerns. For example, by using parameters the usage behavior specification can be handled independently from component specifications and their operation behavior descriptions. Furthermore, the ability to explicitly represent components of a software system allows to evaluate deployment changes more easily. An example would be to move a component from one server to another. If a request that is just represented as a resource demand value in a LQN is processed by multiple components in a system, it is hard to evaluate such changes using LQNs. In the example outlined in this section, an interesting question could be if the *HTTPProxyService* could run directly on the *AppServer Machine* to save space and energy. PCM would allow to simply change the allocation by moving the component from one server to another. Such things are not possible using monolithic models such as LQNs. Due to the advantages of architecture-level performance models, they are used throughout this dissertation instead of other performance modeling notations.

In order to derive performance predictions based on PCM models, two existing simulation engines are used throughout this work: the process-oriented simulation engine SimuCom (Becker/Koziolek/Reussner, 2009) and the event-driven simulation engine EventSim (Merkle/Henß, 2011). Both engines translate PCM models into Java code using model-to-text transformations and execute the resulting Java code to start a simulation. The key difference between both engines is that the process-orientation of SimuCom reduces its scalability characteristics as each simulated user is represented as one Java thread. EventSim has better scalability characteristics as the event-driven approach allows for simulations with a lot less Java threads that need to be executed concurrently. However, as EventSim does not support the full feature-set of SimuCom, it is only used when the application of SimuCom is not possible anymore.

# Chapter 3

# Research Approach

This chapter outlines the research strategy and methods that are applied in this work. Afterwards, it provides an overview of publications embedded in this dissertation and related research publications that have been (co-)authored.

## 3.1 Research Strategy

Research within the information systems (IS) field is often a two-step process. In a first step, IS research tries to understand existing socio-technical phenomena by applying quantitative and qualitative research methods from the social sciences. Once a phenomenon is understood and areas of improvement can be identified, IS research addresses these areas using design-oriented methodologies often adapted from computer science.

This dissertation follows a design-oriented research strategy. It builds upon existing research in the area of software performance and its challenges for EAs. To address the existing challenges, new artifacts (i.e., concepts, approaches and software prototypes) are introduced as solution proposals and are continuously improved according to evaluation results of their utility.

In order to evaluate the utility of artifacts that result from design-oriented research, several evaluation methods exist. An overview of design evaluation methods according to Hevner et al. (2004) is shown in table 3.1. These evaluation methodologies can be categorized into five different classes: observational, analytical, experimental, testing and descriptive. To evaluate the developed artifacts, only evaluation methods of the experimental and descriptive categories are applied.

In addition to the methods of artifact creation (e.g., software prototyping) and the four different types of evaluation methods, literature reviews are applied to construct and validate the proposed artifacts. The following section describes how these research methods are used.

| Category | Method |
|---|---|
| 1. Observational | **Case Study:** Study artifact in depth in business environment<br>**Field Study**: Monitor use of artifact in multiple projects |
| 2. Analytical | **Static Analysis:** Examine structure of artifact for static qualities (e.g., complexity)<br>**Architecture Analysis:** Study fit of artifact into technical information system architecture<br>**Optimization:** Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior<br>**Dynamic Analysis:** Study artifact in use for dynamic qualities (e.g., performance) |
| 3. Experimental | **Controlled Experiment:** Study artifact in controlled environment for qualities (e.g., usability)<br>**Simulation:** Execute artifact with artificial data |
| 4. Testing | **Functional (Black Box) Testing:** Execute artifact interfaces to discover failures and identify defects<br>**Structural (White Box) Testing:** Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation |
| 5. Descriptive | **Informed Argument:** Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility<br>**Scenarios:** Construct detailed scenarios around the artifact to demonstrate its utility |

**Table 3.1:** *Design evaluation methods (adapted from Hevner* et al. (2004)*)*

## 3.2 Research Methods

**Literature Reviews** As outlined in the previous section, this work follows a design-oriented research strategy. Nevertheless, literature reviews are continuously applied to ensure that the introduced concepts and approaches are novel and to put the results in context of the existing body of knowledge.

Only one co-authored publication (Danciu et al., 2015b) is a pure literature review and follows an approach called systematic literature review defined by Kitchenham/Charters (2007). The purpose of the literature reviews in the other publications is primarily to find related work. This is done by adapting parts of the established methodologies for literature reviews (Levy/Ellis, 2006; Webster/Watson, 2002). One approach that is continuously used is backward search. This means, that once a relevant publication is found, its references are analyzed to identify the overall set of existing work that was published in the field.

The sources used for the literature reviews are primarily conferences in the field of software performance to ensure that new results are found in a timely manner. The main conferences and high-profile workshops within this field that were closely followed by the author are the following:

- Association for Computing Machinery (ACM) SIGMETRICS Conference [1]
- European Performance Engineering Workshop (EPEW)[2]
- International Conference on Performance Engineering (ICPE)[3]

[1] `http://www.sigmetrics.org/`
[2] `http://www.epew2014.unifi.it/`
[3] `http://icpe.ipd.kit.edu/`

- International Conference on the Quality of Software Architecture (QoSA)[4]
- International Conference on Performance Evaluation Methodologies and Tools (ValueTools)[5]
- International Symposium on Computer Performance, Modeling, Measurements and Evaluation (IFIP Performance)[6]
- International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)[7]
- International Conference on Quantitative Evaluation of SysTems (QEST)[8]

Even though this is not an exhaustive list, the proceedings of these conferences (and their predecessors, e.g., the Workshop on Software Performance (WOSP) and SPEC International Performance Evaluation Workshop (SIPEW) for ICPE) represent main sources of up-to-date information for this work.

Furthermore, journals with close ties to performance research such as Performance Evaluation and the Journal of Systems and Software (JSS) are analyzed. As research results in the area of software performance are also published in traditional software engineering conferences and journals, conferences listed in the online databases of ACM and the Institute of Electrical and Electronics Engineers (IEEE) repositories such as the International Conference on Software Engineering (ICSE), the Automated Software Engineering (ASE) conference or journals such as ACM Transactions on Software Engineering are analyzed. As some work has been published in the IS field on calculating cost for capacity planning purposes or on the charge-back for the provisioning of IT Services based on resource demand data, IS journals such as Business & Information Systems Engineering (BISE) and MIS Quarterly are analyzed.

**Software Prototypes** The research in this work makes extensive use of software prototypes that implement the concepts and approaches presented. The reason for implementing prototypes is to be able to evaluate the developed artifacts. Furthermore, prototypes allow for a frequent iteration from evaluations of their utility to improvements as all changes to the underlying concepts and approaches for these prototypes can be rapidly represented in improved prototype versions.

**Scenarios** In order to evaluate prototypes it is important to set them into context of scenarios in which they achieve the intended utility. As it is practically not feasible to evaluate the same utility function multiple times in different settings, scenario descriptions are used. These descriptions explain how the utility that is evaluated in a more generic way using controlled experiments, can be applied in different scenarios.

**Controlled Experiments** The core of the evaluations in this work are controlled experiments. These experiments are used to evaluate the utility of introduced artifacts in a

[4] http://qosa.ipd.kit.edu/
[5] http://valuetools.org/
[6] http://performance2014.di.unito.it/
[7] http://ieeexplore.ieee.org/servlet/opac?punumber=1000469
[8] http://www.qest.org/

generic setting to make them comparable and reproducible for others. The comparability and reproducibility of these experiments is achieved by using an industry standard EA benchmark (Sim/Easterbrook/Holt, 2003). This benchmark defines an EA, its workload and a dataset. The results are therefore reproducible, as the EA, the dataset and the workload are well defined and are provided by the benchmark. The results are comparable as the benchmark is used in several studies related to performance research and a lot of benchmark results are published on the website of the benchmark provider[9].

**Simulations** Model-based performance predictions can either be performed by using performance models as input for analytical solvers or simulation engines. Simulations as a means to derive performance predictions are therefore a key part of the conducted experiments.

**Informed Arguments** One publication included in this dissertation uses informed arguments to build a convincing case for the concept presented.

## 3.3 Embedded Publications

As outlined in the introduction, part B of this dissertation is composed of seven publications of the author. An overview of all publications that have been (co-)authored during the research work is given in tables 3.2 and 3.3. Both tables include a publication number, the authors, the title and the outlet for each publication. Publications P1 to P7 are included in this work with permission of the corresponding publishers. These publications address the research questions outlined in section 1.2 as follows:

Publications P1 (Brunnert et al., 2014) and P2 (Brunnert et al., 2013) address research question one: *"What are the challenges in applying model-based performance evaluations for enterprise applications?"*

P1 does so by applying literature reviews and informed arguments to outline the challenges of integrating SPE and APM activities with a focus on model-based performance evaluations. P2 supports these arguments by explaining the challenges of integrating model-based performance evaluations into a software development process of an industry project.

Publications P3 (Brunnert/Vögele/Krcmar, 2013) and P4 (Brunnert/Neubig/Krcmar, 2014) address research question two: *"How can enterprise applications built upon the Java Enterprise Edition (EE) standard be automatically represented in performance models?"*

P3 is a solution proposal that outlines an approach to translate measurements of running Java EE applications into performance models. It also evaluates the proposed approach by implementing a prototype and using it in a controlled experiment with a benchmark application. The publication P4 improves the approach presented in P3 and furthermore evaluates its accuracy in up- and downscaling scenarios. These are common scenarios in capacity planning and management activities.

[9] `https://www.spec.org/jEnterprise2010/results/`

| No. | Authors | Title | Outlet |
|---|---|---|---|
| P1 | **Brunnert**, Vögele, Danciu, Pfaff, Mayer, Krcmar | Performance Management Work | Business & Information Systems Engineering (BISE) 2014, Wirtschaftsinformatik (WI)[10] 2014 |
| P2 | **Brunnert**, Danciu, Vögele, Tertilt, Krcmar | Integrating the Palladio-Bench into the Software Development Process of a SOA Project | Symposium on Software Performance (SOSP) 2013 |
| P3 | **Brunnert**, Vögele, Krcmar | Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications | European Workshop on Performance Engineering (EPEW) 2013 |
| P4 | **Brunnert**, Neubig, Krcmar | Evaluating the Prediction Accuracy of Generated Performance Models in Up- and Downscaling Scenarios | SOSP 2014 |
| P5 | **Brunnert**, Wischer, Krcmar | Using Architecture-Level Performance Models as Resource Profiles for Enterprise Applications | International Conference on the Quality of Software Architectures (QoSA)[11] 2014 |
| P6 | **Brunnert**, Krcmar | Detecting Performance Change in Enterprise Application Versions Using Resource Profiles | International Conference on Performance Evaluation Methodologies and Tools (ValueTools) 2014 |
| P7 | **Brunnert**, Krcmar | Continuous Performance Evaluation and Capacity Planning Using Resource Profiles for Enterprise Applications | Journal of Systems and Software (JSS)[12] 2015 |

**Table 3.2:** *Publications embedded in this dissertation*

Publication P5 (Brunnert/Wischer/Krcmar, 2014) addresses research question three: *"How can model-based performance evaluations support capacity planning and management processes for enterprise applications?"*

Building upon the results of P1 to P4, publication P5 proposes the use of architecture-level performance models as resource profiles. Resource profiles are specifically formed models to describe the current state of an EA. Resource profiles can serve as communication medium between parties involved in a capacity planning process. The utility of resource profiles is evaluated using scenarios and a controlled experiment for a migration of an exemplary EA from one hardware environment to another. As capacity planning involves specifying the cost for a deployment and energy cost is one of the major cost drivers nowadays, P5 introduces an approach to not only predict performance but also energy consumption for specific deployment scenarios.

Publication P6 (Brunnert/Krcmar, 2014) addresses research question four: *"How can performance be continuously evaluated during the software development process using model-based techniques while an application is constantly modified?"*

As outlined in the introduction, software development processes have been modified to allow for more frequent releases to quickly adapt to changes in business environments. A common concept to allow for such frequent releases is continuous delivery (Humble/Farley, 2010). A key element of continuous delivery is a deployment pipeline that describes the steps from creating a deployable EA version until an EA version exists that can be released. Publication P6 describes a solution proposal that outlines how resource profiles can be

[10]Ranked A according to WKWI list (WKWI, 2008)
[11]Ranked A according to CORE 2013 conference ranking (CORE, 2013)
[12]Ranked A according to ERA 2010 journal ranking (ERA, 2010)

| No. | Authors | Title | Outlet |
|---|---|---|---|
| P8 | **Brunnert**, Danciu, Krcmar | Towards a Performance Model Management Repository for Component-based Enterprise Applications | International Conference on Performance Engineering (ICPE) 2015 |
| P9 | Danciu, Kroß, **Brunnert**, Willnecker, Vögele, Kapadia, Krcmar | Landscaping Performance Research at the ICPE and its Predecessors: A Systematic Literature Review | ICPE 2015 |
| P10 | Willnecker, **Brunnert**, Gottesheim, Krcmar | Using dynaTrace Monitoring Data for Generating Performance Models of Java EE Applications | ICPE 2015 |
| P11 | Willnecker, Dlugi, **Brunnert**, Spinner, Kounev, Gottesheim, Krcmar | Comparing the Accuracy of Resource Demand Measurement and Estimation Techniques | EPEW 2015 |
| P12 | Vögele, **Brunnert**, Danciu, Tertilt, Krcmar | Using Performance Models to Support Load Testing in a Large SOA Environment | International Workshop on Large-Scale Testing (LT) 2014 |
| P13 | Willnecker, **Brunnert**, Krcmar | Model-based Energy Consumption Prediction for Mobile Applications | Workshop on Energy-Aware Software Engineering and Development (EASED) 2014 |
| P14 | Willnecker, **Brunnert**, Krcmar | Predicting Energy Consumption by Extending the Palladio Component Model | SOSP 2014 |
| P15 | Dlugi, **Brunnert**, Krcmar | Model-based Performance Evaluations in Continuous Delivery Pipelines | International Workshop on Quality-Aware DevOps (QUDOS) 2015 |
| P16 | Danciu, **Brunnert**, Krcmar | Towards Performance Awareness in Java EE Development Environments | SOSP 2014 |
| P17 | Danciu, Chrusciel, **Brunnert**, Krcmar | Performance Awareness in Java EE Development Environments | EPEW 2015 |
| P18 | Jiang, **Brunnert** | LT 2015: The Fourth International Workshop on Large-Scale Testing | ICPE 2015 |
| P19 | Kroß, **Brunnert**, Prehofer, Runkler, Krcmar | Model-based Performance Evaluation of Large-Scale Smart Metering Architectures | LT 2015 |
| P20 | Kroß, **Brunnert**, Prehofer, Runkler, Krcmar | Stream Processing On Demand for Lambda Architectures | EPEW 2015 |

**Table 3.3:** *Further publications during the work on this dissertation*

used within a deployment pipeline to detect performance changes due to feature additions or bug fixes. For this purpose, the resource profile definition of P5 and an extended version of the model generator from publications of P3 and P4 are used. The utility of the performance change detection approach is evaluated in a controlled experiment.

Publication P7 (Brunnert/Krcmar, 2015) builds upon the individual results of publications P3 to P6. It integrates them into a generalized way of supporting the continuous performance evaluation and capacity planning using resource profiles for EAs. It specifically outlines how resource profiles created during development as shown in publication P6 can be distributed with their corresponding EA versions to support capacity planning scenarios as described in publication P5.

In addition to the embedded publications, the author has been involved as (co-)author in several publications related to the topic of this work. An overview of these publications

is given in table 3.3. A brief description of the content of each of these publications and their relationship to this work is given in the following.

The first publication in table 3.3 (P8, Brunnert/Danciu/Krcmar (2015)) is a work-in-progress paper that outlines current work to better support the collaboration of teams within corporations while solving performance problems. It builds upon the SPE and APM integration challenges outlined in P1 and the experiences gained while solving the challenges of applying performance models in an industrial project outlined in P2.

Publication P9 (Danciu et al., 2015b) is an effort to get a better understanding of the existing research in the field of software performance at the ICPE and its predecessors. It contains a systematic literature review that outlines the artifact types, evaluation methods and types of systems that have been used in this field since the inception of the predecessors of the ICPE.

The next publication P10 (Willnecker et al., 2015a) outlines current work with an industry partner that extends the results of the publications P3 and P4 to be able to work with an industrial APM solution instead of a custom monitoring solution used within this work. As not all monitoring solutions provide capabilities to measure resource demands directly, they often have to be estimated based on other metrics. The accuracy of such estimations is evaluated in publication P11 (Willnecker et al., 2015b).

A specific aspect of using model-based performance evaluations in the industrial project outlined in publication P2 is explained in publication P12 (Vögele et al., 2014). It shows how performance models can be used to select performance test cases in service-oriented architectures (SOA).

Publications P13 (Willnecker/Brunnert/Krcmar, 2014a) and P14 (Willnecker/Brunnert/Krcmar, 2014b) build upon the work in publication P5 that introduces an approach to not only predict performance but also energy consumption using PCM. Publications P13 and P14 extend the approach presented in P5 in so far, that not only EAs but also mobile applications are supported.

Publication P15 (Dlugi/Brunnert/Krcmar, 2015) presents the realization of the concepts presented in publication P6 as a tool paper. The next publications P16 (Danciu/Brunnert/Krcmar, 2014), P17 (Danciu et al., 2015a), P18 (Jiang/Brunnert, 2015), P19 (Kroß et al., 2015a) and P20 (Kroß et al., 2015b) are less related to the exact topic of this dissertation but address important aspects of performance engineering. P16 and P17 introduce an approach to support the performance evaluation of Java EE application components during their implementation and before the approaches presented in this work can be applied. P18 represents the proceedings of a workshop co-organized by the author that addresses the challenge of large-scale testing[13]. P19 includes the result of work with an industry partner to use performance models to estimate the capacity for large-scale smart grid systems. P20 builds upon the work in P19 and introduces a concept to reduce the cost of data processing by using performance models to decide when additional processing resources are required to provide a real-time view on an incoming data stream.

[13] `http://lt2015.eecs.yorku.ca/`

# Part B

# Chapter 4

# Performance Management Work

| | |
|---|---|
| Authors | Brunnert, Andreas[1] (brunnert@fortiss.org)<br>Vögele, Christian[1] (voegele@fortiss.org)<br>Danciu, Alexandru[1] (danciu@fortiss.org)<br>Pfaff, Matthias[1] (pfaff@fortiss.org)<br>Mayer, Manuel[2] (mmayer@in.tum.de)<br>Krcmar, Helmut[2] (krcmar@in.tum.de)<br><br>[1]fortiss GmbH, Guerickestraße 25, 80805 München, Germany<br>[2]Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany |
| Outlet | Business & Information Systems Engineering (BISE, 03/2014)<br>Wirtschaftsinformatik (WI, 03/2014) |
| Status | Accepted |
| Contribution of first author | Content and scope definition, construction of conceptual framework, paper writing, paper editing, coordination of reviews |

**Table 4.1:** *Fact sheet publication P1*

## 4.1 Performance Management Work as Continuous Task

Performance is a key quality factor of application systems (AS). AS performance is quantified by the metrics response time, resource utilization and throughput (Becker et al., 2013). To guarantee AS performance, it is important to define quantifiable performance goals using performance metrics. These metrics have to be continuously measured and evaluated. Based on these metrics, activities can be defined to ensure that performance goals are met. The coordination and execution of all activities required to achieve performance goals during system development are described by the term software performance engineering (SPE) (Woodside/Franks/Petriu, 2007). Corresponding activities during operation are typically summarized by the term application performance management (APM) (Menascé, 2002b). An isolated consideration of SPE and APM neglects their interrelation. The combination of SPE and APM activities is therefore summarized by the term performance

management work (PMW). PMW is becoming a growing challenge due to developments in the areas of AS architecture, IT governance and system life cycle.

AS architectures have evolved over time from monolithic to distributed to system of systems architectures (Jamshidi, 2011). The spatial, organizational, cultural and technical diversity of system of systems architectures increases the difficulty of PMW activities (Grabski et al., 2007). As different AS subsystems[1] are associated with different organizations, this architectural style also implies a change from a uni- to a multilateral IT governance thereby making it necessary to coordinate PMW activities across multiple subsystems and organizations.

The subsystem life cycle is driven by functional enhancements and maintenance efforts. A subsystem life cycle is defined by a continuous iteration from system development into operation. Subsystems can be in different life cycle phases at any one point in time. But life cycles need to be synchronized and a key challenge to achieving this synchronization is the pursuit of different goals by development and operation teams. Development teams try to realize new functionalities with high quality requirements as fast as possible. Operation teams, on the other hand, are more interested in keeping their environments in a stable state. The term "DevOps" denotes concepts to better combine and integrate efforts in both life cycle phases (Humble/Molesky, 2011). DevOps concepts can only ensure that performance goals are met if corresponding activities are closely interlinked.

Thus there is necessity to coordinate PMW activities across organizations and life cycles. The current state of the art of PMW activities does not support such global coordination. Whenever new PMW activities or tools are introduced, they are usually concerned with ensuring the performance for certain AS architectures or within specific life cycle phases (Becker et al., 2013). Thus, the business and information systems engineering community should extend the existing research by a process view that supports a comprehensive coordination of PMW activities.

## 4.2 Performance Management Work Activities

PMW activities can be categorized according to the performance goals they support during system development and operation.

### *4.2.1 Performance Management Work During System Development*

During system development, performance goals are to ensure that given non-functional requirements, such as the scalability of an AS architecture, are met. Non-functional performance requirements are often specified by maximum response times for specific transactions. The scalability, in particular, is specified by the flexible adaption of an AS architecture to different user counts and the required throughput. In order to ensure that

[1]For readability reasons, we are referring to AS subsystems as subsystems in the following.

these performance goals are met, different activities are combined to collect the required metrics and to derive and realize optimizations based on these metrics.

Load and performance (L&P) tests are often executed at the end of the system development process. The resulting performance metrics describe an AS in its current state. The representativeness of the collected performance metrics depends on whether or not the test system is comparable to the production environment. Executing representative L&P tests is a huge challenge in practice because the organizational separation of subsystems makes it difficult to access representative instances of the dependent systems.

For detailed performance analysis in the early phases of the system development process, activities such as code analysis, profiling or an instrumentation of the source code are used. The validity of performance metrics collected using these activities is often limited because only subsystems can be analyzed which often have different configurations compared to the target environments.

The activities presented so far are combined in the SPE methodology (Woodside/Franks/Petriu, 2007). Additionally, SPE supports system development by introducing performance models. Performance models can predict the performance of a system based on its software designs. To improve the predictions, these models can be enhanced with performance metrics collected during the system development process. Performance models are not yet in wide spread use in industrial practice (Koziolek, 2010) because the modeling effort currently outweighs its benefits.

### *4.2.2 Performance Management Work During Operation*

The primary performance goal during the operation phase of an AS is to ensure that service-level agreements (SLA) are met. SLAs can be specified by any combination of the performance metrics response time, throughput and resource utilization. Monitoring systems are used to continuously collect these metrics. These systems allow operations staff to get an up-to-date view of the current situation and to evaluate if SLAs are met.

Furthermore, new systems are introduced that automatically analyze performance metrics collected by monitoring systems to reconfigure AS before SLA violations occur. An example for such systems is the dynamic resource allocation in virtualized environments (i.e. cloud infrastructures). The use of such systems can increase the flexibly of organizations while providing new applications and services. Should SLA violations occur, new soft- and/or hardware can be added to the system.

Not all AS architectures can be scaled elastically (Vaquero/Rodero-Merino/Buyya, 2011). Moreover, virtualization cannot guarantee that an AS behaves equally over a period of use. The reason for these behavior differences is the concurrent access of multiple AS to shared IT resources. Therefore, one of the main research directions in the performance field is to explore approaches that improve the dynamic resource allocation. Other important topics in this research area are scalable AS architectures and runtime prediction models (Becker et al., 2013).

Another goal of PMW activities in the operation phase is the coordination and control of continuous changes introduced into production systems. It is essential to evaluate the performance impact of any alternations (i.e. hard- or software changes) before they go into production. Because larger changes are often carried out in separate change projects, all activities mentioned in the system development phase are of relevance.

## 4.3 Future Developments, Capabilities and Application Areas

A look at existing approaches reveals that these individual activities need to be integrated to meet performance goals. If performance is not considered during the system development process, it can also not be guaranteed during operation. Additionally, experience from the operation phase is necessary to make informed performance predictions. This is especially the case in early system development phases. A process-oriented view, which combines all activities required to fulfill performance goals is still missing. The following sections, therefore, present integration options of PMW activities from the AS architecture, IT governance and system life cycle perspectives.

### *4.3.1 Integrating Individual Activities*

To integrate PMW activities from the AS architecture and IT governance perspectives a mapping from subsystems and PMW activities to organizational units is necessary. In the context of cross-organizational IT value-added chains, possibilities need to be investigated to coordinate and integrate PMW activities of different organizations. A basic requirement for such integration is the possibility for a common interchangeability of performance metrics across subsystems (Schmietendorf, 2001). To simplify this exchange, independent methods and tools need to be combined from a technical as well as from an organizational perspective. The results of research in this area are environments and process models for monitoring, analysis and optimization of system of systems architectures.

Integrating the system life cycle and AS architecture perspectives supports PMW activities from the requirements phase to the operation phase. In order to achieve this goal, approaches for designing the transition between life cycle phases need to be identified. Storing and transferring information between different life cycle phases is a considerable challenge; the feedback cycle between these phases should be automated in an effort to address this challenge.

The integration of the IT governance and system life cycle perspectives addresses the organizational framework for PMW activities. It is important to determine which competences are required for this integration in organizations. A new competence profile should be defined that addresses the processes and tools to ensure that performance goals can be met. As performance is a key quality factor of AS, an integration of this competence profile into the European e-Competence Framework should be attempted (EU, 2013). Additionally, an investigation should be undertaken as to how the rights and responsibilities of different organizational units can be represented throughout the system life cycle and how PMW activities can be integrated into the IT service management of an organization.

### *4.3.2 Capabilities and Application Areas*

An increased integration of PMW activities creates new application areas. An example is the description of the resource requirements of AS. Such resource descriptions help to refine accounting models for internal and external IT providers (Brandl/Bichler/Ströbel, 2007). Thus, hardware, energy, licensing and administration costs can be allocated to the organizational units creating these costs. Additionally, transparency of the resource demands helps to reduce these costs in total. Thus, integrated PMW activities support cross-organizational investment and purchasing decisions for complex system of systems architectures. A transparency of performance metrics for different vendors also simplifies the selection of cloud and other service providers.

Overall, a better integration of PMW activities increases the transparency of bottlenecks in the IT value-added chain. As soon as performance metrics are available across organizations, the local optimization of the subsystem performance can be replaced by a global optimization of the AS. Thus, AS planning should be handled cross-organizational as is the case in traditional value-added chains.

Increasing energy costs will further strengthen green IT initiatives. From an energy perspective the current focus is on increasing the efficiency of hardware and cooling. However, because the software running on the hardware influences the IT resource and the resulting energy demand, a stronger focus on the energy efficiency of software is inevitable and needs to be integrated into the acceptance process. The transparency of performance metrics and an increased integration of PMW activities therefore contributes to a reduction of the energy demand in data centers. Thus, PMW ensures the environmental friendliness of AS and prepares the IT for its way into a more efficient future.

# Chapter 5

# Integrating the Palladio-Bench into the Software Development Process of a SOA Project

| | |
|---|---|
| Authors | Brunnert, Andreas[1] (brunnert@fortiss.org)<br>Danciu, Alexandru[1] (danciu@fortiss.org)<br>Vögele, Christian[1] (voegele@fortiss.org)<br>Tertilt, Daniel[1] (tertilt@fortiss.org)<br>Krcmar, Helmut[2] (krcmar@in.tum.de)<br><br>[1]fortiss GmbH, Guerickestraße 25, 80805 München, Germany<br>[2]Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany |
| Outlet | Symposium on Software Performance (SOSP, 2013) |
| Status | Accepted |
| Contribution of first author | Scope definition, project planning and management, development of prototypes, paper writing, paper editing |

**Table 5.1:** *Fact sheet publication P2*

**Abstract** This paper presents how the performance modeling capabilities of the Palladio-Bench are integrated into the development process of new enterprise applications based on a service-oriented architecture (SOA). The Palladio-Bench is used to predict the performance of applications early in the software development process. To better integrate the Palladio-Bench into this process, an automated transformation of existing software models into Palladio Component Models (PCM) is implemented. These software models contain the business processes represented in the new applications and implementation details such as web services used within the processes. The performance of the modeled applications is mainly influenced by the response times of the web services. Therefore, the web service response time behavior is modeled using software performance curves, which are automatically generated using monitoring data collected during software tests or in the production environment. Several integration tools are developed to support this feedback loop between the different phases of a software life cycle. Besides these integration capabilities, the challenges of using PCM within this project are discussed and future enhancements for the Palladio-Bench itself are proposed.

## 5.1 Introduction

The overall goal of the industrial project that we describe in this paper is to transform an existing IT landscape into a service-oriented architecture (SOA). The SOA paradigm describes how loosely coupled software components offer services in a distributed environment. SOA enables the integration of legacy applications and aims at increasing the flexibility of enterprises. However, one of the main concerns when implementing a SOA is the expected performance of the overall system (O'Brien/Brebner/Gray, 2008; Liu/Gorton/Zhu, 2007). Legacy systems found in enterprises are often not designed for this type of interaction. New access patterns and additional software layers therefore lead to different performance characteristics.

Our goal in this project context is to develop a continuous performance management process that allows to integrate performance evaluations early into the software development process. One of the key tools that we use for this purpose is Palladio-Bench (Becker/ Koziolek/Reussner, 2009). This paper describes how we apply the Palladio-Bench within this project and our experiences in terms of the effort required for its introduction in an existing software development process.

The remainder of this paper is organized as follows: Section 5.2 describes the context of the industrial project. Afterwards, Palladio-Bench use cases and enhancements are explained in section 5.3. Limitations of using Palladio-Bench in the project context and proposed feature enhancements are also outlined in the same section followed by a conclusion and directions for future work in section 5.4.

## 5.2 Project Context

This section explains the project context by describing the transition of the current IT landscape into a SOA. Afterwards, our approach to support this transition by introducing a continuous performance management process is illustrated.

### *5.2.1 Transition to a Service-Oriented Architecture*

The current IT landscape of our project partner consists of several legacy systems which support business processes isolated from each other. Users performing business operations interact directly with each system by using different user interfaces. This leads to a high redundancy of data as well as to a high overhead for the users. The reason for this overhead is that similar data needs to be managed using different systems which often leads to inconsistencies. Thus, to better support the employees, our partner develops an integrated user interface as single point of entry for all business processes. To support this kind of integration on a user interface layer, the data and processes that are available in the existing legacy applications need to be integrated into the new front-end. For this purpose, a Java Enterprise Edition (EE)-based middleware integration layer is used in their infrastructure. The existing functionalities of the legacy systems are provided using SOAP

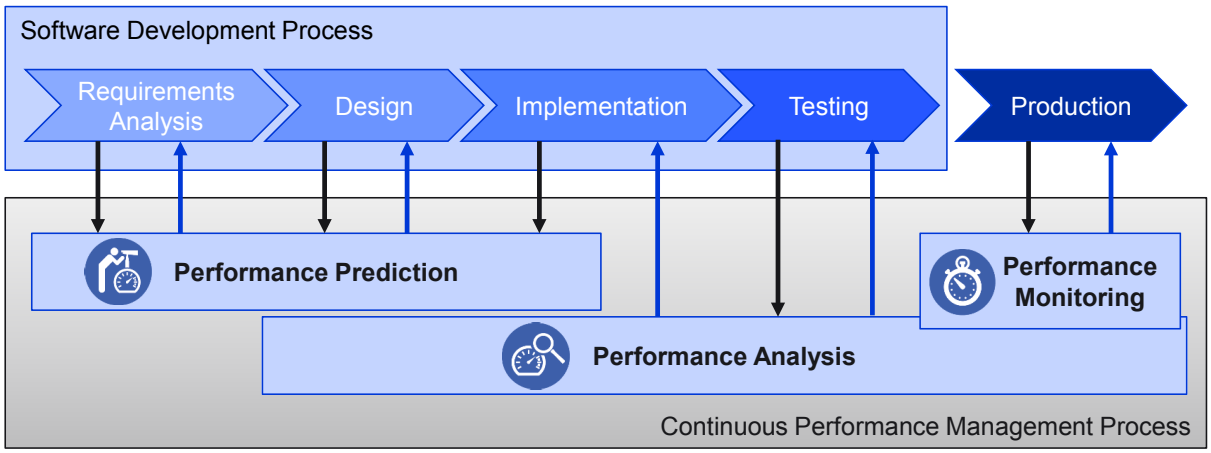



**Figure 5.1:** *Continuous performance management process*

web services running on Java EE servers as facades in front of the existing systems. This approach allows to provide these services consistently from a service consumer perspective and to abstract the technology differences of the legacy systems. The single services provided by the legacy systems are all connected to an enterprise service bus (ESB) to avoid point-to-point connections between service providers and consumers.

The integrated user interface is realized using several JavaServer Faces (JSF)-based web applications that access the web services over the ESB. These applications are created in the software development process using a model-driven development approach. Business logic is modeled as event-driven process chains (EPC) by field specialists during the requirements analysis phase. These EPC models are then transformed to Unified Modeling Language (UML) models. At the same time, the UML models are extended with technical information such as interactions with web services in the design phase. Afterwards, these UML models are transformed into executable components that can be run as JSF web applications on a Java EE server.

As multiple of these JSF-based web applications need to be developed to support all business processes, several software development projects are necessary to implement the vision of an integrated user front-end. These projects need to carefully evaluate the performance of each new web application, as users of these new front-ends expect similar performance as what they are used to get from their isolated legacy systems.

### *5.2.2 Continuous Performance Management Process*

The continuous performance management process that we intend to develop supports the development as well as the production phase of new applications. As the performance needs to be investigated early in the software life cycle, our primary project goal is to support the performance evaluation in the software development process.

As shown in figure 5.1, the performance management process consists of three phases: performance prediction, performance analysis and performance monitoring. Each of these phases consists of activities which are required to ensure that given performance

requirements can be met. The sum of these activities describe the work that a performance analyst needs to perform in an organization.

Performance prediction includes all activities to support performance evaluations early in the development process. During this phase, performance models are derived from UML models created in the design phase. These performance models are parameterized using information about the response times of external systems and the expected user behavior. Using these models as input for a simulation engine allows to evaluate the system performance for different workloads.

Performance analysis includes activities that can be performed once running versions of the software applications are available. The performance characteristics of applications are evaluated in this phase using profiling and load testing. The results of these activities are used to optimize the architecture of the applications and to improve test designs.

The performance monitoring phase supports all activities to ensure the compliance with defined service-level agreements (SLA) and to gather data about the response time behavior of the web services in the SOA environment. The performance management process is designed in a way that performance data gathered using performance measurement and monitoring tools in the test and production phases can be used in the performance prediction and analysis phases. This feedback cycle improves the accuracy of the performance predictions early in the process as existing knowledge (i.e. response time measurements of the SOAP web services) can be reused.

### *5.2.3 Performance Management Tool Chain*

To support this continuous performance management process, an integrated performance management tool chain (see figure 5.2) is being developed. This tool chain consists of several tools (such as the Palladio-Bench) to provide an integrated support for the performance management work. Currently, a number of these tools are already available in the organization but they only support isolated performance management activities.

For example, several load test, profiling and monitoring tools are already in use. However, these tools are not integrated and the data gathered using such tools is not accessible in all project phases or projects within the organization. Thus, the integration of these tools in order to automate the performance management activities is the main focus of our development efforts.

During the test and production phases, performance data (i.e. response times) is collected from running applications and services. This data is aggregated in a performance database and can therefore be used in other phases of the software life cycle to support the performance evaluation.

Apart from making existing measurements consistently available to different users in an organization, this data is used to parameterize performance models (see section 5.2.2). These performance models conform to the Palladio Component Model (PCM) meta-model

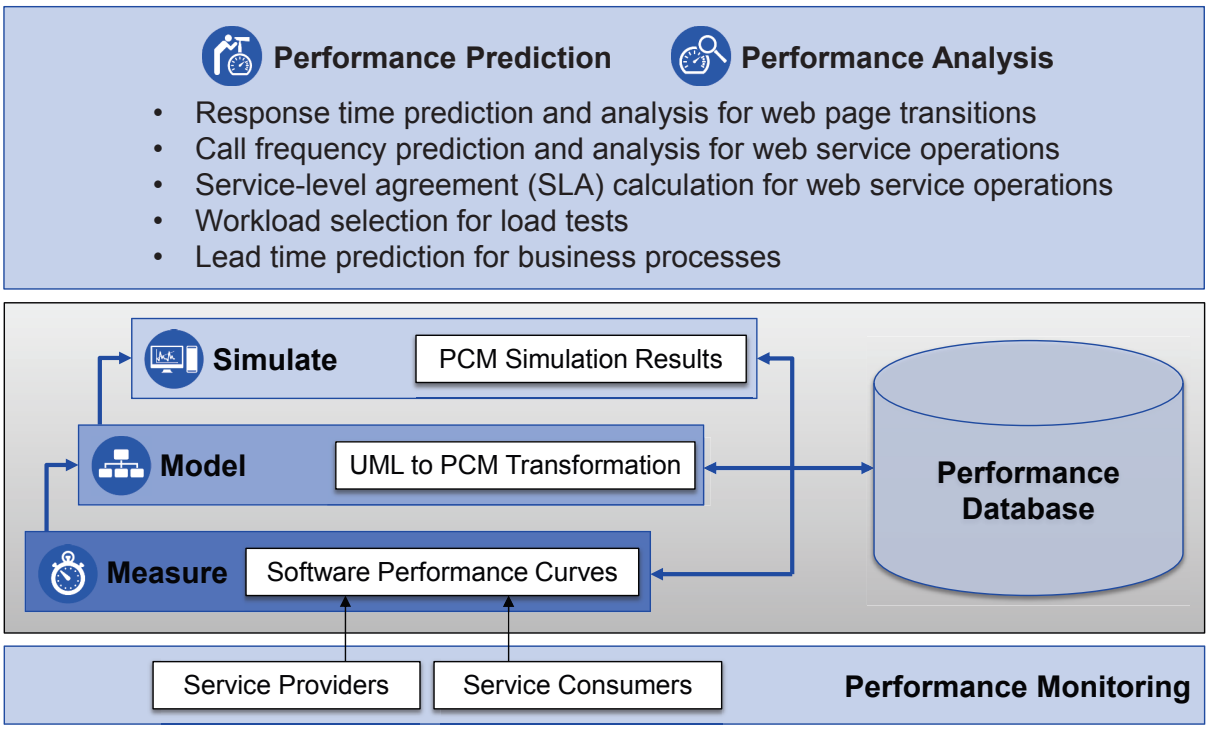



**Figure 5.2:** *Performance management tool chain (Brunnert* et al.*, 2012)*

and are used within the Palladio-Bench. The remainder of this paper focuses on the specific use cases for the Palladio-Bench within this tool chain.

## 5.3 Palladio-Bench Integration

The Palladio-Bench is used during different phases of the performance management process. One of the main reasons for choosing the Palladio-Bench over other performance modeling tools is the comprehensibility of the performance models. PCM models are easily comprehensible by technical staff in an organization due to the UML alignment and the different views on the system. PCM models also allow to store performance related information that often will not be documented otherwise (i.e. response times or resource demands for specific service operations). The following sections describe use cases and enhancements for the Palladio-Bench within our project context.

### *5.3.1 Performance Model Generation*

To support the performance evaluation early in the development process, PCM models are automatically generated based on UML models in the design phase. A business process is represented by several UML activity diagrams that can reference each other. Thus, specific UML activity diagrams are reused by different business processes. The UML activity diagrams are accessed in a model repository through a Java application programming interface (API) to extract the required information for generating PCM

models. Performance models can be generated for a set of business processes selected by the user.

The UML activity diagrams contain usage behavior information like usage probabilities and think times as well as the main control flow for the application logic. The usage probabilities and think times are collected through interviews with domain experts. Thus, realistic workload scenarios can be used for the performance evaluation of the new enterprise applications (Menascé, 2002a).

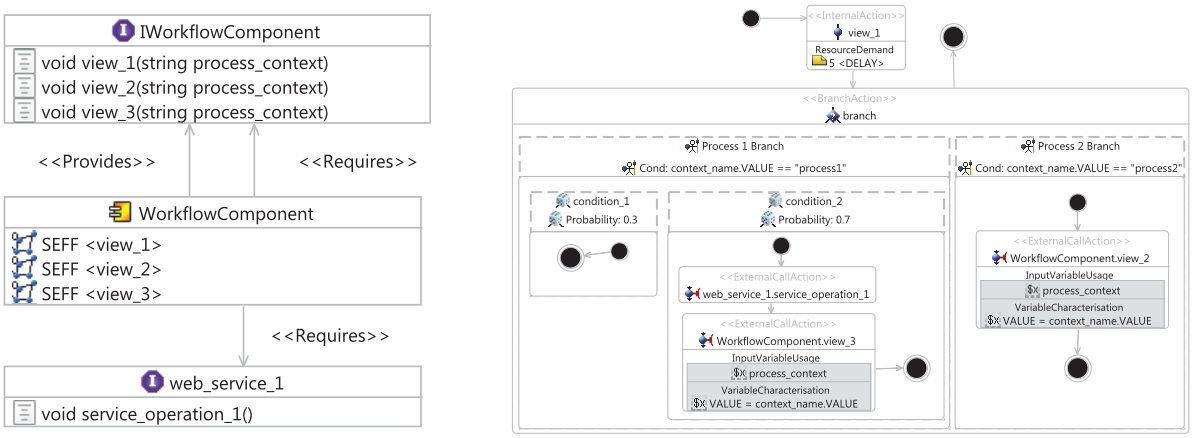
PCM specifies several model layers for different views on a system (Becker/Koziolek/ Reussner, 2009). Two of the main model layers are the PCM repository and usage models. The PCM usage model specifies the workload on a system and the repository model contains the system components, their behavior and relationships. PCM usage and repository models are generated based on the information in the UML models. The other model layers, namely the PCM resource environment, system and allocation models are generated based on static information. The reason for this decision is that these models specify which hardware resources (i.e. servers and central processing unit (CPU) cores) exist and how the repository model components are mapped to these resources. As the available hardware resources are not changing very often, this information is provided as configuration to the model generation code. In the following, the generation of PCM repository models is explained in detail.

As a first step, all view-to-view transitions are extracted from the UML activity diagrams. A view is defined as a web page presented to users of an application. For each view, all possible successor views are extracted and the execution flows between the current view and its successors are represented in Resource Demanding Service Effect Specifications (RDSEFF). RDSEFFs are component behavior descriptions similar to UML activity diagrams. The generated RDSEFFs are all associated with a so called *WorkflowComponent* in the PCM repository model (see *view_1, view_2 and view_3* in figure 5.3(a)).

Each successor view is dependent on the context (business process) in which a specific view is used. Therefore, a context parameter called *process_context* is used in the corresponding RDSEFFs to differentiate the view-to-view transition behavior for the different business processes. This is shown in figure 5.3(b), the generated RDSEFF for *view_1* receives a *process_context* parameter and uses this parameter to differentiate its behavior according to the business process in which it is used. In this example, the view is used in *process1* and *process2* and behaves differently depending on the process context.

Once all view-to-view transitions are extracted from the UML activity diagrams and their RSDSEFFs are available in a repository model, the single RDSEFFs are connected with each other. This means, that at the end of a view transition in a RSDSEFF, another RDSEFF is called for the new view which then specifies the execution flows for the next possible view transitions. In the example in figure 5.3(b), *view_1* calls the RDSEFF of *view_2* when it is called in the context of business process two and the RDSEFF of *view_3* if it is called in the context of business process one.

These interconnected view-to-view transitions specified in the single RDSEFFs, model the complete execution flow of a business process. The usage behavior for different business processes can thus be easily modeled by calling the first view of the process and specifying




(a) Generated repository model example (b) Generated RDSEFF for *WorkflowComponent.view_1*

**Figure 5.3:** *Generated PCM repository model elements*

the current process name in the *process_context* parameter. This parameter is then used in all RDSEFFs that are called within the execution flow, as it is passed on when another view RDSEFF is called (see figure 5.3(b)).

During the transitions between the different views, calls to external web services are represented in the UML activity diagrams. Thus, each RDSEFF does not only contain views, think times as well as the probabilities for specific execution flows but also references to external web services (see figure 5.3(b)). For each web service found in the UML models, a new PCM component interface is created in the PCM repository model (i.e. *web_service_1* in figure 5.3(a)). How the behavior of these service interfaces is represented in PCM models is explained in the next section.

### *5.3.2 External System Representation*

Once the PCM model generation based on UML activity diagrams is completed, the behavior of external systems is added to these models. For this purpose, SOAP web services provided by existing legacy systems are modeled as black box components using the response time behavior of their operations for different request counts.

To represent this response time behavior, our Palladio-Bench instances are extended with the performance curve integration (PCI) plugin (Wert/Happe/Westermann, 2012). The PCI plugin allows to specify the behavior of PCM repository interface operations (i.e. *web_service_1.operation_1* in figure 5.3(a)) without modeling a RDSEFF. Instead, the response time of an operation can be specified as a function with multiple input parameters. In our scenario, we only use the amount of parallel users of a component as input for these functions to calculate the response time.

In order to use performance curves within the current version of the Palladio-Bench, the PCI plugin which is currently available for version 3.2 is adapted to version 3.4.1. In addition to the migration of the PCI plugin we have enhanced its functionality. The original version of the PCI plugin is only able to work with so called data tables that map

response times to input parameters, e.g. to the number of concurrent requests. In our new version, the response time behavior can also be specified using formulas. To derive these formulas linear and polynomial regression approaches are used to associate the number of concurrent requests with corresponding response times observed during the performance monitoring or test phases.

### *5.3.3 Palladio-Bench Use Cases*

Using the automatically generated PCM models enhanced with usage probabilities and software performance curves allows us to support different performance management tasks. A set of examples can be found in figure 5.2.

First of all, the expected response time for page transitions in JSF web applications can be analyzed by simulating the corresponding PCM models. Even though these models do not contain any resource demands of the UI layer, the response time behavior of web service operations used during page transitions allows to estimate response times for the user. Additionally, the call frequency for specific web service operations can be analyzed using the PCM models. This helps to support service providers in their sizing process as each development project can communicate their estimated workload more precisely. The information on page transition times and call frequencies can also be used to derive SLAs for the maximum response times of specific web service operations.

Apart from analyzing and specifying response time requirements, the performance simulations can support the selection of load test cases. The simulation results show how often a specific usage path is performed by the end users and how many web services are involved in this process. Both information pieces can help the test team to derive load test scripts.

Additionally, data on usage probabilities, think times and the web service response time behavior can be used to predict the overall process lead times of specific business processes. This information may be used to redesign business processes if potential improvement areas are detected.

### *5.3.4 Limitations and Proposed Feature Enhancements*

During our application of the Palladio-Bench and while implementing the enhancements explained in the previous section, we have faced some challenges for which we would like to suggest some future enhancement ideas.

First of all, the usage model editor should offer the possibility to reference usage models from usage models. In practice, many usage behaviors are reused in different scenarios or have to be factored out for complexity reduction. Currently, these usage behaviors have to be modeled redundantly.

Another useful enhancement would be a better visual representation of nested branches as they become very complex when large business processes are modeled. Furthermore,

a capability to stop complete execution flows on certain conditions would simplify the modeling process. This would also reduce the model complexity as termination conditions don't need to be modeled multiple times.

The representation of memory within PCM models would also be very beneficial. JSF applications require a considerable amount of Java heap memory. As the heap demand grows with each additional user on the system, the heap size is an important factor to estimate the required capacity for the production environment. Especially, as the garbage collection overhead in each Java EE server instance grows with the amount of heap that needs to be managed.

Additional feature enhancements should include an improved visualization of simulation results. Simulation results contain large sets of data, which are hard to examine. A preprocessing of simulation results to support users by using color schemes or visual indicators for noticeable problems such as potential bottlenecks would be very helpful. A better migration support for existing models between different Palladio-Bench versions is another desirable feature enhancement.

## 5.4 Conclusion and Future Work

As shown in this paper, the Palladio-Bench can greatly benefit the performance evaluation in different phases of the software development process. The features implemented in our part of this project help to make the Palladio tooling better applicable in practice as creating PCM models by hand is not feasible in most industry projects.

As this project is ongoing, most of the proposed approaches are still under development and need to be evaluated. A key challenge for future enhancements of the PCM-related features will be to simplify the use of the performance modeling and simulation capabilities. This is especially important to allow non-performance modeling experts to use the performance management tool chain proposed in this work.

To include the resource demands in the performance evaluation of the new enterprise applications, the performance model generation capabilities introduced in (Brosig/Huber/Kounev, 2011; Brunnert/Vögele/Krcmar, 2013) might be applied in the context of this project.

# Chapter 6

# Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications

| | |
|---|---|
| Authors | Brunnert, Andreas[1] (brunnert@fortiss.org)<br>Vögele, Christian[1] (voegele@fortiss.org)<br>Krcmar, Helmut[2] (krcmar@in.tum.de)<br><br>[1]fortiss GmbH, Guerickestraße 25, 80805 München, Germany<br>[2]Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany |
| Outlet | European Performance Engineering Workshop (EPEW, 2013) |
| Status | Accepted |
| Contribution of first author | Problem and scope definition, development of the conceptual approach, prototype development, experiment design, execution and result analysis, paper writing, paper editing |

**Table 6.1:** *Fact sheet publication P3*

**Abstract** The effort required to create performance models for enterprise applications is often out of proportion compared to their benefits. This work aims to reduce this effort by introducing an approach to automatically generate component-based performance models for running Java EE applications. The approach is applicable for all Java EE server products as it relies on standardized component types and interfaces to gather the required data for modeling an application. The feasibility of the approach and the accuracy of the generated performance models are evaluated in a case study using a SPECjEnterprise2010 industry standard benchmark deployment. Simulations based on a generated performance model of this reference deployment show a prediction error of 1 to 20 % for response time and of less than 10 % for CPU utilization and throughput.

## 6.1 Introduction

Performance modeling of software systems has been a research topic for several decades (Balsamo et al., 2004). Even though the flexibility of a performance evaluation using performance models would be beneficial for many industry software projects (Woodside/ Franks/Petriu, 2007; Smith, 2007), they are not in widespread use as of today (Koziolek, 2010; Mayer et al., 2011). One of the key challenges is the effort required to create representative performance models that often outweighs their benefits (Kounev, 2005).

Many recently introduced performance modeling approaches focus on the evaluation of component-based software systems such as modern enterprise applications (Koziolek, 2010). The performance evaluation of such enterprise applications is especially challenging as they are used by several hundred or thousand users concurrently with varying workloads. Component-based performance modeling languages (Koziolek, 2010) have already simplified the modeling process because component-based enterprise applications can be represented in these models using the same components they are composed of (Wu/Woodside, 2004). Additionally, different aspects that influence the performance of a component (such as the deployment platform or the usage profile) can be modeled separately. This is a huge step forward regarding the applicability in practice compared to performance models using abstract notations such as layered queuing networks or queuing petri nets (Balsamo et al., 2004). Tools emerged for these component-based performance modeling languages that help to make performance modeling a lot more accessible to practitioners (Reussner et al., 2007). Unfortunately, there are still some challenging questions left during the model creation process that need to be answered by researchers and practitioners alike:

1. Which components should be represented?
2. Which component relationships should be represented?
3. What data needs to be collected to parametrize a performance model?
4. How can the required data be collected?
5. How can the required data be processed and added to a performance model?

Answering these questions requires a lot of experience in the software engineering as well as in the performance modeling process. General guidelines to answer these questions for different software development domains would therefore help to simplify the modeling process. The automated performance model generation approach proposed in this work answers these questions for Java Enterprise Edition (EE) applications. The Java EE specification (Shannon, 2006) defines the component types an application needs to be composed of and a runtime environment for hosting Java EE applications that is consistently available across Java EE server products. Therefore, the suggested performance model generation approach is designed in a way that it can be applied for Java EE applications running on all Java EE server products that are compliant with the specification.

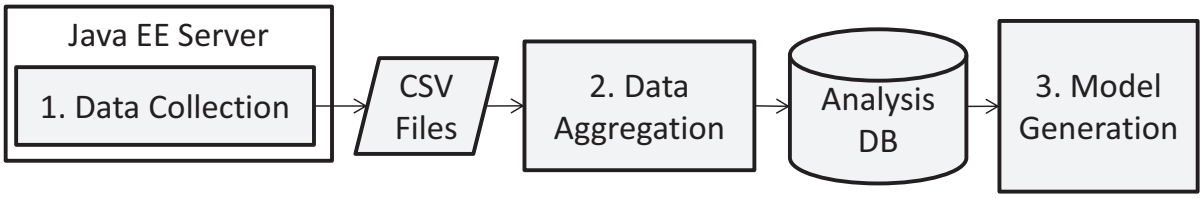



**Figure 6.1:** *Performance model generation process*

## 6.2 Automatic Performance Model Generation

The performance model generation is executed in three different steps which are shown in figure 6.1: First of all, the data to create a performance model is collected from a running Java EE application (1.); afterwards the data is preprocessed (2.) to aggregate the required information. Finally, the aggregated data is used to generate a component-based performance model (3.). These three steps are explained below.

### *6.2.1 Data Collection*

One of the main challenges when representing enterprise applications in performance models is choosing an appropriate level of detail for the model elements. This decision directly influences the data required to create a performance model. Wu/Woodside (2004) suggest that software systems that are assembled by predefined components should be represented using the same components in a performance model. Following this suggestion, the approach presented in this work uses component types defined in the Java EE specification to construct a performance model. The Java EE specification (Shannon, 2006) defines the main application component types as Applets, Application Clients, Enterprise JavaBeans (EJB) and web components (i.e. Servlets, JavaServer Pages (JSP)). Complex Java EE applications are typically composed of a combination of such application component types. As Applets and Application Clients are executed outside of a Java EE server runtime, the remainder of this paper focuses on EJBs and web components. Using this level of detail for modeling Java EE applications comes along with the advantage that users of such performance models can easily map their findings to real application components and thus solve performance issues more easily. Furthermore, interfaces defined in the Java EE specification can be used to collect the required performance model parameters automatically. To parametrize a component-based performance model that contains all EJB and web components of a Java EE application as model elements, the following data needs to be collected:

1. EJB and web component names
2. EJB and web component operation names accessible to other components
3. EJB and web component relationships on the level of component operations
4. Resource demands for all EJB and web component operations

The data collection is described first for Servlets and JSPs. For these component types the Java EE specification defines Servlet filters that are always invoked before and after a Servlet or JSP is called (Shannon, 2006). Each request-response cycle of a Servlet or JSP invocation is assigned to exactly one thread at a time. This enables a very fine-grained data collection for all web components of an application.

The basic logic of the Servlet filter for collecting the required data can be found in listing 6.1. The *doFilter* method of the *PerformanceMonitoringFilter* is invoked whenever a Servlet or JSP is called. Before forwarding the current request to the Servlet or JSP (using *chain.doFilter*) the resource demand that the current thread has consumed so far is stored in a temporary variable (*startRD*). Once the request processing is completed, the updated resource demand for the current thread is stored in the *stopRD* variable. By subtracting the corresponding stop and start values, the resource demand for the current request can be calculated afterwards. As of today, the *PerformanceMonitoringFilter* can be configured to collect the central processing unit (CPU) demand in nanoseconds (ns) and the allocated bytes in the heap for the current thread. By default, only the CPU demand is collected. The *storeDemand* method stores the resource demands of the current invocation in multiple comma-separated value (CSV) files (one file per thread) for further analysis. Additionally, the Servlet path is stored as the component name and a configurable request parameter that is passed to the JSP or Servlet is stored as operation name for the current invocation.

The CPU demand in nanoseconds (ns) for the current thread is collected using the *getCurrentThreadCpuTime()* method provided by the *java.lang.management.ThreadMXBean* of the Java Virtual Machine (JVM). An approximation of the bytes allocated in the Java heap by the current thread can be acquired by using the *getThreadAllocatedBytes()* method of the *com.sun.management.ThreadMXBean*. It is important to note, that even though the returned values of the *getCurrentThreadCpuTime()* method are of nanosecond precision, the accuracy of this method varies on different operating systems (Kuperberg, 2010). Typical Windows operating systems provide an accuracy of 10 milliseconds (ms), whereas some UNIX based operating systems provide an accuracy of 1 ns (Kuperberg, 2010). In consequence, if a high accuracy is required, this measurement approach is only feasible on these systems.

**Listing 6.1:** *Basic Servlet filter logic*

```
public class PerformanceMonitoringFilter implements Filter {
 public void doFilter(req, res, chain){
  ResourceDemand startRD = getCurrentThreadResourceDemand();
  chain.doFilter(req, res);
  ResourceDemand stopRD = getCurrentThreadResourceDemand();
  storeDemand(startResourceDemand, stopResourceDemand);
 }
}
```

Similar to a Servlet filter, an EJB interceptor can be applied to filter all calls to specific or all EJBs of an application. Such an EJB interceptor is used to gather the resource demand of single method invocations for different EJBs in a system (DeMichiel/Keith, 2006). The basic logic of the EJB interceptor is similar to the one of the Servlet filter: an *intercept* method is called for each invocation of an EJB method and stores the resource

demands of the request processing thread before and after the invocation of the EJB method. Afterwards, the EJB interceptor also stores the data in multiple CSV files (one file per thread) for further analysis. The EJB class name is used as the component name and the called EJB method name passed to the *intercept* method as component operation name.

Multiple nested Servlet filter and EJB interceptor invocations can occur within one request-response cycle. It is therefore important to mention that the resource demand measurements of the first filters (or interceptors) in the chain already contain the resource demand measurements of the filters and interceptors that get called afterwards. To differentiate these demands afterwards, a request in the system needs to be uniquely identifiable. For that purpose, the filter or interceptor that gets called first, generates a unique transaction identifier (ID). The transaction ID is used in subsequent filter or interceptor invocations in the same thread to identify a request. Using this ID allows to track the resource usage of different components for single transactions in the system. In addition to this transaction ID, the Servlet filter and the EJB interceptor track the call-stack depth. The call-stack depth defines the amount of filters and interceptors that are nested within one request, to recreate the execution flow during the data analysis. Using an additional call-order attribute for each operation invocation during one thread execution, the different invocations can be ordered according to their execution sequence. Therefore, each entry of the CSV files contains the following information: transaction ID; call-order; call-stack depth; Servlet, JSP or EJB name; EJB method name or the selected Servlet/JSP request parameter; startCPUTime; stopCPUTime; startHeapByteAllocation; stopHeapByteAllocation; time stamp of the data collection. It is therefore possible to extract the application components, their relationships as well as their resource demand from this data.

The file operations are optimized by using different caches and output files for each thread to reduce the Servlet filter and the EJB interceptor impact on the monitored system. To assess the influence of the monitoring code on the measurement results, test runs to collect the CPU demand for a reference system are performed twice. In the first run only the CPU demands for components that started transactions (without any sub-measurements) are collected. In the second run the same values are measured while all Servlet filters or EJB interceptors are active. Comparing the results shows that each Servlet filter or EJB interceptor invocation results in a mean CPU demand overhead of 0.03 ms. This overhead is always included in the CPU demand measurements of the calling component.

### *6.2.2 Data Aggregation*

The CSV files that are generated during the data collection phase are used as the input for a data aggregation component. The purpose of this component is to pre-process the data to reduce the model generation time afterwards.

All the pre-processed data is stored in a relational database (DB) called *analysis DB* to allow for a more flexible access to the collected data. As explained in the previous section, each entry in the CSV files represents a single component invocation. Therefore, the first step in the data aggregation process is to extract the existing components of

the Java EE application from the CSV files. The component names found in this data are stored in a specific table in the database. At the same time, the existing component operations (EJB methods or Servlet/JSP request parameters) are extracted from the data. Component operations are stored in a separate database table. While searching for the existing components and their operations, the component invocations are stored in another database table. The component invocations are associated with the component names and operations they belong to. Additionally, the data aggregation component associates all component invocations with the transactions they are involved in based on the transaction ID in the log files. This data model simplifies the model generation process as component relationships can be easily identified on the level of single component operations.

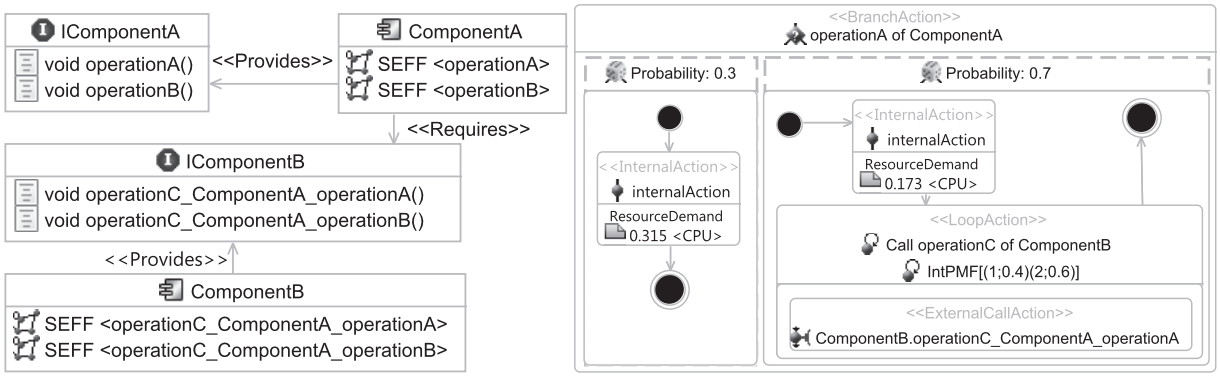
### *6.2.3 Model Generation*

The data stored in the *analysis DB* is used to generate a component-based performance model based on the available information. The Palladio Component Model (PCM) is used as the meta-model for the generated models. PCM is described by Reussner et al. (2007) as a software component model for business information systems to enable model-driven quality of service (QoS, i.e. performance) predictions. A software system is represented in PCM by several model types which can reference each other (Reussner et al., 2007). The most important model within PCM is called repository model. This model contains the basic components of a software system and their relationships. These components are assembled in a system model to represent an application. The user interactions with the system are described in a usage model. The other two models in PCM are the resource environment and allocation models. The purpose of the resource environment model is to specify available resource containers (i.e. servers) with their associated hardware resources (i.e. CPU cores). The allocation model specifies the mapping of components to resource containers. To take these different model types into account, the model generation process is divided in three sub-tasks:

1. PCM repository model generation
2. Associating resource demands with the PCM repository model components
3. Generating the system, resource environment and allocation models

#### 6.2.3.1 PCM Repository Model Generation

First of all, a PCM repository model is generated to represent the Java EE application components and their relationships. The component relationships are represented in a repository model using interfaces. Each component can *provide* an interface to offer operations to other components and *require* interfaces from other components to access their operations (see figure 6.2(a)). As the component relationships are stored in the *analysis DB* on the level of single component operations, the generated component interfaces contain all externally accessible operations of a component. Internal component operations are not represented in the model.




(a) Repository model example (b) RDSEFF of ComponentA.operationA

**Figure 6.2:** *PCM repository model elements*

The behavior of the component operations is specified in so called Resource Demanding Service Effect Specifications (RDSEFF). RDSEFFs are behavior descriptions of single component operations similar to activity diagrams in the Unified Modeling Language (UML) (Krogmann, 2012). Input- or output-parameters of the component operations are not represented to simplify the model and because they are not available in the *analysis DB*. However, as these parameters can have great influence on the component performance, the probabilities of different execution flows (caused by different input- or output-parameters) are represented in the individual RDSEFFs. These probabilities are represented as so called probability branches in each RDSEFF (Reussner et al., 2007). An example for such a probability branch can be found in figure 6.2(b). The RDSEFF of the *operationA* of *ComponentA* contains a probability branch with two execution flows. One execution flow is executed with 30 % probability whereas the second execution flow is executed with 70 % probability.

To calculate the probability of each execution flow, the transactions in the *analysis DB* are first of all grouped by the component operation that was invoked first (i.e. by the user or external systems). In a second step, the execution flows that have been started by a specific component operation are grouped by the order in which they call external component operations. The total amount of transactions started by a specific component operation and the amount of transactions for each external component operation call order allow to calculate the probability of each execution flow. The second grouping does not consider the amount of times an external component operation was called in a row. To account for such invocation count variations, these values are represented as loop probabilities in the corresponding execution flows. An example for such a loop probability can be found in figure 6.2(b): In the execution flow with 70 % probability, the external *operationC* of *ComponentB* is called one time in 40 % of the cases and two times in the other 60 %.

A component operation can be included in execution flows that have been started by different component operations. To simplify the RDSEFFs, a new interface operation and a corresponding RDSEFF is created for each execution flow a component operation is involved in. The naming pattern for the new operation is as follows: *[operation name]-_[component initially called]_[operation name initially called]*. For example, in figure 6.2(a), the *operationC* of *ComponentB* is called in an execution flow started from *operationA* and also in a flow started from *operationB* of *ComponentA*. The model generation code

therefore generates two services for this operation: *operationC_ComponentA_operationA* and *operationC_ComponentA_operationB*.

#### 6.2.3.2 Associating Resource Demands

The resource demand of a component operation is also stored in the corresponding RDSEFF. As representing memory is not directly supported by the PCM meta-model without further extensions, the model generation currently only uses the CPU demand logged in the data collection step. As explained in section 6.2.1, the CPU demand of a component invocation already contains the CPU demands of all sub-invocations. Therefore, each transaction is processed in reverse order to calculate and remove the CPU demand of all external calls from the current operation. As the external calls vary between each execution flow, the CPU demand values are calculated separately for each of the flows in a probability branch. The mean CPU demand in ms for each execution flow is then assigned to the component operation. In the example in figure 6.2(b), one execution flow of *operationA* consumes 0.315 ms CPU time whereas the other flow consumes 0.173 ms.

#### 6.2.3.3 Generating the Remaining PCM Models

When the repository model generation is completed, a system model is generated to represent a Java EE application that contains all repository model components. The automatic model generation specifies all interfaces that are not required by other components as externally accessible interfaces of the system. Thus, the model generation assumes that end-users or external systems interacting with a Java EE application have invoked the operations contained in these interfaces. Component operations that are not contained in these interfaces are not accessible outside of the generated system (i.e. not accessible from the usage model).

The resource environment and the allocation models are also generated automatically. The resource environment model is generated to define a reference system for the CPU demand values specified in the RDSEFFs. This is necessary to ensure that the CPU demand values are interpreted as ms values. The allocation model maps the system model components to the resource container in the resource environment model. Currently, only one server (resource container) with a single CPU core is generated in the resource environment and all components are mapped to this resource container. The resource environment model should therefore be configured according to the required setup (i.e. number of CPU cores) before it is used for a performance evaluation. The only PCM model that is not automatically generated is the usage model.

## 6.3 Evaluation

To evaluate the feasibility of the suggested performance model generation approach, a performance model for a SPECjEnterprise2010[1] industry standard benchmark deployment is generated using a software prototype that implements the approach (SPEC, 2012). SPECjEnterprise2010 specifies a Java EE application, a workload as well as a dataset that needs to be used for a performance test execution. The tests are therefore easily reproducible. Using the SPECjEnterprise2010 applications and predefined workloads performance tests are executed and a performance model is derived. Afterwards, workloads using varying amounts of users are executed both as a simulation using the performance model and on a real system to compare the results. This quantitative validation ensures that the automatically generated performance model provides a solid base for the performance evaluation.

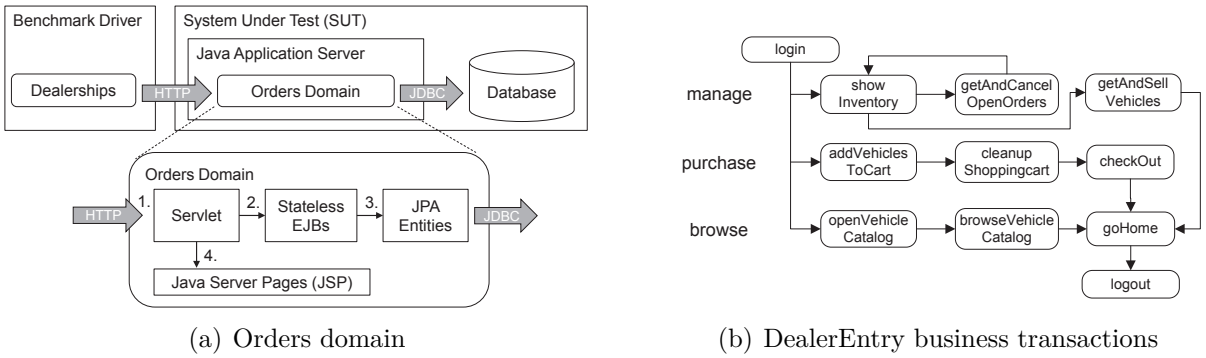
### *6.3.1 SPECjEnterprise2010 Industry Benchmark Deployment*

The business case for the SPECjEnterprise2010 application is a scenario that incorporates Supply Chain Management (SCM), Manufacturing and Customer Relationship Management (CRM) for an automobile manufacturer (SPEC, 2012). Following these key functional areas, the application is divided into three major parts: the Supplier domain for the SCM, the Manufacturing domain and the Orders domain for the CRM. The analysis in this paper focuses on the Orders domain. The communication between the domains is also not examined in this paper. The SPECjEnterprise2010 Orders domain is used as reference application for the case study because it is fully compliant to the Java EE specification and therefore portable across different application server products. This portability is a rare characteristic for such a complex Java EE application that already uses a lot of common technologies within the Java EE technology stack. Furthermore, the SPECjEnterprise2010 Orders domain is slightly different compared to the other domains as it represents a complete application intended to be used by end-users. The other two domains are mainly intended to be used by other applications as (web-)services.

#### 6.3.1.1 Application Architecture

The Orders domain is a Java EE web application implemented using Servlets and JSPs (Shannon, 2006) as key technologies. Apart from these technologies the Orders domain uses stateless EJBs and the Java Persistence API (JPA) (Shannon, 2006) to implement an E-Commerce catalog and a shopping cart for automobile dealers (SPEC, 2012). The setup for the Orders domain consists of a System Under Test (SUT) on which the Orders domain application components are deployed and a Benchmark Driver used for generating load on the system (see figure 6.3(a)). A relational DB is the persistence layer of the

[1]SPECjEnterprise is a trademark of the Standard Performance Evaluation Corp. (SPEC). The SPECjEnterprise2010 results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The official web site for SPECjEnterprise2010 is located at http://www.spec.org/osg/Enterprise2010.




(a) Orders domain (b) DealerEntry business transactions

**Figure 6.3:** *SPECjEnterprise2010*

Orders domain application. The automobile dealers access the web application over the Hypertext Transfer Protocol (HTTP) using their web browsers to order and sell cars. The execution flow of the Orders domain application can be described in four steps: Each HTTP request of the automobile dealers is processed by a Servlet (1.) which determines the type of request and executes the business logic while calling one or more stateless EJBs (2.). Afterwards, the request is forwarded to a JSP to render the view (4.). The stateless EJBs interact with JPA Entities (3.) which represent the application model that is persisted in a relational database.

#### 6.3.1.2 System Topology

The SUT is deployed on a virtualized hardware environment exclusively used for the SPECjEnterprise2010 benchmarks performed for this paper. Two IBM System X3755M3 servers with 256 gigabyte (GB) random-access memory (RAM) and four AMD Opteron 6172 Processors with 12 cores and a 2.1 GHz frequency each are virtualized using the VMWare ESXi 5.0.0 (build 469512) hypervisor. The SUT is represented by a single virtual machine (VM) with four virtual CPU cores, 40 GB RAM and openSUSE 12.3 64 bit as operating system. The runtime for the Orders domain application components is a JBoss Application Server (AS) 7.1.1 in the Java EE 6.0 full profile and an Apache Derby DB in version 10.9.1.0 as persistence layer. The JBoss AS and the Apache Derby DB are executed within a 64 bit Java OpenJDK Server VM in version 1.7.0 (IcedTea7 2.3.8, build 23.7-b01). The database is therefore included in the server JVM. A different virtual machine is used for the Benchmark Driver. SPECjEnterprise2010 uses the Faban Harness and Benchmark Framework (Faban, 2012) to generate load on the SUT. To avoid influences on the system performance by the load generation, the Benchmark Driver VM is placed on a different hardware server than the SUT VM. Both servers are connected by a gigabit ethernet connection. The workloads that are generated by the Benchmark Driver will be explained in the next section.

#### 6.3.1.3 Workload Description

The automobile dealers can perform three different business transactions: browse, manage and purchase. The dealers interact with the Orders domain application by browsing

the catalog of available cars (browse), purchasing new cars (purchase) and managing their inventory by selling vehicles or cancel orders (manage). The main steps for each of these transactions are shown in figure 6.3(b). These transactions are implemented in the SPECjEnterprise2010 benchmark by the DealerEntry application that executes the corresponding transactions on the SUT (SPEC, 2012). This application specifies the probability for each transaction and its single steps during a single DealerEntry execution. Each transaction starts with a login of an automobile dealer, whose identity is randomly selected. While the automobile dealer is logged in, the user can perform multiple browse, purchase and manage operations with a specific probability. After a specified time interval the user logs out of the application.

### *6.3.2 Automatic Performance Model Generation*

A moderate load (~55 % CPU utilization) is generated on the SUT to gather the required data for the model generation using the data collection approach outlined earlier. The load is generated for 30 minutes while only data collected between a five minute ramp up and a five minute ramp down phase is stored in the *analysis DB*. As the database is included within the server JVM, these measurements already contain the database CPU demands. Afterwards, a software prototype that implements the performance model generation process is used to generate a component-based performance model based on this data. The generated PCM repository model of the Orders domain application is shown in figure 6.4. Following the application architecture, the generated model contains a controller Servlet component (*app*), several EJB components (*CustomerSession*, *ItemBrowserSession*, *LargeOrderSenderSession* and *OrderSession*) and different JSP components to render the view (*dealerinventory.jsp*, *error.jsp*, *order.jsp*, *purchase.jsp*, *shoppingcart.jsp*, *SpecJhome.jsp* and *welcome.jsp*). The main entry point for the user is the *app* Servlet that dispatches the calls to the other components. Which component operations are called by the different *app* Servlet operations can be seen in the generated operation names of the other components.

In the generated resource environment model the CPU core count is set to four according to the SUT configuration. The repository-, system- and allocation models that are generated automatically are not changed manually. The usage model is modeled manually following the DealerEntry application source code.

### *6.3.3 Measurement and Simulation Results in Comparison*

PCM models can be taken as the input for a simulation engine to predict and evaluate the application performance for different workloads or resource environments. The standard simulation engine for PCM models is SimuCom which uses model-2-text transformations to translate PCM models into Java code, that is compiled and executed to start a simulation (Becker, 2008). To evaluate the accuracy of the model introduced in the previous section this simulation engine is used to predict the application performance from low (~40 % CPU utilization) to high load conditions (~90 % CPU utilization). The simulation results are compared with measurement data of the real system under the same load conditions. To generate low load on the system, the different tests start with 500 parallel dealer clients

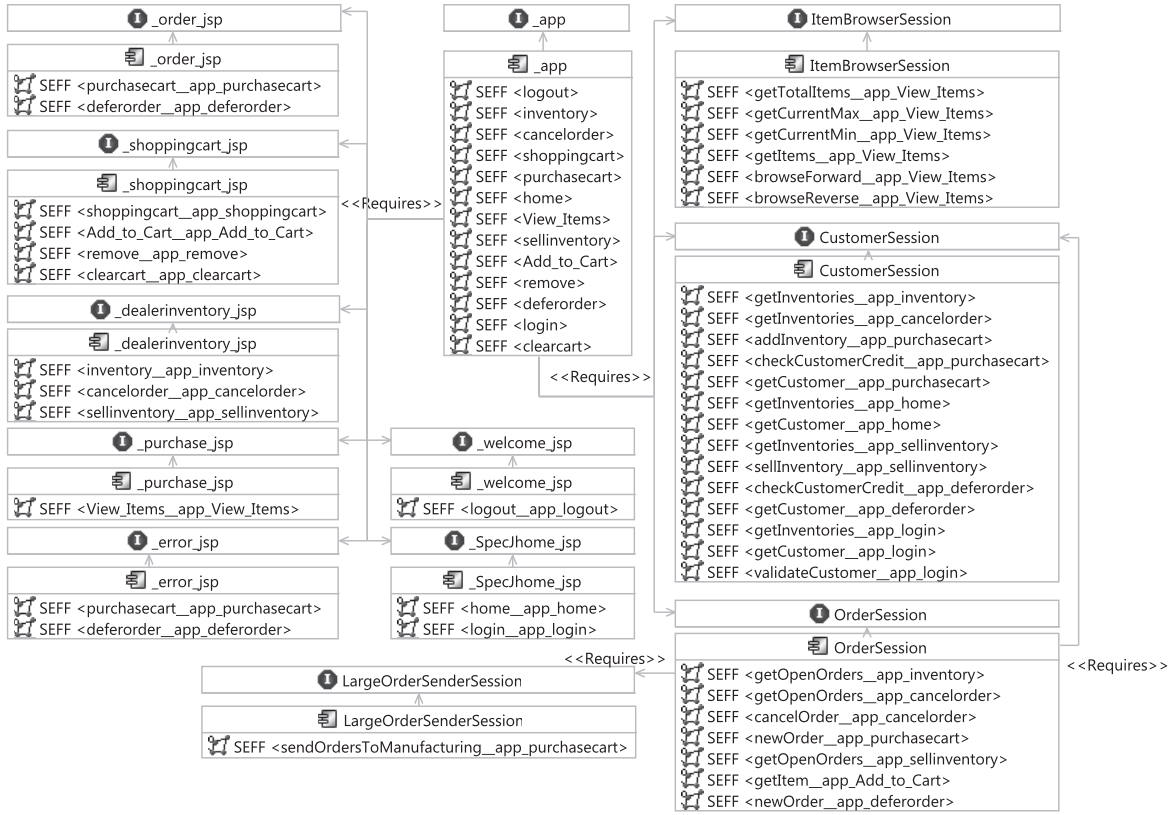



**Figure 6.4:** *Simplified performance model of the Orders domain application*

and then gradually increase in steps of 100 until high load conditions with 1100 parallel dealer clients are reached.

The comparison includes the throughput and response time of the browse, manage and purchase transactions as well as the CPU utilization of the SUT. The instrumentation to gather the CPU demand for the different application components is removed from the real system for the tests. Each simulation and test run on the real system lasted 30 minutes. To avoid influences of warm up effects and varying user counts, only data between a five minute ramp up and a five minute ramp down phase is included in the following comparison. The measured and simulated values within this 20 minute phase are used for the calculation of the mean response time and mean CPU utilization values without further filtering. During the test the CPU demand of the real system is collected using the system activity reporter (SAR). The throughput data for the different business transactions is taken from the reports generated by the Faban harness for each test run. Even though the Faban harness also reports response times for the business transactions, they cannot be compared with the simulation results as the network overhead between the Benchmark Driver VM and the SUT VM is not represented in the automatically generated performance model. Therefore, the response time values for the real system are gathered using a Servlet filter by logging the response times for each of the operations processed by the controller Servlet of the Orders domain application. The mean response times for the different Servlet operations are used to calculate the business transaction response time according to their distribution in the business transactions of the DealerEntry application. This approach enables the comparison of the simulated business transaction response times with those of the real system.

| C | T | MMRT | SMRT | RTE | MT | ST | TE | MCPU | SCPU | CPUE |
|---|---|---|---|---|---|---|---|---|---|---|
| 500 | B | 52.08 ms | 57.49 ms | 10.38 % | 30,065 | 30,343 | 0.92 % | 43.71 % | 39.65 % | 9.28 % |
| | M | 12.27 ms | 13.54 ms | 10.34 % | 15,081 | 15,099 | 0.12 % | | | |
| | P | 22.18 ms | 23.76 ms | 7.13 % | 15,105 | 14,991 | 0.75 % | | | |
| 600 | B | 52.94 ms | 57.76 ms | 9.10 % | 36,325 | 36,349 | 0.07 % | 51.93 % | 47.50 % | 8.52 % |
| | M | 12.41 ms | 13.62 ms | 9.75 % | 18,085 | 18,093 | 0.04 % | | | |
| | P | 22.28 ms | 24.06 ms | 7.97 % | 18,223 | 18,100 | 0.67 % | | | |
| 700 | B | 56.10 ms | 60.49 ms | 7.83 % | 42,262 | 42,496 | 0.55 % | 60.47 % | 55.40 % | 8.38 % |
| | M | 12.42 ms | 14.24 ms | 14.70 % | 21,381 | 21,194 | 0.87 % | | | |
| | P | 23.18 ms | 24.97 ms | 7.72 % | 21,131 | 20,923 | 0.98 % | | | |
| 800 | B | 59.55 ms | 64.38 ms | 8.11 % | 48,623 | 48,243 | 0.78 % | 68.78 % | 63.11 % | 8.25 % |
| | M | 13.15 ms | 15.21 ms | 15.64 % | 24,532 | 24,227 | 1.24 % | | | |
| | P | 24.42 ms | 26.67 ms | 9.21 % | 24,159 | 24,149 | 0.04 % | | | |
| 900 | B | 65.43 ms | 65.74 ms | 0.48 % | 54,231 | 54,350 | 0.22 % | 75.67 % | 71.04 % | 6.11 % |
| | M | 14.08 ms | 15.53 ms | 10.32 % | 27,487 | 27,171 | 1.15 % | | | |
| | P | 26.33 ms | 27.28 ms | 3.60 % | 26,752 | 27,085 | 1.24 % | | | |
| 1000 | B | 84.02 ms | 80.54 ms | 4.14 % | 60,658 | 60,312 | 0.57 % | 83.70 % | 78.88 % | 5.76 % |
| | M | 16.20 ms | 18.97 ms | 17.12 % | 30,231 | 30,203 | 0.09 % | | | |
| | P | 32.57 ms | 33.53 ms | 2.95 % | 29,938 | 30,049 | 0.37 % | | | |
| 1100 | B | 140.81 ms | 113.02 ms | 19.73 % | 66,563 | 66,364 | 0.30 % | 90.94 % | 86.61 % | 4.76 % |
| | M | 22.32 ms | 26.69 ms | 19.57 % | 33,269 | 33,146 | 0.37 % | | | |
| | P | 51.90 ms | 47.13 ms | 9.20 % | 33,384 | 32,820 | 1.69 % | | | |

**Table 6.2:** *Measured and simulated results*

The measured and simulated results are shown in table 6.2. For each load condition specified by the number of clients (C) the table contains the following data per business transaction (T): Measured Mean Response Time (MMRT), Simulated Mean Response Time (SMRT), Response Time Error (RTE), Measured Throughput (MT), Simulated Throughput (ST), Throughput Error (TE), Measured (MCPU) and Simulated (SCPU) Mean CPU Utilization and the CPU Utilization Error (CPUE). The simulation predicts the mean response time of the business transactions with an error of mostly 7 to 17 %. Only the browse and manage transactions have a prediction error of 20 % for a load of 1100 concurrent dealer clients. As the performance model is solely based on the components CPU demand, external effects like input/output operations are one possible reason for the deviation in high load conditions. The throughput of the system is predicted with an error below 2 %. This high accuracy can be explained by the fact that the average think time of 9.9 s between all dealer client requests is much higher than the average execution time of a business transaction, which is at maximum 140.81 ms (see table 6.2). Therefore, prediction errors of the response times have a low impact on the predicted number of executed transactions. A prediction error of less than 10 % is achieved for the CPU utilization. The simulated mean CPU utilization is constantly below the measured mean CPU utilization. This is expected, as the simulated data represents the CPU utilization of the JBoss AS process whereas the measured data represents the CPU utilization of the whole system. Thus, the measured CPU utilization of the SUT also contains other processes running on the operating system. Additionally, the CPU demand of the garbage collector and other JVM activities that are not executed within the request processing threads is not included in the model.

## 6.4 Related Work

Several approaches to evaluate the performance of Java EE (or similar component-based) applications using performance models have already been discussed by Kounev (2005). Extending the previous work, Kounev (Kounev/Buchmann, 2003) shows how to apply model-based performance analysis to large-scale Java EE applications. Using the SPECjAppServer2002 (SPEC, 2002) industrial benchmark, Kounev analyzes the applicability of analytical performance models for Java EE applications with realistic complexity. The author extends his work in (Kounev, 2005; Kounev, 2006) by using Queuing Petri Nets (Bause, 1993) to evaluate the performance of a SPECjAppServer2004 industrial benchmark (SPEC, 2002) deployment. In these works, Kounev models the system manually as a number of server nodes without detailing single components of the application or differentiating between single applications running within a server. Therefore, the models can evaluate the performance of the whole system but do not provide sufficient detail to evaluate an application or its components.

Brosig/Huber/Kounev (2011) show that they are able to semi-automatically extract Palladio Component Models (PCM) for Java EE applications using a SPECjEnterprise2010 deployment as case study. The authors define methods for an automatic identification of connections between single runtime components based on monitoring data of a WebLogic Application Server. Unfortunately, the identification of an appropriate granularity level for modeling the components is still left to the user. Their approach also requires a manual calculation and distribution of the CPU demands to the application components based on the overall utilization and throughput of the system.

## 6.5 Conclusion and Future Work

This paper introduced an approach to generate component-based performance models for Java EE applications automatically. The approach is applicable for Java EE applications running on server products that are compliant with the Java EE specification. Using the approach does not require detailed knowledge about the application architecture as the performance model components are based on component types defined in the Java EE specification. It is also not required to have detailed knowledge about the performance modeling process as the generation process already answers the questions raised in the beginning. These characteristics reduce the effort required to create performance models and thus make them better applicable in practice.

Future work for this approach includes case studies for other applications and Java EE server products. Additionally, the approach needs to be extended to work with distributed systems. A key requirement for this extension is the possibility to uniquely identify transactions across multiple server instances. Especially if the approach should work with Java EE components typically used as back-ends such as web-services or message driven beans. Other external systems should be represented using black box approaches to reduce the need to collect data outside of Java EE runtime environments. Representing the heap demand in the generated models is another challenge that needs to be addressed.

# Chapter 7

# Evaluating the Prediction Accuracy of Generated Performance Models in Up- and Downscaling Scenarios

| | |
|---|---|
| Authors | Brunnert, Andreas[1] (brunnert@fortiss.org)<br>Neubig, Stefan[1] (neubig@fortiss.org)<br>Krcmar, Helmut[2] (krcmar@in.tum.de)<br><br>[1]fortiss GmbH, Guerickestraße 25, 80805 München, Germany<br>[2]Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany |
| Outlet | Symposium on Software Performance (SOSP, 2014) |
| Status | Accepted |
| Contribution of first author | Scenario and scope definition, prototype development, experiment design and result analysis, paper writing, paper editing |

**Table 7.1:** *Fact sheet publication P4*

**Abstract** This paper evaluates an improved performance model generation approach for Java Enterprise Edition (EE) applications. Performance models are generated for a Java EE application deployment and are used as input for a simulation engine to predict performance (i.e., response time, throughput, resource utilization) in up- and downscaling scenarios. Performance is predicted for increased and reduced numbers of CPU cores as well as for different workload scenarios. Simulation results are compared with measurements for corresponding scenarios using average values and measures of dispersion to evaluate the prediction accuracy of the models. The results show that these models predict mean response time, CPU utilization and throughput in all scenarios with a relative error of mostly below 20 %.

## 7.1 Introduction

Numerous performance modeling approaches have been proposed to evaluate the performance (i.e., response time, throughput, resource utilization) of enterprise applications

(Balsamo et al., 2004; Koziolek, 2010; Brunnert/Wischer/Krcmar, 2014). These models can be used as input for analytical solvers and simulation engines to predict performance. Performance models are especially useful when scenarios need to be evaluated that cannot be tested on a real system. Scaling a system up or down in terms of the available hardware resources (e.g., number of CPU cores) are examples for such scenarios.

Evaluating the impact of up- or downscaling on performance is a typical activity during the capacity planning and management processes. Capacity planning concerns questions such as "How many hardware resources are required for the expected workload of new enterprise application deployments?" and involves evaluating the behavior of an application when a system is scaled up. Capacity management on the other hand is usually concerned with evaluating whether the existing hardware resources are sufficient for the current or expected load. This involves not only upscaling but also downscaling scenarios in which the amount of hardware resources needs to be reduced to save costs (e.g., license fees that depend on the number of CPU cores used).

Nowadays, creating a performance model requires considerable manual effort (Brunnert et al., 2014). This effort leads to low adoption rates of performance models in practice (Koziolek, 2010). To address this challenge for Java Enterprise Edition (EE) applications, we have proposed an automatic performance model generation approach in Brunnert/ Vögele/Krcmar (2013). This work improves the existing approach by further reducing the effort and time for the model generation.

In order to evaluate whether the automatically generated performance models are fit for use during capacity planning and management, we evaluate the improved model generation approach in up- and downscaling scenarios. In a first step, an automatically generated performance model is used to predict the performance of a system in an upscaling scenario, in which additional CPU cores are added to the system. Afterwards, a downscaling scenario is evaluated in which the number of CPUs is reduced. During the evaluation of the up- and downscaling scenarios not only the number of CPU cores is modified, but also the amount of users interacting with the system simultaneously.

## 7.2 Generating Performance Models

This section is based on our previous work on generating performance models for Java EE applications (Brunnert/Vögele/Krcmar, 2013). In this work, we are using the same concepts for the model generation but reduce the time required for generating a performance model to mostly less than a minute. To make this work self-contained, a brief overview of the model generation process is given, changes are described in more detail. The model generation process is divided into a data collection and a model generation step, the explanation follows these two steps.

### *7.2.1 Data Collection*

The data that needs to be collected to create a representative performance model is dependent on which components should be represented in the model (Brunnert/Vögele/Krcmar, 2013). Following Wu/Woodside (2004), Java EE applications are represented using the component types they are composed of. The main Java EE application component types are Applets, Application Clients, Enterprise JavaBeans (EJB) and web components (i.e., JavaServer Pages (JSP) and Servlets) (Shannon, 2006). As Applets and Application Clients are external processes that are not running within a Java EE server runtime, the remainder of this paper focuses on EJB and web components. To model a Java EE application based on these component types, the following data needs to be collected (Brunnert/Vögele/Krcmar, 2013):

1. EJB and web component as well as operation names
2. EJB and web component relationships on the level of component operations
3. Resource demands for all EJB and web component operations

In Brunnert/Vögele/Krcmar (2013) we have collected this information using dynamic analysis, saved it in comma-separated value (CSV) files and used an additional process step to aggregate this information into a database. To speed up the model generation process we are no longer using files as persistence layer and have removed the additional step of aggregating the data stored in the files in a relational database. Instead, the data required for modeling an application is collected and aggregated in Managed Beans (MBeans) (Microsystems, 2006) of the Java Virtual Machine (JVM). MBeans are managed Java objects in a JVM controlled by an MBean server.

The reason for choosing MBeans as persistence layer is that the Java Management Extension (JMX) specification defines them as the standard mechanism to monitor and manage JVMs (Microsystems, 2006). The JMX and related specifications also define ways to manage, access and control such MBeans locally as well as from remote clients. For example, the JMX remote application programming interface (API) allows access to all MBeans of a system remotely using different network protocols. Building upon the JMX standard therefore ensures that the approach is applicable for all products that are compliant with the Java EE (Shannon, 2006) and JMX (Microsystems, 2006) specifications.

One of the key challenges for the transition from CSV files to MBeans is to find a data model with low impact on an instrumented system. The instrumentation for the dynamic analysis collects structural and behavioral information as well as resource demands for each component operation invocation. As storing the data for each invocation separately in an MBean is not possible due to the high memory consumption, the data needs to be aggregated. Additionally, recreation of existing control flows from the data needs to be possible. To accompany these requirements and to implement the MBean data collection with low impact on the monitored system, the data model shown in figure 7.1 is used.

A *JavaEEComponentOperationMBean* is registered for each externally accessible component operation. Internal component operations are not represented in the data model.

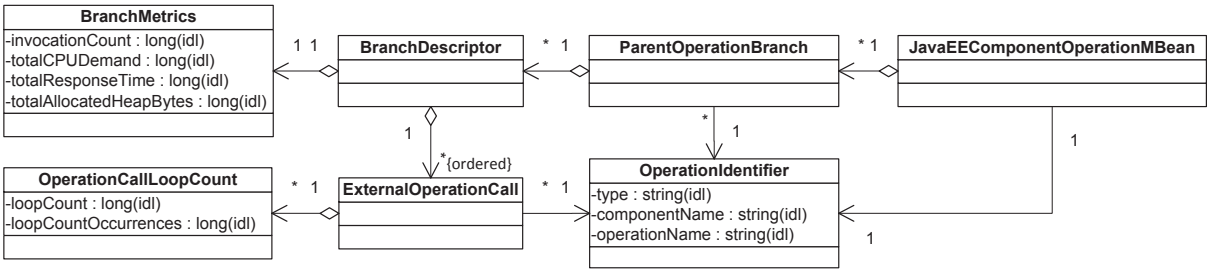



**Figure 7.1:** *JavaEEComponentOperationMBean data model*

Each *JavaEEComponentOperationMBean* instance is identified by an *OperationIdentifier* attribute. The *OperationIdentifier* is a unique identifier of a component operation in a Java EE runtime. It therefore contains the respective *componentName* (i.e., EJB or web component name), the component *type* (i.e., Servlet/JSP/EJB) and its *operationName* (i.e., Servlet/JSP request parameter or EJB method name).

The component relationships are also stored in the data model. These relationships differ depending on component states as well as input and output parameters of their operations (i.e., whether an external operation is called or not). To simplify the data model, component states and parameters of component operations are not represented. Instead, the invocation counts for different control flows of a component operation are stored in the model.

A control flow of a component operation is represented by the *BranchDescriptor* class and its ordered list of *ExternalOperationCalls*. *ExternalOperationCalls* are identified using an *OperationIdentifier* attribute. Using this attribute, *ExternalOperationCalls* can be linked to the corresponding *JavaEEComponentOperationMBeans*. This link allows recreating the complete control flows of requests processed by applications in a Java EE runtime.

*ExternalOperationCalls* have an additional *OperationCallLoopCount* attribute, which is used to track the number of times an external operation is called in a row. This attribute helps to limit the amount of data that needs to be stored in the MBeans, as repeating invocations do not need to be stored separately. Instead, each *loopCount* that may occur in an otherwise equal control flow can be tracked using the same data structure. For each *loopCount*, the *OperationCallLoopCount* class stores the number of times a *loopCount* occurred in the *loopCountOccurrences* attribute.

A *BranchDescriptor* also contains a *BranchMetrics* attribute, which tracks the number of times a control flow occurred (*invocationCount*) and how much CPU, heap and response time is consumed by the current component operation in this control flow in total. This information allows calculating the control flow probability and its resource demand during the model generation.

To differentiate requests processed by applications in a Java EE runtime, control flows of an operation are grouped according to the component operation that is invoked first during a request (i.e., by users or external systems). This grouping is specified in the *ParentOperationBranch* class. It maps a list of *BranchDescriptors* to a *JavaEEComponent-OperationMBean* and contains a reference to the *OperationIdentifier* of the first operation. This reference improves the data collection and model generation performance.

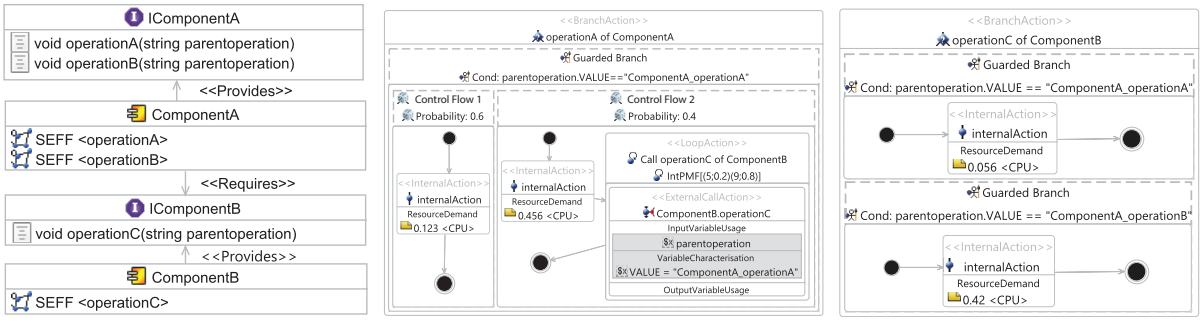
### *7.2.2 Performance Model Generation*

The data stored in the MBean data model (see figure 7.1) is used to generate component-based performance models. The meta-model for the generated models is the Palladio Component Model (PCM) (Becker/Koziolek/Reussner, 2009). PCM consists of several model layers that are all required to use PCM for performance predictions (Becker/Koziolek/Reussner, 2009). This section describes how the repository model layer can be generated automatically as the other model layers can only be created using the information contained in this model. The PCM repository model contains the components of a system, their operation behavior and resource demands as well as their relationships. Repository model components are assembled in a system model to represent an application. User interactions with the system are described in a usage model. The other two model layers in PCM are the resource environment and allocation model. The purpose of a resource environment model is to specify available resource containers (i.e., servers) with their associated hardware resources (CPU or HDD). An allocation model specifies the mapping of components to resource containers. To simplify the use of generated repository models, default models for the other PCM model layers are generated automatically once the repository model generation is complete (Brunnert/Vögele/Krcmar, 2013). The PCM usage model is not generated automatically and has to be modeled manually.

Following the data model in figure 7.1, *JavaEEComponentOperationMBean* instances in a Java EE runtime are used to generate a repository model. The model generation is implemented as a Java-based client that accesses the MBean data using the JMX Remote API. The list of available *JavaEEComponentOperationMBeans* is first of all used to generate component elements in the repository model (e.g., *ComponentA* and *ComponentB* in figure 7.2(a)). The component list can be derived from the data model by filtering the available *JavaEEComponentOperationMBeans* by the *componentName* attribute of the associated *OperationIdentifier*. Operations provided by Java EE components are also available in the same data structure and are generated in the same step. In a PCM repository model, operations provided by a component are specified in an interface (e.g., *IComponentA* and *IComponentB* in figure 7.2(a)).

Afterwards, the data in the list of *ExternalOperationCalls* for each *BranchDescriptor* of a *JavaEEComponentOperationMBean* is used to represent component relationships. These relationships are specified in a repository model by a *requires* relationship between the repository component that calls a specific component operation and the interface that provides this operation (e.g., *ComponentA requires IComponentB* in figure 7.2(a)). The model generation can therefore use information about external operations called in specific operation control flows (using the *OperationIdentifier* of the *ExternalOperationCalls*) to create the relationships of repository model components.

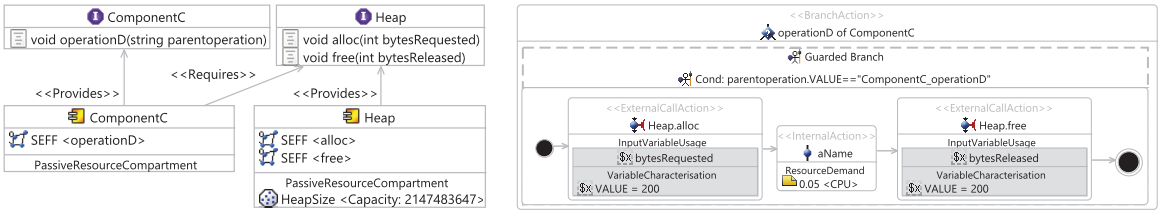
So far, only components, interfaces and their relationships are available in the repository model. In the next step, the behavior of component operations needs to be specified. The component operation behavior is specified in Resource Demanding Service Effect Specifications (RDSEFF). RDSEFFs are behavior descriptions similar to activity diagrams in the Unified Modeling Language (UML).




(a) Repository model example (Brunnert/Vögele/Krcmar, 2013)

(b) RDSEFF of ComponentA.operationA

(c) RDSEFF of ComponentB.operationC (Brunnert/Vögele/Krcmar, 2013)




(d) Heap component

(e) Calling Heap.alloc and Heap.free from RDSEFFs

**Figure 7.2:** *PCM repository model elements*

As explained in the data collection section 7.2.1, a component operation can be included in different requests processed by a Java EE runtime (see *ParentOperationBranch* in figure 7.1). To represent the resulting behavior differences in a performance model, a *parentoperation* parameter is passed between component operations. An example can be found in figure 7.2(b): *operationA* of *ComponentA* is the first operation called during a request, it thus specifies the *parentoperation* as *ComponentA_operationA* in the external call to *ComponentB.operationC*. This parameter is used in the RDSEFF of *operationC* of *ComponentB* to differentiate the operation behavior depending on the parameter value (see figure 7.2(c)). The initial *parentoperation* parameter value that is set when a request starts is passed on to all sub-invocations. For example, *ComponentA_operationA* would be passed on if *ComponentB.operationC* would call another external operation within this request. The behavior description of *ComponentA.operationA* in figure 7.2(b) is also contained in a guarded branch with the condition that the current operation needs to be *ComponentA_operationA*. This is necessary to ensure that all component operations can be used equally in the PCM repository and usage model layers. Thus, operations that only start requests and those that are also (or only) used within requests are indistinguishable.

A component operation can behave differently even though the same *parentoperation* initiated the request processing. These control flow differences are represented in RDSEFFs using probability branches (Becker/Koziolek/Reussner, 2009). The probability of each branch (=*BranchDescriptor*) can be calculated based on data in *BranchMetrics* objects. If only one *BranchDescriptor* object exists for a *ParentOperationBranch* object, the probability is one. Otherwise, the *invocationCount* sum of all *BranchMetrics* objects for a *ParentOperationBranch* is used to calculate the probability for a single probability branch in a RDSEFF. An example for such probability branches can be found in figure 7.2(b). The RDSEFF of *ComponentA.operationA* contains two probability branches (*Control Flow 1* and *Control Flow 2*). One is executed with 60 % probability whereas the second is executed

with 40 % probability. The *OperationCallLoopCounts* for different *ExternalOperationCalls* in a specific branch are represented as loop probabilities. For example, in figure 7.2(b), the external call to *operationC* of *ComponentB* is executed five times in 20 % of the cases and nine times in the other 80 %.

Resource demand data in *BranchMetric* objects is also represented in a probability branch of a RDSEFF. The mean CPU demand in milliseconds (ms) calculated based on the *BranchMetrics* data can be directly assigned to an internal action of a probability branch. In the example in figure 7.2(b), *ComponentA.operationA* consumes 0.123 ms CPU time in *Control Flow 1*, whereas *Control Flow 2* consumes 0.456 ms.

Representing heap memory demand of a component operation is not directly supported by the PCM meta-model. Therefore, the passive resources element of the meta-model is reused for this purpose (Becker/Koziolek/Reussner, 2009). Even though passive resources are intended to be used as semaphores or to represented limited pool sizes (e.g., for database connections), one can also use them to create a simplistic representation of the memory demand of an application. For this purpose, a heap component is generated in each repository model as shown in figure 7.2(d). This heap component contains a specified amount of passive resources that represents the maximum heap size available in a JVM. The maximum configurable value for the available passive resources of the heap component is $2^{31}$-1. Thus, if one interprets one passive resource as one byte (B), the maximum configurable heap is two gigabytes (GB). As this is a very low value for Java EE applications nowadays, the model generation can be configured to interpret one passive resource as 10 bytes, so that the maximum representable heap is 20 GB. To do this, all heap memory demands read from the *BranchMetrics* objects are divided by ten and are rounded because passive resources can only be acquired as integer values. As this reduces the accuracy of the model, one passive resource is interpreted as one byte by default.

To allow other component operations in the repository model to consume heap memory, the heap component offers two operations: *alloc(int bytesRequested)* and *free(int bytesReleased)* (see figure 7.2(d)). This model follows the API for applications written in the programming language C. Using the information about the heap demand gathered in the data collection step (see section 7.2.1), calls to the *Heap.alloc* operation are generated at the beginning of each execution flow and calls to *Heap.free* at the end. An example is shown in figure 7.2(e): *operationD* of *ComponentC* calls *alloc* with a value of 200 bytes at the beginning, performs some internal processing and releases the 200 bytes allocated previously. Even though this memory model representation is not realistic for Java applications as the garbage collector (GC) behavior is not represented, the overall utilization of the passive resources helps to get an idea of the heap memory demand of an application.

## 7.3 Evaluating the Performance Prediction Accuracy

The feasibility of the model generation approach is evaluated in a case study using a SPECjEnterprise2010[1] industry standard benchmark deployment. SPECjEnterprise2010

[1]SPECjEnterprise is a trademark of the Standard Performance Evaluation Corp. (SPEC). The SPECjEnterprise2010 results or findings in this publication have not been reviewed or accepted by SPEC,

is used for this evaluation to make it reproducible as it defines an application, a workload as well as a dataset for a benchmark execution.

### *7.3.1 SPECjEnterprise2010 Deployment*

The SPECjEnterprise2010 benchmark represents the business case of an automobile manufacturer. It is divided into three application domains. The evaluation in this paper focuses on the Orders domain. This domain is used by automobile dealers to order and sell cars. To avoid the need to model all domains, the communication between the Orders and the other domains is disabled. The setup of the Orders domain consists of a benchmark driver to generate load and a system under test (SUT) on which the Orders domain application components are executed. The Orders domain is a Java EE web application that is composed of Servlet, JSP and EJB components. The automobile dealers (hereafter called users) access this application using a web interface over the hypertext transfer protocol (HTTP) and can perform three different business transactions: browse, manage and purchase. These three business transactions are composed of several HTTP requests to the system. The user interactions with the system are implemented as load test scripts in the Faban harness[2]. Faban is a workload creation and execution framework which is used to generate load on the SUT.

The benchmark driver and the SUT are each deployed on a virtual machine (VM) to simplify changing the number of available CPU cores. These two virtual machines are connected by a one gigabyte-per-second network connection and are mapped to an IBM System X3755M3 hardware server which is exclusively used for the SPECjEnterprise2010 benchmarks performed for this evaluation. Both virtual machines run CentOS 6.4 as operating system and are configured to have 20 GB of random-access memory (RAM). The benchmark driver is equipped with eight CPU cores, the number of CPU cores of the SUT is varied during the evaluation. The SPECjEnterprise2010 Orders domain application is deployed on a JBoss Application Server (AS) 7.1.1 in the Java EE 6.0 full profile. The database on the SUT VM is an Apache Derby DB in version 10.9.1.0. The JBoss AS and the Apache Derby DB are both executed in the same JVM, which is a 64 bit Java OpenJDK Server VM in version 1.7.0. An external Java-based client for the model generation is connected to the SUT using the JBoss JMX remote API.

### *7.3.2 Evaluating the Data Collection Overhead*

The model generation approach introduced in this work relies on data collected using a runtime instrumentation. The instrumentation overhead for collecting the required data is analyzed in this section. As mentioned in the data collection section (see section 7.2.1), the instrumentation is capable of collecting the CPU and Java heap memory demand for each externally accessible component operation. The instrumentation code is therefore

therefore no comparison nor performance inference can be made against any published SPEC result. The official web site for SPECjEnterprise2010 is located at http://www.spec.org/jEnterprise2010.

[2]http://java.net/projects/faban/

always executed before and after a component operation and can have a great influence on the performance data stored in the performance model.

To evaluate the impact of the data collection, the resource demand of several control flows of the SPECjEnterprise2010 Orders domain application is analyzed using different data collection configurations. For each of the following data collection configurations a SPECjEnterprise2010 benchmark run is executed and afterwards a performance model is generated. The SPECjEnterprise2010 deployment outlined in section 7.3.1 is used in a configuration with four CPU cores for the SUT. To avoid influences of warm up effects and varying user counts, only steady state data (i.e., data collected during 10 minutes between a five minute ramp up and a 150 second ramp down phase) is collected. The data collection runs are executed with a workload of 600 concurrent users which corresponds to a CPU utilization of the SUT of about 50 %. The resulting performance models contain the aggregated resource demands collected in the MBeans for these component operations and therefore simplify the analysis. The resource demand for the database is already included in the following measurements, as the embedded derby DB is executed in the same thread as the Servlet, JSP and EJB components.

The mean CPU and heap demands for single component operations involved in three different control flows are shown in tables 7.2, 7.3 and 7.4. Both resource demand types (CPU and heap) are represented as mean values for the data collected during the steady state. The heap demand values in this table are rounded to 10 byte intervals as the model generation is configured to do so to have 20 GB of heap available in the model (see section 7.2.1).

In a first step, a benchmark run is executed while the CPU and heap demand for all component operations involved in the request processing is collected. The performance model generated based on this configuration is called *Model 1.1* in tables 7.2, 7.3 and 7.4. Afterwards, a benchmark run is executed while only the CPU demand for each component operation is collected. The resulting performance model based on this data collection configuration is called *Model 2.1* in tables 7.2, 7.3 and 7.4. Both benchmark runs are repeated but this time, only resource demand data (CPU or CPU & heap) for the first component operations (those where Order==1 in tables 7.2, 7.3 and 7.4) of each HTTP request is collected. These measurements already include the CPU and heap demands of the sub-invocations (Order >1 in tables 7.2, 7.3 and 7.4). The resulting performance models are called *Model 1.2* for the first configuration (CPU and heap collection turned on) and *Model 2.2* for the second configuration (CPU collection turned on).

The total mean CPU and heap demand values in the first model versions (*1.1* and *2.1*) are compared with the corresponding values for the second model versions (*1.2* and *2.2*) to calculate the instrumentation overhead. It can be shown that collecting heap and CPU demands for each component operation is a lot more expensive than only collecting CPU demand. For the HTTP request analyzed in table 7.2, the mean overhead for the data collection including heap demand is 0.116 ms CPU time and 1378 byte heap memory for each component operation. If only the CPU demand is collected, the mean data collection overhead drops dramatically to 0.003 ms for each component operation.

Other execution flows during the same benchmark runs confirm these values (two additional examples are given in tables 7.3 and 7.4). The mean instrumentation overhead for the

| Component Operation | | Model 1.1 | | Model 1.2 | | Model 2.1 | Model 2.2 |
|---|---|---|---|---|---|---|---|
| **Order** | **Name** | **CPU** | **Heap** | **CPU** | **Heap** | **CPU** | **CPU** |
| 1 | app.sellinventory | 1.023 ms | 33,650 B | 3.001 ms | 225,390 B | 0.756 ms | 3.003 ms |
| 2 | CustomerSession.sellInventory | 0.785 ms | 60,450 B | | | 0.731 ms | |
| 3 | CustomerSession.getInventories | 0.594 ms | 49,540 B | | | 0.548 ms | |
| 4 | OrderSession.getOpenOrders | 0.954 ms | 70,600 B | | | 0.878 ms | |
| 5 | dealerinventory.jsp.sellinventory | 0.108 ms | 16,660 B | | | 0.103 ms | |
| **Total Resource Demand** | | 3.464 ms | 230,900 B | 3.001 ms | 225,390 B | 3.015 ms | 3.003 ms |
| **Mean Data Collection Overhead** | | 0.116 ms | 1378 B | | | 0.003 ms | |

**Table 7.2:** *Measured instrumentation overhead for the data collection - control flow one*

| Component Operation | | Model 1.1 | | Model 1.2 | | Model 2.1 | Model 2.2 |
|---|---|---|---|---|---|---|---|
| **Order** | **Name** | **CPU** | **Heap** | **CPU** | **Heap** | **CPU** | **CPU** |
| 1 | app.view_items | 0.406 ms | 20,560 B | 3.529 ms | 615,440 B | 0.165 ms | 3.566 ms |
| 2 | ItemBrowserSession.browseForward | 3.315 ms | 565,130 B | | | 3.282 ms | |
| 3 | ItemBrowserSession.getCurrentMin | 0.003 ms | 60 B | | | 0.003 ms | |
| 4 | ItemBrowserSession.getCurrentMax | 0.003 ms | 60 B | | | 0.002 ms | |
| 5 | ItemBrowserSession.getTotalItems | 0.003 ms | 60 B | | | 0.002 ms | |
| 6 | purchase.jsp.view_items | 0.147 ms | 40,380 B | | | 0.142 ms | |
| **Total Resource Demand** | | 3.877 ms | 626,250 B | 3.529 ms | 615,440 B | 3.598 ms | 3.566 ms |
| **Mean Data Collection Overhead** | | 0.070 ms | 2162 B | | | 0.006 ms | |

**Table 7.3:** *Measured instrumentation overhead for the data collection - control flow two*

| Component Operation | | Model 1.1 | | Model 1.2 | | Model 2.1 | Model 2.2 |
|---|---|---|---|---|---|---|---|
| **Order** | **Name** | **CPU** | **Heap** | **CPU** | **Heap** | **CPU** | **CPU** |
| 1 | app.add_to_cart | 0.213 ms | 11,300 B | 0.393 ms | 27,520 B | 0.108 ms | 0.388 ms |
| 2 | OrderSession.getItem | 0.276 ms | 13,460 B | | | 0.255 ms | |
| 3 | shoppingcart.jsp.add_to_cart | 0.059 ms | 5960 B | | | 0.058 ms | |
| **Total Resource Demand** | | 0.548 ms | 30,720 B | 0.393 ms | 27,520 B | 0.421 ms | 0.388 ms |
| **Mean Data Collection Overhead** | | 0.077 ms | 1600 B | | | 0.017 ms | |

**Table 7.4:** *Measured instrumentation overhead for the data collection - control flow three*

CPU-only collection is mostly below 0.020 ms whereas the mean instrumentation overhead for the CPU and heap collection ranges mostly between 0.060 and 0.120 ms. As some component operations in the SPECjEnterprise2010 benchmark have an overall CPU demand of below 0.150 ms, collecting the heap demand for this deployment causes too much overhead. The following evaluation therefore focuses on models generated based on the CPU demand collection.

### *7.3.3 Comparing Measured and Simulated Results*

In the next two sections, the prediction accuracy of generated performance models is evaluated in an upscaling and a downscaling scenario. The steps for both evaluations are similar and are described in the following paragraphs.

Load is generated on the SUT to gather the required data for the model generation in each scenario using the data collection approach outlined in section 7.2.1. As the database is included within the server JVM, the collected data already contains its CPU demands. Similar to the benchmark runs in the overhead evaluation, only steady state data (i.e., data collected during 10 minutes between a five minute ramp up and a 150 second ramp down phase) is collected. Afterwards, a software prototype that implements the performance model generation approach is used to generate a PCM model based on the collected data.

PCM models can be taken as the input for a simulation engine to predict the application performance for different workloads and resource environments. The standard simulation engine for PCM models is SimuCom which uses model-2-text transformations to translate PCM models into Java code (Becker/Koziolek/Reussner, 2009). The code is then compiled and executed to start a simulation. To evaluate the accuracy of the simulation results, they are compared with measurements on the SUT. The following comparisons only use steady state data collected during simulation and benchmark runs of similar length.

The benchmark driver reports the mean response time and throughput for each business transaction for a benchmark run. However, the predicted response time values cannot be compared with response time values reported by the driver, because they do not contain the network overhead between the driver and the SUT. Therefore, response time of the business transactions browse (B), manage (M) and purchase (P) is measured on the SUT using an additional instrumentation. To identify business transactions using this instrumentation, the benchmark driver is patched to add a unique transaction identifier to each request. This identifier allows combining several HTTP requests into one business transaction. Incoming requests are aggregated on the fly to the business transaction they belong to by summing up their response times. The resulting business transaction response time measurements are stored with a timestamp to calculate the mean throughput on a per-minute basis.

The CPU time consumed by the JVM process of the JBoss AS on the SUT (and thus its CPU utilization) is collected every second to reconstruct its approximate progression and to compare the measured and simulated ranges. The calculation of the mean CPU utilization is based on the first and the last CPU time consumption value in the steady state of a benchmark run in order to avoid biasing effects caused by unequal measurement intervals.

Each benchmark run is performed three times, the results are combined giving each run the same weight. Since all runs have the same duration, the overall mean value of CPU utilization can be calculated by averaging the corresponding values of each run. The throughput values represent the amount of times a business transaction is invoked per minute, thus the collected per-minute values are combined to a mean value. To evaluate response times, samples of equal sizes are drawn from each result. Response time measurement and simulation results are described using mean and median values as well as values of dispersion, namely the quartiles and the interquartile range (IQR). Variance and standard deviation are excluded from our investigation due to the skewness of the underlying distributions (Jain, 1991) of the response times of browse, manage and purchase. In the following sections, means are illustrated tabularly, medians and quartiles are illustrated using boxplot diagrams.

### *7.3.4 Evaluating Prediction Accuracy in an Upscaling Scenario*

To evaluate the performance prediction accuracy of automatically generated performance models in an upscaling scenario, the number of CPU cores for simulation and benchmark runs is increased step by step. To increase the CPU core count of the SUT for the

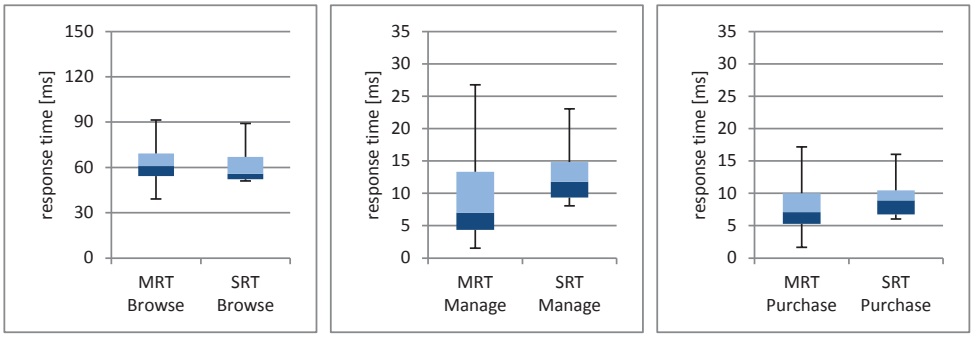
benchmark runs, the VM is reconfigured accordingly. The number of simulated CPU cores is varied by editing the generated resource environment model.

If workload stays stable, the CPU utilization significantly declines with each increase of cores as does its impact on the overall application performance. As a result, after reaching a sufficient number of CPU cores, the measured response times stay almost constant regardless of any further increases while the simulated response times decrease further reaching their lower bound only at a very high number of CPU cores. Therefore, an increasing inaccuracy in the simulated values is expected since the generated model solely depends on CPU demands and disregards other factors such as I/O operations on hard disk drives. Thus, to keep the CPU utilized, the workload on the system is varied proportional to the number of CPU cores by increasing the number of concurrent users accessing the SUT. In the following, a performance model generated on the SUT configured with 4 CPU cores is used. The average CPU utilization while gathering the data required for the model generation was 52.46 % which corresponds to a closed workload consisting of 600 users with an average think time of 9.9 s.

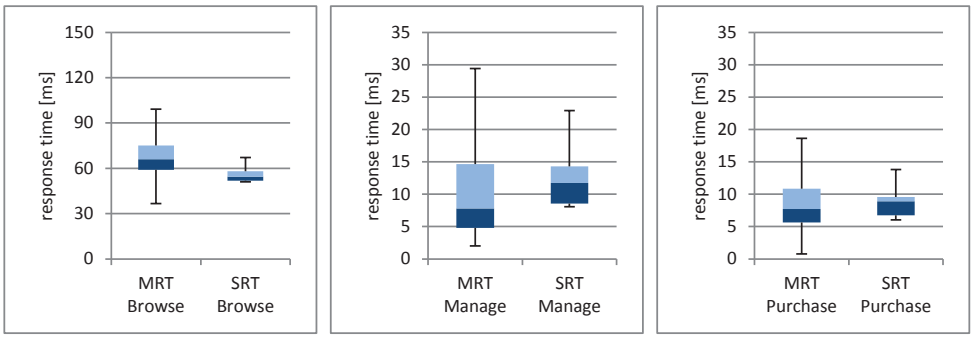
In a first step, the generated model is evaluated by simulating the application performance for an environment which is equal to the one the model has been generated with. Afterwards, the model is evaluated for environments with an increased number of CPU cores. The measured and simulated results are shown in table 7.5. For each configuration specified by the number of cores (C) and the number of users (U), the table contains the following data per business transaction (T): Measured Mean Response Time (MMRT), Simulated Mean Response Time (SMRT), relative Response Time Prediction Error (RTPE), Measured Mean Throughput (MMT), Simulated Mean Throughput (SMT), relative Throughput Prediction Error (TPE), Measured (MCPU) and Simulated (SCPU) Mean CPU Utilization and the relative CPU Utilization Prediction Error (CPUPE).

The simulation predicts the mean response time of the business transactions with a relative error of less than 20 %, except for the browse transaction in the case of 8 CPU cores and 1200 concurrent users, which shows a relative prediction error of 36.66 %. CPU utilization is predicted with relative errors ranging from 3.88 % to 18.09 %. Due to space limitations, the span consisting of the minimum and maximum of the measured and simulated CPU utilization values is not shown. However, while both ranges mostly overlap, the measured span lies slightly above the simulated one. The same applies to the mean CPU utilization values shown in table 7.5, as the simulated mean is slightly lower than the measured one. The prediction of the mean throughput is very close to the real values, as the think time of 9.9 s is much higher than the highest response time. Response time prediction errors thus have a low impact on the throughput. Except for the last simulation of browse, the quality of the predictions ranges from very good to still acceptable for the purpose of capacity planning (Menascé et al., 2004).

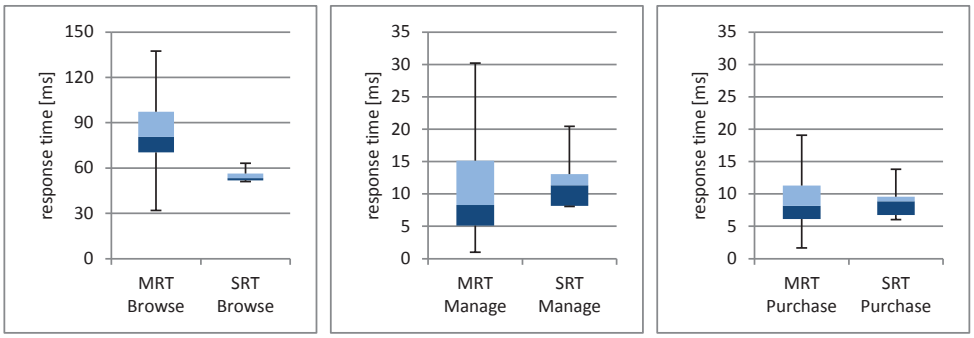
Further statistical measures are illustrated as boxplot diagrams in figure 7.3. Boxplot diagrams consist of a box whose bounds denote the first quartile $Q_1$ (lower bound) as well as the third quartile $Q_3$ (upper bound) of the underlying data sample. The quartiles are connected by vertical lines to form the box that indicates the interquartile range (IQR) which is defined as $Q_3 - Q_1$. Furthermore, the median $Q_2$ is illustrated by a horizontal line within the box, thus separating it into two parts. Vertical lines outside the box (whiskers) indicate the range of possible outliers while their length is limited to 1.5 times the $IQR$.




(a) 4 CPU cores and 600 users




(b) 6 CPU cores and 900 users




(c) 8 CPU cores and 1200 users

**Figure 7.3:** *Boxplot diagrams of an upscaling scenario*

The relative prediction error of the median response time ranges from 8.38 % to 33.80 % for the browse and purchase transactions. The median response time of the manage transaction, however, is predicted with a relative error of 36.52 % to 68.29 %. The skewness of a business transaction's underlying distribution can be determined considering the median's position between the quartiles $Q_1$ and $Q_3$. The boxplot diagrams in figure 7.3 show that the skewness is not simulated correctly. To investigate the dispersion of business transactions, we determine the IQR. Its relative prediction error ranges from 21.96 % to 50.94 % for the manage and purchase transactions and is up to 83.13 % for the browse transaction.

In the measurement results, the effect of increasing the workload dominates, thus the measured CPU utilization slightly increases from 48.76 % to 57.34 %. The response times increase accordingly. In the simulation results, the effect of core increase slightly dominates over the effect of increasing the workload. Therefore, the response times slightly decrease

| C | U | T | MMRT | SMRT | RTPE | MMT | SMT | TPE | MCPU | SCPU | CPUPE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 600 | B | 63.23 ms | 65.06 ms | 2.91 % | 1820.6 | 1813.1 | 0.41 % | 48.76 % | 46.87 % | 3.88 % |
| | | M | 11.58 ms | 13.28 ms | 14.71 % | 906.8 | 917.3 | 1.16 % | | | |
| | | P | 8.27 ms | 9.73 ms | 17.67 % | 904.9 | 900.3 | 0.50 % | | | |
| 6 | 900 | B | 69.25 ms | 57.56 ms | 16.89 % | 2708.3 | 2721.5 | 0.49 % | 51.72 % | 46.85 % | 9.42 % |
| | | M | 12.54 ms | 11.95 ms | 4.69 % | 1354.3 | 1354.4 | 0.01 % | | | |
| | | P | 8.95 ms | 8.72 ms | 2.60 % | 1352.4 | 1368.1 | 1.16 % | | | |
| 8 | 1200 | B | 88.82 ms | 56.25 ms | 36.66 % | 3617.8 | 3641.9 | 0.67 % | 57.34 % | 46.97 % | 18.09 % |
| | | M | 14.13 ms | 11.64 ms | 17.67 % | 1806.4 | 1795.0 | 0.63 % | | | |
| | | P | 9.31 ms | 8.46 ms | 9.15 % | 1811.6 | 1819.2 | 0.42 % | | | |

**Table 7.5:** *Measured and simulated results in an upscaling scenario*

over the course of the experiment, while the simulated CPU utilization remains almost constant.

### *7.3.5 Evaluating Prediction Accuracy in a Downscaling Scenario*

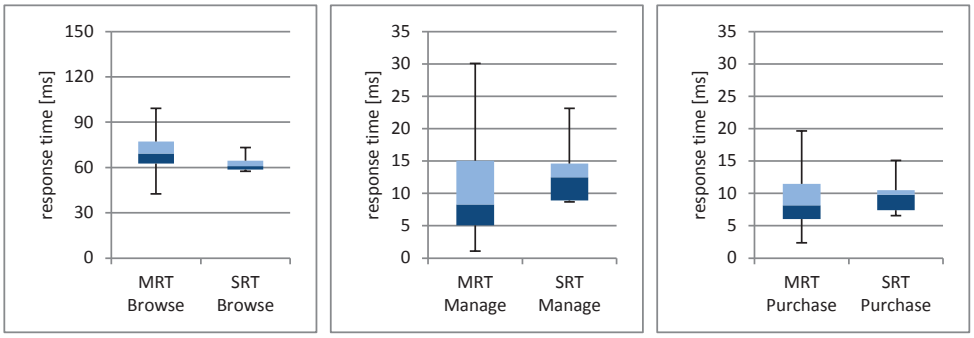
The prediction accuracy of generated performance models in a downscaling scenario is evaluated by reducing the number of CPU cores step by step. Starting with 8 CPU cores, the number of cores is decreased by 2 in each evaluation step. This scenario does not require the number of users to be varied, as the CPU utilization increases. The business case of scaling the number of CPU cores down is to optimize production systems (e.g., to evaluate if several applications can be hosted on one machine or to reduce license fees). In this case, the number of users does not change. Therefore, the workload is kept constant at 800 users with a think time of 9.9 s accessing the SUT in parallel.

Since the number of cores is reduced during the experiment, a sufficiently low starting value of CPU utilization is required. Therefore, data to generate a performance model for this evaluation is collected with an average CPU utilization of 38.9 %. To compare the simulation results with the measured ones, the previously described evaluation process is applied. The comparison of the mean response time, CPU utilization and throughput values is shown in table 7.6.

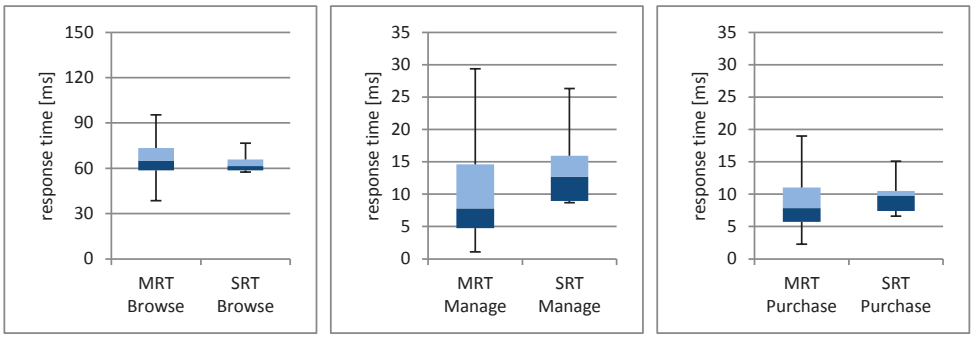
The relative prediction error for the mean response time of all business transactions is at most 44.33 %. CPU utilization is predicted with a maximum relative error of 7.12 %. In contrast to the upscaling scenario, the simulated CPU utilization grows slightly above the measured results as the CPU cores are decreased. The relative prediction error of the mean throughput is about 1 %.

The relative prediction error of the median response time as shown in the boxplots in figure 7.4 ranges from 5.27 % to 38.50 % for the browse and purchase transactions and from 50.51 % to 82.96 % for the manage transaction. This is in line with the observations previously made in the upscaling scenario. The relative IQR prediction error ranges from 27.34 % to 43.48 % for the manage and purchase transaction; for the browse transaction it is up to 83.37 %.

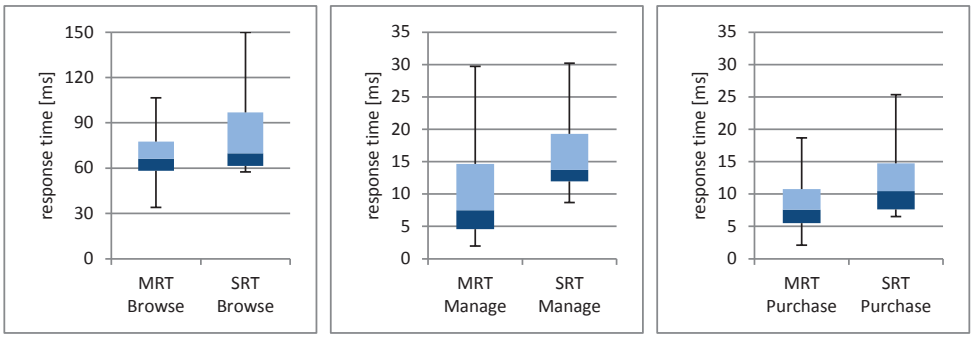
Comparing the measured mean and median response times shows that the lowest values are achieved in the 6 CPU core configuration. Even with 4 CPU cores, the browse and




(a) 8 CPU cores and 800 users




(b) 6 CPU cores and 800 users




(c) 4 CPU cores and 800 users

**Figure 7.4:** *Boxplot diagrams of a downscaling scenario*

purchase response times are lower than in the 8 CPU core configuration. As the CPU utilization of the investigated configurations is relatively low, the lower performance of the SUT with 8 CPU cores can be explained by an increased scheduling overhead. Due to the low CPU utilization, its impact on the overall performance of the SUT is lower than the impact of other factors such as I/O operations. The response time prediction behaves incorrectly in these cases, as the generated performance model only relies on the CPU demand measured during the data collection step and does not take these effects into account. However, the simulation of CPU utilization is still very close to the measurements. This is useful for determining a lower bound of feasible configurations regarding the amount of cores. Simulating an environment consisting of 3 CPU cores results in a simulated CPU utilization of 91.96 % and indicates that this would lead to instability of the SUT for the given workload. This configuration is thus not investigated in this downscaling scenario.

| C | U | T | MMRT | SMRT | RTPE | MMT | SMT | TPE | MCPU | SCPU | CPUPE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 800 | B | 71.54 ms | 64.03 ms | 10.50 % | 2413.9 | 2415.8 | 0.08 % | 37.41 % | 35.17 % | 5.99 % |
| | | M | 12.96 ms | 12.64 ms | 2.49 % | 1203.5 | 1209.2 | 0.48 % | | | |
| | | P | 9.36 ms | 9.33 ms | 0.25 % | 1215.9 | 1228.7 | 1.05 % | | | |
| 6 | 800 | B | 67.62 ms | 66.03 ms | 2.35 % | 2413.9 | 2425.4 | 0.48 % | 46.38 % | 46.94 % | 1.21 % |
| | | M | 12.52 ms | 13.08 ms | 4.45 % | 1202.0 | 1196.6 | 0.45 % | | | |
| | | P | 9.05 ms | 9.64 ms | 6.57 % | 1208.2 | 1215.0 | 0.56 % | | | |
| 4 | 800 | B | 71.15 ms | 87.46 ms | 22.92 % | 2437.0 | 2420.8 | 0.66 % | 65.60 % | 70.27 % | 7.12 % |
| | | M | 12.98 ms | 17.04 ms | 31.29 % | 1199.7 | 1193.5 | 0.51 % | | | |
| | | P | 8.93 ms | 12.88 ms | 44.33 % | 1211.6 | 1212.1 | 0.04 % | | | |

**Table 7.6:** *Measured and simulated results in a downscaling scenario*

## 7.4 Related Work

Running Java EE applications have already been evaluated using performance models by several authors. Chen et al. (2005) derive mathematical models from measurements to create product-specific performance profiles for the EJB runtime of a Java EE server. These models are intended to be used for performance predictions of EJB components running on different Java EE products. Their approach is thus limited to Java EE applications that solely consist of this component type.

Liu/Kumaran/Luo (2001) also focus on EJB components and show how layered queuing networks can be used for the capacity planning of EJB-based applications. In their work, they model an EJB-based application manually and describe how an analytical performance model needs to be calibrated before it can be used for the capacity planning. To improve this manual process, Mania/Murphy (2002) proposed a framework to create analytical performance models for EJB applications automatically. However, the framework was never evaluated to the best of our knowledge.

The difficulties in building and calibrating performance models for EJB applications manually are also described by McGuinness/Murphy/Lee (2004). Instead of using analytical solutions to predict the performance of an EJB-based application, they are using simulation models. The authors argue that simulation models are better suited for the performance evaluation of EJB applications due to their flexibility and increased accuracy compared to analytical models.

The applicability of analytical performance models for Java EE applications with realistic complexity is analyzed by Kounev/Buchmann (2003) using the SPECjAppServer2002 industrial benchmark. Kounev extends this work in Kounev (2006) by using queuing Petri nets to evaluate the performance of a SPECjAppServer2004 benchmark deployment. The latest version of the SPECjAppServer benchmark (SPECjEnterprise2010) is used by Brosig/Huber/Kounev (2011) to show that they are able to semi-automatically extract PCM models for Java EE applications. Their model generation approach is based on data generated by the monitoring framework of Oracle's WebLogic product and thus not transferable to other Java EE server products. It also requires manual effort to distribute the resource demand based on the service demand law once a model is generated.

The previous work is extended by the approach introduced in this work as it is applicable for all Java EE server products and can generate performance models for EJB as well as for web components automatically.

## 7.5 Conclusion and Future Work

The approach presented in this work aims to make performance modeling better applicable in practice. The ability to generate performance models at any time simplifies their use in Java EE development projects, as the effort to create such models is very low. The evaluation showed that the generated performance models predict the performance of a system in up- and downscaling scenarios with acceptable accuracy. The approach can thus support related activities during the capacity planning and management processes.

Future work for this approach includes extending the data collection and model generation capabilities. First of all, we need to investigate whether the user session information available in the Java EE runtime can be used to generate usage models automatically. Further extensions are required to support additional technologies specified under the umbrella of the Java EE specification, such as JavaServer Faces (JSF) or web services. For this purpose, the model generation approach also needs to be extended to support distributed systems. A key challenge for such an extension is the integration and correlation of MBean data collected from multiple Java EE servers. Additional improvements are required to reduce the instrumentation overhead as soon as heap demand needs to be collected.

## 7.6 Acknowledgements

The authors would like to thank Jörg Henß, Klaus Krogmann and Philipp Merkle from the Karlsruhe Institute of Technology (KIT) and FZI Research Center for Information Technology at KIT for their valuable input and support while implementing the heap representation approach in PCM.

# Chapter 8

# Using Architecture-Level Performance Models as Resource Profiles for Enterprise Applications

| | |
|---|---|
| Authors | Brunnert, Andreas[1] (brunnert@fortiss.org)<br>Wischer, Kilian[2] (wischer@in.tum.de)<br>Krcmar, Helmut[2] (krcmar@in.tum.de)<br><br>[1]fortiss GmbH, Guerickestraße 25, 80805 München, Germany<br>[2]Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany |
| Outlet | 10th International Conference on the Quality of Software Architectures (QoSA, 2014) |
| Status | Accepted |
| Contribution of first author | Problem and scope definition, construction of the conceptual approach, experiment design and result analysis, paper writing, paper editing |

**Table 8.1:** *Fact sheet publication P5*

**Abstract** The rising energy and hardware demand is a growing concern in enterprise data centers. It is therefore desirable to limit the hardware resources that need to be added for new enterprise applications (EA). Detailed capacity planning is required to achieve this goal. Otherwise, performance requirements (i.e. response time, throughput, resource utilization) might not be met. This paper introduces resource profiles to support capacity planning. These profiles can be created by EA vendors and allow evaluating energy consumption and performance of EAs for different workloads and hardware environments. Resource profiles are based on architecture-level performance models. These models allow to represent performance-relevant aspects of an EA architecture separately from the hardware environment and workload. The target hardware environment and the expected workload can only be specified by EA hosts and users respectively. To account for these distinct responsibilities, an approach is introduced to adapt resource profiles created by EA vendors to different hardware environments. A case study validates this concept by creating a resource profile for the SPECjEnterprise2010 benchmark application. Predictions using this profile for two hardware environments match energy consumption and performance measurements with an error of mostly below 15 %.

## 8.1 Introduction

Enterprise applications are the backbone of many business processes. These applications need to meet performance requirements (i.e. response time, throughput, resource utilization) to avoid problems during the process execution. Detailed capacity planning effort (Menascé et al., 2004) is therefore required before new enterprise applications are deployed in a data center. This effort is driven by a basic question: How much hardware resources are required to fulfill performance requirements for the expected workload? This question leads to a more specific inquiry about the applications demand for hardware resources such as central processing units (CPU), hard disk drives (HDD) and memory. The capacity planning finally requires an assessment of the workload impact on the resource demand.

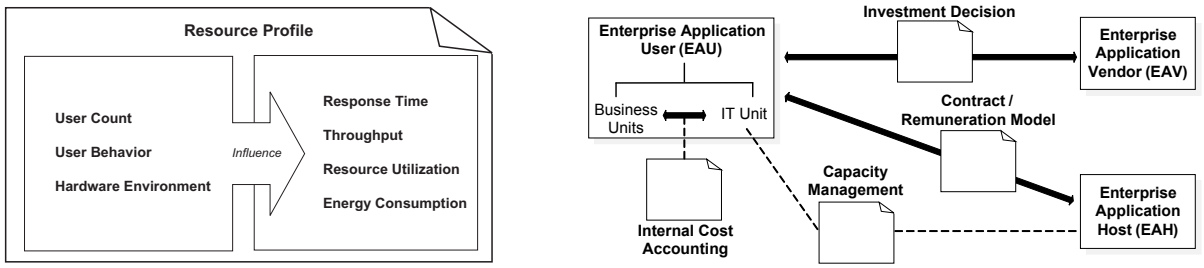
At the same time, the rising energy consumption is a major cost driver in data centers nowadays (Fan/Weber/Barroso, 2007; Rivoire et al., 2007). The energy consumption is defined as the power consumption integrated over time. It would thus be beneficial to know how much power will be consumed by the resources utilized by an application in beforehand (Capra et al., 2010).

Estimating hardware requirements and energy consumption of new enterprise application deployments is only possible if different parties work together (Brunnert et al., 2014). First of all, enterprise application vendors (EAV, i.e. software or consulting companies) need to quantify the resource demand of their software products. Enterprise application users (EAU, i.e. companies who source software from EAVs) need to specify the expected workload for their use cases. Finally, enterprise application hosts (EAH, i.e. data center providers) need to specify the characteristics of the hardware environment on which the applications can be deployed. Nowadays, the communication between these parties lacks a medium in which all these aspects are captured. The resource profile concept introduced in this work serves as such a communication medium and helps to answer the questions raised above.

This work proposes the use of architecture-level performance models to represent resource profiles. The contribution of this work therefore includes the resource profile concept as a use case for these models in between their traditional application domains: software performance engineering (Woodside/Franks/Petriu, 2007) and application performance management (Menascé, 2002b). Additionally, an approach is presented to adapt these models to different hardware environments. Furthermore, a performance modeling approach is extended to allow predictions of the energy consumption. A case study finally evaluates the feasibility of the resource profile concept.

## 8.2 Resource Profiles

Resource profiles are models that allow evaluating energy consumption and performance of enterprise applications (see figure 8.1(a)). In an ideal case, a resource profile is constructed once by an EAV and can then be used by EAUs and EAHs for several use cases as shown in figure 8.1(b). The intended use cases are similar to the ones of system requirements for end user desktop software (such as games or office tools). System requirements show




(a) Resource profile (b) Use cases for a resource profile

**Figure 8.1:** *Resource profiles for enterprise applications*

whether a user is able to run a software on his current environment. If an end user needs to modify his soft- or hardware environment, this often has a huge impact on his purchasing decision. If EAUs would have similar information at hand during a software purchasing scenario, it would also influence their investment decisions.

### *8.2.1 Content and Structure*

The energy consumption and performance of an enterprise application is influenced by several factors as depicted in figure 8.1(a). One of the key factors is the hardware environment on which an enterprise application is deployed. As hardware resources (e.g. the CPU) have different performance and power consumption characteristics, the resource demand (e.g. required CPU time) and energy consumption of single transactions is directly dependent on the components used within a server. Different hardware environments thus have a big impact on the performance and energy consumption of an application. Resource profiles therefore provide a means to represent different hardware environments.

Besides the hardware environment, the workload on a system directly influences the performance and energy consumption of an application. The workload of an enterprise application is typically specified by the amount of users accessing the system and their behavior (Menascé/Almeida, 2002). Depending on the workload, the hardware environment is utilized on different levels, which lead to varying performance and energy characteristics. Consequently, the user count and their behavior can be represented in a resource profile.

To evaluate the impact of different hardware environments and workloads on energy consumption and performance, a model is required that describes the performance-relevant aspects of an enterprise application architecture independently from these influencing factors. These aspects include the components of an enterprise application, their interfaces, relationships, control flow, parametric dependencies and resource demands (Koziolek, 2010). Resource profiles describe these aspects independently from the workload and hardware environment. To simplify their use, the aforementioned aspects are hidden for resource profile users (i.e. EAU, EAH). Resource profiles abstract these aspects on the level of detail of single deployment units of an enterprise application. These deployment units represent a collection of application components and thus reduce the complexity for the users. Instead of dealing with individual application components, they can use these deployment units for specifying allocations on different hardware environments.

To represent the workload, resource profiles describe external interfaces provided by an enterprise application (e.g. functionalities provided by user interfaces). These external interfaces can be used for specifying the workload. Based on these specifications, the influence of different workloads and hardware environments on performance and energy consumption can be evaluated as shown in figure 8.1(a).

The knowledge required for constructing a resource profile and for specifying the input variables for the evaluation is often distributed between different parties (i.e. EAV, EAU and EAH, see figure 8.1(b)). Resource profiles are therefore meant to be used differently depending on the available information. An EAV should create resource profiles for all enterprise applications sold (off-the-shelf and custom developments), which then can be adapted by EAUs and EAHs for their specific needs. They can modify the workload and the hardware environment but reuse the specifications provided by an EAV.

### *8.2.2 Use Case Examples*

The primary use case of a resource profile is to estimate the required hardware resources for an enterprise application. As this task is required in different contexts, the following paragraphs explain some use cases in which the transferable nature of resource profiles helps to simplify the relationships of EAUs, EAVs and EAHs.

If an EAU is interested in a new enterprise application, he could use the corresponding resource profile as one component in an overall investment decision. The EAU can specify the expected amount of users, their behavior and his existing hardware environment to evaluate if this hardware would be sufficient to run the application for his needs. At the same time, the EAU could evaluate the impact of this particular application on his energy bill. If new hardware is needed for the application, the resource profile helps to compare different hardware configurations in terms of their impact on performance and energy consumption. Resource profiles can also be used to choose between different off-the-shelf software products with similar functionality with regard to the above mentioned criteria.

When software is purchased and hosted by an EAU internally, a resource profile supports the internal cost accounting between different business units and the IT unit (Brandl/Bichler/Ströbel, 2007). Using resource profiles, the resource demand and power consumption of an enterprise application can be broken down to user profile levels or transaction classes. Brandl/Bichler/Ströbel (2007) showed how such a breakdown of the resource consumption on the level of transaction classes can be used to allocate costs to different business units according to their workload.

If an EAU does not want to host an application himself, resource profiles can be used to negotiate a contract between an EAU and an EAH (e.g. cloud providers). As cloud computing is gaining more popularity, the demand for usage-based costing will increase (Brandl/Bichler/Ströbel, 2007). Similar to the internal cost accounting approach explained above, EAUs and EAHs could agree on a remuneration model which is directly dependent on the resource and energy consumption of the hosted application (Li/Casale/Ellahi, 2010). Resource profiles help both parties to better estimate their costs in such a scenario.

If an enterprise application is already running in a production environment, resource profiles help in the capacity management process. For example, the impact of an increased user load on performance and energy consumption of an application can be examined in beforehand and appropriate conclusions can be drawn.

The next section explains the construction of resource profiles based on architecture-level performance models.

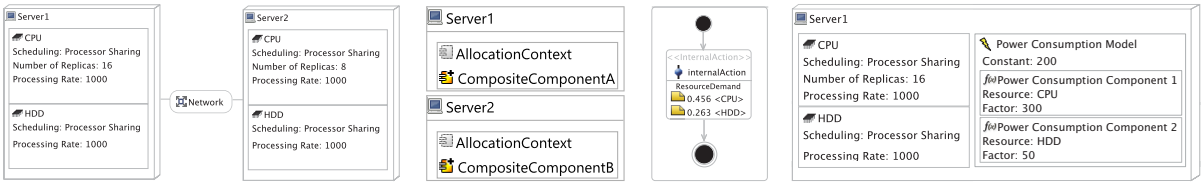
### *8.2.3 Performance Models as Resource Profiles*

Evaluating the performance of an application is a common problem in the software engineering domain. Numerous performance modeling approaches have been proposed to address this challenge (Balsamo et al., 2004; Koziolek, 2010). A performance model of an enterprise application typically contains performance-relevant aspects of an application architecture, the hardware environment and the workload. Using these models as input for analytical solvers or simulation engines allows predicting response time, throughput and resource utilization for the modeled software system. Several case studies (see section 8.4) showed the applicability of performance models for predicting these performance metrics.

Performance models thus seem to be generally suitable to represent resource profiles. As resource profiles must be adapted to different target environments, a key requirement for their representation is that performance-relevant aspects of an application architecture, the hardware environment and the workload can be modeled independently from each other. In an ideal case, an EAV can model the performance-relevant aspects of an application architecture and distribute this incomplete model to an EAU who complements the model with the expected workload. Afterwards, an EAH can add the hardware environment to the resource profile.

Conventional performance modeling approaches (Balsamo et al., 2004) such as Queuing Networks, Layered Queuing Networks (LQN) or Queuing Petri Nets depict all these aspects nested in one single monolithic performance model. It is therefore hard to change a single aspect like the hardware environment or the workload without needing to substantially change the whole performance model. Architecture-level performance models (Koziolek, 2010) try to separate these aspects to simplify the modeling process. A popular architecture-level performance model is the Palladio Component Model (PCM) (Reussner et al., 2011). PCM separates all the aspects mentioned above and is thus used as meta-model to represent resource profiles.

PCM is described by Reussner et al. (2011) as a software component model for business information systems to enable model-driven quality of service (QoS, i.e. performance) predictions. A software system is represented in PCM by several model layers which can reference each other (Reussner et al., 2011). The most important model within PCM is called repository model. This model contains the components of a software system and their relationships. The control flow of a component operation, its resource demand and parametric dependencies are specified in so called Resource Demanding Service Effect Specifications (RDSEFF). Components are assembled in a system model to represent an application. User interactions with the system are described in a usage model. The other




(a) Resource environment model (b) Allocation model (c) RD-SEFF (d) Power consumption model

**Figure 8.2:** *PCM models*

two model types in PCM are the resource environment and allocation model. A resource environment model allows to specify available resource containers (i.e. servers) with their associated hardware resources (CPU or HDD). An allocation model specifies the mapping of system model elements on resource containers.

A resource profile can thus be represented using this meta-model by creating a system model. Such a system model describes the external interfaces of an enterprise application and references repository model components (including RDSEFFs) to describe the performance-relevant aspects of an application architecture. Several approaches to construct these models, either based on design model transformations (Becker, 2008) or dynamic analysis (Brunnert/Vögele/Krcmar, 2013), already exist and are therefore not described in detail in this work. The workload for a PCM-based resource profile can be specified in a usage model. The hardware environment can be specified by the resource environment model whereas the deployment on this environment is specified in the allocation model.

Even though PCM provides a good foundation for building resource profiles, just creating a performance model is not sufficient for supporting the structure and use cases of a resource profile outlined earlier. The remainder of this section therefore focuses on questions that cannot be answered by the PCM modeling capabilities:

- How can resource profiles based on the PCM meta-model be adapted to different hardware environments?
- How can the PCM meta-model be extended to support energy consumption predictions?

### *8.2.4 Adapting Resource Profiles to Different Hardware Environments*

Hardware environments are specified in PCM resource environment models. An example of a resource environment model is shown in figure 8.2(a). In this example, the resource environment consists of two hardware servers which are connected via a network connection. The model depicts the CPU and HDD of both servers. The CPU of *Server1* consists of 16 cores (number of replicas) and the CPU of *Server2* consists of 8 cores. Each core has a processing rate of 1000. The HDDs of both servers also specify a processing rate of 1000. A processing rate of 1000 means that one CPU core respectively the HDD can process 1000 units of work within a simulated time frame. In this example, one simulated time

frame is interpreted as one second. Thus, each CPU core and HDD can process 1000 milliseconds (ms) of work per simulated time frame.

An allocation model defines how system model elements are mapped on servers in the resource environment model. To simplify the use of a resource profile for evaluating different deployment options, an EAV should represent indivisible deployment units using composite components (Reussner et al., 2011). Composite components allow to combine several repository model components so that they can only be allocated as a whole and not individually. This representation ensures that EAUs or EAHs cannot evaluate deployment options which are not supported by an EAV. The allocation model in figure 8.2(b) shows the allocation of two composite components on the two servers modeled in the resource environment model. *CompositeComponentA* is mapped on *Server1* and *CompositeComponentB* on *Server2*.

A component implements several operations which can be invoked by users or other components. The resource demand of an operation is defined in internal actions of an RDSEFF. It is specified as the amount of units of work needed on a particular hardware resource to be processed. This resource demand is thus modeled relative to the processing rate of a hardware resource in the resource environment model. Figure 8.2(c) shows a simplified RDSEFF. The internal action consumes 0.456 units of work on the corresponding CPU resource and 0.263 units of work on the HDD. If this component is mapped on a resource container with a CPU core that can process 1000 units of work within one second, the 0.456 units of work can also be interpreted as 0.456 ms CPU time required by the component operation. The same interpretation is valid for the HDD demand.

This dependency between resource demands in RDSEFFs and processing rates of hardware resources in resource environment models avoids the need to adapt resource demands of every internal action in every RDSEFF if hardware resources are changed. It is only necessary to adapt the processing rate of modeled hardware resources if they are replaced by other types of the same resource. If more attributes of a hardware environment are changed (e.g. the number of servers), the allocation model must also be changed to map the (composite) components on the new resources. What needs to be done to model these structural changes is described by Reussner et al. (2011).

If a resource profile is used to predict performance and energy consumption for a hardware environment that is not the one the resource profile was initially created with, the profile must be adapted to the target environment. To adapt the resource profile from one environment to another, the processing rate of resources must be scaled according to the performance of the hardware resources.

Following Menascé/Almeida (2002), the processing rate of a resource is scaled according to the hardware benchmark results of the initial and target hardware resource. After performing a suitable benchmark on the initial and the target server, we assume to get two benchmark scores of the investigated hardware resource. The benchmark score of the initial ($b_{initial}$) and target ($b_{target}$) server and the initial processing rate ($r_{initial}$) allow to calculate the new processing rate ($r_{target}$) for the target server's resource as follows:

$$r_{target} = \frac{b_{target}}{b_{initial}} * r_{initial} \tag{8.1}$$

For example, when benchmarking a specific hardware resource, an initial server gets a benchmark score of 40 whereas a score of 50 is achieved on the target server. By using formula 8.1 with an initial processing rate of 1000 a target processing rate of 1250 can be calculated. This calculation is possible for different hardware resources. For CPU benchmarks it is important that the benchmark can evaluate the performance of a single core, otherwise it is much harder to adapt the resource environment model from one server to another. If standardized benchmarks are used for this purpose, the benchmarks must not necessarily be performed by the user of a resource profile, as results for common hardware systems are often available on the web sites of the benchmark providers. Nevertheless, processing rates and benchmark scores of hardware resources used to derive resource demands need to be distributed along with a resource profile.

This approach assumes that all resource demands in the RDSEFFs of a repository model are initially derived from measurements on the same hardware types. Otherwise, it would be necessary to adapt the resource demands in an RDSEFF individually if a component is moved from one server to another in the allocation model. Similarly, the network traffic between all components needs to be represented in a resource profile even if the profile is created on a single machine. Without this information, it would not be possible to distribute components in a resource profile to different machines connected by a network.

### *8.2.5 Predicting Energy Consumption*

To the best of our knowledge, there is no performance modeling approach available which is able to predict the energy consumption of an application. Hence, it can also not be predicted using PCM. As energy consumption is defined as the power consumption integrated over time, the PCM meta-model is extended by a *power consumption model* element. The PCM simulation engines SimuCom (Becker, 2008) and EventSim (Merkle/Henß, 2011) are also extended to use the new *power consumption model* element. Simulation results now allow to evaluate the power consumption of servers in a resource environment over time. The integral of the resulting function can be used to predict the energy consumption of an application. The construction of a *power consumption model* is explained in the following.

PCM is already capable of predicting the utilization rate of modeled hardware resources such as CPU or HDD. The full server power consumption can thus be modeled based on the utilization of the server's hardware resources as shown in several existing works on this topic (Fan/Weber/Barroso, 2007; Rivoire/Ranganathan/Kozyrakis, 2008). Full server power consumption refers to the power consumed by the power adapters of the server. As shown in section 8.2.2, this is the key figure from an economic point of view.

Rivoire/Ranganathan/Kozyrakis (2008) and Fan/Weber/Barroso (2007) showed that simple linear power models based on resource utilization metrics produce very accurate results. Even very simple models, which only capture the CPU utilization to predict the power consumption, are very accurate. A linear model with the predicted power consumption of a server as the dependent variable $P_{pred}$ and multiple resource utilization

metrics as the independent variables $u_i$ can therefore be specified by the following equation (Fan/Weber/Barroso, 2007; Rivoire/Ranganathan/Kozyrakis, 2008):

$$P_{pred} = C_0 + \sum_{1}^{i} C_i * u_i \tag{8.2}$$

In PCM resource environment models a hardware server is represented by a resource container. The new *power consumption model* element is thus attached to the existing resource container meta-model element. It represents a linear model in the form of equation 8.2. Figure 8.2(d) shows an example of such a *power consumption model* for one server with 16 CPU cores and a HDD resource. A *power consumption model* contains a constant ($C_0$) which represents an approximation of the idle power consumption of a server. The independent variables of equation 8.2 are represented by multiple *power consumption components*. A *power consumption component* contains a multiplication factor ($C_i$) and a reference to the utilization of a hardware resource of the resource container ($u_i$). In the example of figure 8.2(d), the *power consumption model* represents the equation: $P_{pred} = 200 + 300 * u_{CPU} + 50 * u_{HDD}$. A CPU utilization of 50 % and a utilization of the HDD of 20 % would thus lead to a predicted power consumption of 360 watts (W).

The best way to construct such a linear *power consumption model* is a calibration run on the target hardware similar to the approach presented by Economou et al. (2006). In a calibration run, hardware resources are stressed independently from each other with changing intensity. Meanwhile the corresponding resource utilization metrics reported by the operating system and the resulting power consumption of the full server system are measured. Modern enterprise servers implement power measurement sensors for single hardware resources and the whole system. A common way to offer their measurements is to use the Intelligent Platform Management Interface (IPMI)[1]. Thus, there is often no need to use an external power meter device. After finishing the calibration run, a linear regression on the measured metrics is done to generate a linear *power consumption model*.

## 8.3 Evaluation

In this section, a case study using the SPECjEnterprise2010 industry standard benchmark demonstrates the feasibility of the resource profile concept. A resource profile based on the extended PCM meta-model is generated for the SPECjEnterprise2010[2] benchmark application on an initial hardware environment and is adapted to a target hardware environment by using the SPEC CPU2006[3] benchmark. Afterwards, workloads using varying amounts of users are executed both as a simulation using the resource profile and on deployments on the initial and target hardware environment. The simulated

[1] http://www.intel.com/design/servers/ipmi/

[2] SPECjEnterprise is a trademark of the Standard Performance Evaluation Corp. (SPEC). The official web site for SPECjEnterprise2010 is located at http://www.spec.org/jEnterprise2010.

[3] The SPECjEnterprise2010 and SPEC CPU2006 results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The results in this publication should thus be seen as estimates as the benchmark execution might deviate from official run rules. The official web site for SPEC CPU2006 is located at http://www.spec.org/cpu2006.

and measured response time, throughput, power consumption and resource utilization values are afterwards compared with each other. The evaluation steps and the quantitative validation ensure that the extended PCM meta-model provides a solid base for representing resources profiles with the properties explained in section 8.2.1.

### *8.3.1 SPECjEnterprise2010*

The SPECjEnterprise2010 benchmark application represents business processes of an automobile manufacturer and is divided into three different domains: the Supplier domain, the Manufacturing domain and the Orders domain. The Orders domain is used by automobile dealers to sell and order cars. By doing so, they drive the demand for the Manufacturing domain. This domain simulates car manufacturing sites. It interacts with the Supplier domain to order parts required during the manufacturing process.

The evaluation in this paper focuses on the Orders domain as it is intended to be used by end users, whereas the other two domains are used by other applications as (web-) services. The communication between the domains is disabled in order to avoid the need to model and evaluate all domains.

The Orders domain is a Java Enterprise Edition (EE) web application that is composed of Servlet, JavaServer Pages (JSP) and Enterprise JavaBean (EJB) components. The automobile dealers access this application using a web interface over the hypertext transfer protocol (HTTP). The automobile dealers can perform three different business transactions: Browse (B), Manage (M) and Purchase (P). These three business transactions are composed of several HTTP requests to the system.

The dealer interactions with the system are implemented as benchmark driver in the Faban harness[4]. Faban is a load generation framework which is used to execute load on a SPECjEnterprise2010 deployment. For each benchmark run one can specify the number of dealer clients that interact with the Orders domain (benchmark scale). The official driver is implemented in a way that the benchmark scale not only influences the total number of clients but also the behavior of a single simulated client. As this is not typical for an enterprise application, the benchmark driver is patched so that the behavior of a dealer client is now independent of the total number of clients.

### *8.3.2 System Topology*

An overview of the system topology is given in figure 8.3. The initial server, on which a resource profile represented as a system model is created, is an IBM System X3755M3 server. This machine, hereafter referred to as AMD-based server, contains 256 gigabytes (GB) random-access memory (RAM) and four AMD Opteron 6172 processors with four cores and a 2.1 GHz frequency each. The target server, which is represented by adapting the model as explained in section 8.2.4, is an IBM System X3550M3 server. This machine, hereafter referred to as Intel-based server, contains 96 GB RAM and two Intel Xeon E5645

[4]http://java.net/projects/faban/

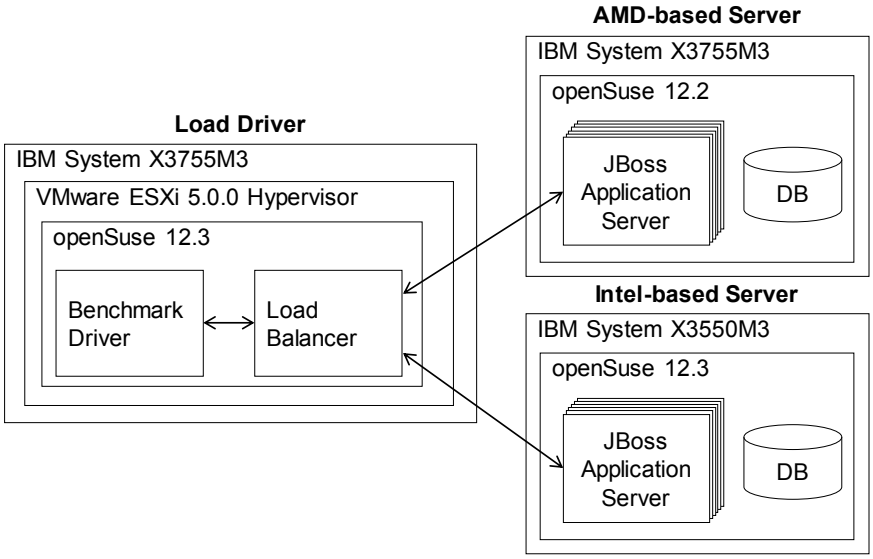


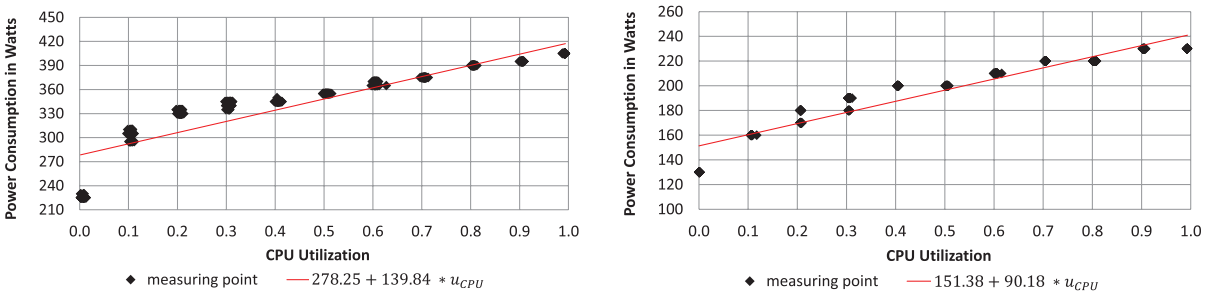

**Figure 8.3:** *SPECjEnterprise2010 system topology*

processors with 6 cores and a 2.4 GHz frequency each. The hyper-threading capability of the Intel processors is disabled for this evaluation. The operating system on the AMD-based server is openSuse 12.2 whereas openSuse 12.3 is used on the Intel-based server. Six JBoss Application Server (AS) 7.1.1 instances are deployed on both servers, all running the SPECjEnterprise2010 benchmark application. Every AS instance uses its own Apache Derby DB in version 10.9.1.0 as persistence layer. The JBoss AS instances and the Apache Derby DBs are executed within a 64 bit Java OpenJDK Server virtual machine (VM) in version 1.7.0. The mod_cluster[5] web server module is used as load balancer for the JBoss AS clusters on a separate VM. The benchmark driver which generates the load on the systems under test (SUT) is also deployed on the same VM. The VM is mapped on the same type of hardware server as the AMD-based server explained above using the VMware ESXi 5.0.0 (build 469512) hypervisor. The VM runs openSuse 12.3 as operating system and is configured to have eight virtual CPU cores and 80 GB RAM. All servers are connected using a one gigabit per second (Gbit/s) network connection.

### *8.3.3 Creating & Adapting the Resource Profile*

To construct a system model of the SPECjEnterprise2010 benchmark application on the AMD-based server, we use an automatic performance model generation approach presented in our previous work (Brunnert/Vögele/Krcmar, 2013). The PCM model generator is configured to represent the CPU resource demand in ms in the generated repository model components used within the system. A moderate load (∼50 % CPU utilization) is generated using the benchmark driver for a period of 20 minutes. During this time, data is collected and afterwards used to generate a system model for the SPECjEnterprise2010 benchmark application. Details of this process can be found in Brunnert/Vögele/Krcmar (2013). Afterwards, the system model is complemented with a usage model, which represents the workload generated by the benchmark driver, and a resource environment model that represents the hardware environment of the AMD-based server.

[5]http://www.jboss.org/mod_cluster




(a) AMD-based server (b) Intel-based server

**Figure 8.4:** *Power consumption models*

As outlined in section 8.2.4, the RDSEFFs of repository model components used within the system model contain CPU demand values specific for the AMD-based server. To adapt this information for the Intel-based server, the processing rate of CPU cores represented in the resource environment model needs to be changed. Additionally, the number of CPU cores needs to be reduced. SPEC CPU2006 is used to benchmark the CPU cores of both servers. The benchmark consists of an integer (SPECint) and a floating point (SPECfp) benchmark which again consist of several sub-benchmarks (Henning, 2006). To calculate the adapted processing rate, the SPEC CPU2006 integer benchmark is executed on the AMD- and Intel-based servers. The AMD-based server achieved a benchmark score of 12.91 for the SPECint_base2006 metric. The Intel-based server achieved a benchmark score of 18.92. By using equation 8.1, a processing rate of 1464 is calculated for CPU cores in the adapted resource environment model of the Intel-based server.

In a next step, *power consumption models* are added to the resource environment models of the AMD- and Intel-based servers, as explained in section 8.2.5. As these models only depict the CPUs of both servers, their *power consumption models* can only model the dependency between the CPU utilization rate and the power consumption. This constraint is acceptable as Capra et al. (2010) state that the CPU consumes the majority of the overall power of a server and the power consumption is significantly dependent on the CPU's utilization rate. All other hardware resources consume roughly the same amount of power independent of their utilization rate (Capra et al., 2010).

To calibrate the *power consumption models*, the command line tool lookbusy[6] is used to stress the CPU. Using lookbusy the hardware resources CPU, HDD and memory can be utilized with a fixed utilization rate. To stress the CPU, lookbusy generates a consecutive CPU utilization in steps of 10 % starting from 0 % to 100 %. Each utilization step lasts for five minutes. After each step the server is kept idle for two minutes. A self written Java tool meanwhile captures the CPU utilization rate and the full server power consumption using IPMI interfaces provided by the servers every second. The IPMI power sensors used in this evaluation showed some ramp-up and ramp-down effects in every utilization step. To avoid these effects, only measurements taken between a ramp-up phase of two minutes and a ramp-down phase of another two minutes are used. This dataset is used to perform a linear regression to construct a *power consumption model* in the form of equation 8.2. The measurements and the thereof derived *power consumption models* for the AMD- and Intel-based servers are shown in figure 8.4(a) and figure 8.4(b).

[6]http://www.devin.com/lookbusy/

The system model of the SPECjEnterprise benchmark application, complemented with a usage model describing the benchmark driver workload and a resource environment model representing one of the two hardware environments, is hereafter called AMD- or Intel-based model respectively.

### *8.3.4 Comparing Measurements & Simulations*

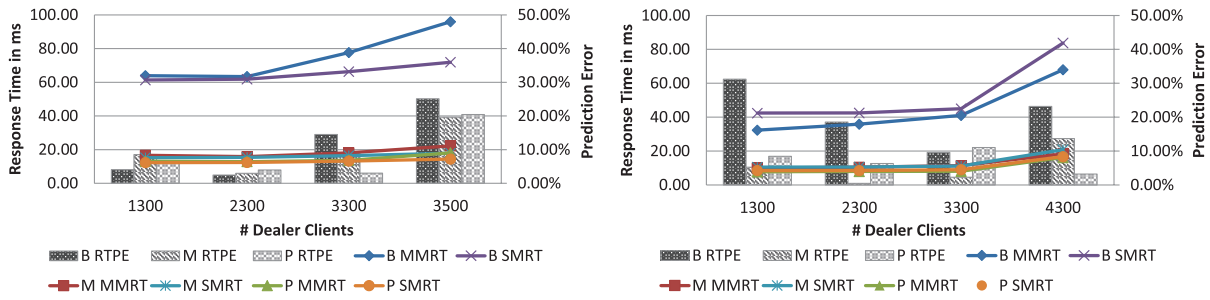
In this section, the simulation results of the AMD- and Intel-based models are compared with measurements on the corresponding servers. The comparison is conducted for different load conditions. Four benchmark runs are executed on both SPECjEnterprise2010 server deployments with varying amounts of dealer clients. The number of clients is increased in steps of 1000 from 1300 to 4300. Benchmark runs on the AMD-based server showed that the system cannot handle 4300 concurrent dealer clients. In a benchmark run with 3500 clients the AMD-based server shows a CPU utilization of approximately 86 % which is equal to the utilization of the Intel-based server in a run with 4300 clients. Therefore, the highest load level is reduced from 4300 to 3500 dealer clients for the AMD-based server.

For each load level the AMD- and Intel-based models are used to predict performance and energy consumption for the respective amount of dealer clients. To simulate such high amounts of dealer clients, the event-oriented simulation engine EventSim (Merkle/Henß, 2011) is used instead of the default process-oriented simulation engine SimuCom (Becker, 2008). EventSim performs better under such load conditions.

For every load level a benchmark and simulation run of 30 minutes is executed. To avoid side effects during the benchmark and simulation runs, only results during a steady state between a five minute ramp-up and a five minute ramp-down phase are considered in the following.

To evaluate the accuracy of the simulation results, the measured and simulated results for each of the following metrics are compared: CPU utilization, power consumption as well as response time and throughput of the three business transactions. To measure the CPU utilization and the power consumption, the same Java tool is used as for the construction of the *power consumption models* in section 8.3.3.

The benchmark driver reports measurements for throughput and response time of the three business transactions performed by the dealer clients. However, the reported response times cannot be used for this evaluation since they contain the network overhead between the driver and the SUT whereas the generated model does not include this information. The simulated and measured response times can thus not be compared with each other. To measure comparable response times, a Servlet filter is used to log the response time of each HTTP request executed during a benchmark run. The mean response times of these HTTP requests are used to calculate the mean response times of the three business transactions. The measurement has an influence on the CPU utilization. To reduce this distortion, the Servlet filter is only deployed on one of the six application server instances. The throughput is taken directly from the benchmark driver. It is included in this comparison as using different thread pool configurations could lead to good response time measurements while




(a) AMD-based server (b) Intel-based server

**Figure 8.5:** *Measured and simulated response times*

the throughput is very low. Therefore, the throughput comparison is important to ensure the validity of the evaluation results.

Table 8.2 and figure 8.5(a) show the comparison of the measured and simulated results for the AMD-based server and model. Table 8.2 shows for every load level specified by the number of dealer clients (C) and for every business transaction (T) the Measured Throughput (MT), the Simulated Throughput (ST), the Throughput Prediction Error (TPE), the Measured Mean CPU Utilization (MMCPU), the Simulated Mean CPU Utilization (SMCPU), the CPU Prediction Error (CPUPE), the Measured Mean Power Consumption (MMPC), the Simulated Mean Power Consumption (SMPC) and the Power Consumption Prediction Error (PCPE). Figure 8.5(a) shows the Measured Mean Response Time (MMRT), the Simulated Mean Response Time (SMRT) and the Response Time Prediction Error (RTPE). Table 8.3 and figure 8.5(b) show the same measurement and simulation results for the Intel-based server and model.

The AMD-based model predicts the response times of the business transactions for low (1300 clients) and medium (2300 clients) load conditions with an error below 10 %. Under high load conditions (3300/3500 clients) the error stays below 26 %. The response time prediction of the Intel-based model is slightly less accurate. The response times are mostly predicted with an error below 20 %. Only the prediction of the browse transaction in the case of 1300 clients and at the highest load level with 4300 clients show deviations of 31.23 % and 23.19 % respectively compared to the measurement results.

Both models predict the throughput with an error below 1 %. This low error is caused by the fact that the average think time of a dealer client between two transactions with approximately 9.9 seconds is much higher than the response time of a single transaction. Thus, the prediction errors of the response times only have a low impact on the prediction accuracy of the throughput. The CPU utilization is mostly predicted with an error below 10 % by both PCM models. Only the CPU utilization prediction for the Intel-based server in a setting with 1300 clients has a higher error of 13.26 %. The power consumption of both servers is predicted with an error below 13 %. As the power consumption values are relatively stable during the steady state of all load levels, the energy consumption can be predicted by multiplying the mean power consumption values by time.

This evaluation shows that energy consumption and performance of two deployments of the same enterprise application can be predicted with an accuracy that is acceptable for capacity planning purposes (Menascé et al., 2004). The approaches to adapt resource

| C | T | MT | ST | TPE | MMCPU | SMCPU | CPUPE | MMPC | SMPC | PCPE |
|---|---|---|---|---|---|---|---|---|---|---|
| **1300** | B | 78623 | 78557 | 0.08 % | 33.37 % | 30.04 % | 9.97 % | 367.55 W | 320.26 W | 12.87 % |
| | M | 39378 | 39245 | 0.34 % | | | | | | |
| | P | 39259 | 39135 | 0.32 % | | | | | | |
| **2300** | B | 139328 | 138383 | 0.68 % | 57.38 % | 52.89 % | 7.82 % | 403.87 W | 352.22 W | 12.79 % |
| | M | 69685 | 70186 | 0.72 % | | | | | | |
| | P | 69932 | 69514 | 0.60 % | | | | | | |
| **3300** | B | 200174 | 199166 | 0.50 % | 82.53 % | 76.00 % | 7.92 % | 433.76 W | 384.52 W | 11.35 % |
| | M | 99643 | 99930 | 0.29 % | | | | | | |
| | P | 99673 | 99315 | 0.36 % | | | | | | |
| **3500** | B | 211585 | 211314 | 0.13 % | 86.10 % | 80.59 % | 6.40 % | 436.47 W | 390.95 W | 10.43 % |
| | M | 105454 | 105506 | 0.05 % | | | | | | |
| | P | 105708 | 105557 | 0.14 % | | | | | | |

**Table 8.2:** *Measured and simulated results for the AMD-based server*

| C | T | MT | ST | TPE | MMCPU | SMCPU | CPUPE | MMPC | SMPC | PCPE |
|---|---|---|---|---|---|---|---|---|---|---|
| **1300** | B | 78502 | 78333 | 0.22 % | 24.05 % | 27.24 % | 13.26 % | 197.05 W | 175.94 W | 10.71 % |
| | M | 39464 | 39376 | 0.22 % | | | | | | |
| | P | 39367 | 39297 | 0.18 % | | | | | | |
| **2300** | B | 139603 | 138961 | 0.46 % | 45.08 % | 48.29 % | 7.12 % | 220.47 W | 194.93 W | 11.58 % |
| | M | 69314 | 69490 | 0.25 % | | | | | | |
| | P | 69266 | 69576 | 0.45 % | | | | | | |
| **3300** | B | 199517 | 199646 | 0.06 % | 64.86 % | 69.34 % | 6.92 % | 241.67 W | 213.91 W | 11.49 % |
| | M | 99254 | 99936 | 0.69 % | | | | | | |
| | P | 99890 | 99355 | 0.54 % | | | | | | |
| **4300** | B | 259548 | 259591 | 0.02 % | 86.03 % | 90.16 % | 4.80 % | 264.29 W | 232.69 W | 11.96 % |
| | M | 130239 | 129641 | 0.46 % | | | | | | |
| | P | 129909 | 129293 | 0.47 % | | | | | | |

**Table 8.3:** *Measured and simulated results for the Intel-based server*

profiles to different hardware environments and to predict energy consumption using an extended PCM meta-model could thus be validated. It is therefore technically feasible to realize the resource profile concept.

## 8.4 Related Work

The resource profile concept relates to several existing research directions. This section is therefore structured according to different directions that contribute to our work. First, we review existing approaches to support capacity planning for enterprise applications using performance models. Afterwards, research in the area of energy consumption of enterprise applications is presented. Finally, approaches that predict energy consumption and performance of enterprise applications are outlined. The review of related work concludes with approaches to improve the relationships of EAVs, EAUs and EAHs using resource demand data.

### Capacity planning using performance models

A model-driven capacity planning tool suite for component- and web service-based applications is proposed by Zhu et al. (2007). The tool suite can be used in early software design phases to support the performance evaluation. It consists of tools to transform existing design models into performance models and benchmark drivers to derive resource demands

for the performance models. These performance models and benchmarks can then be used to support capacity planning tasks. Their tooling is intended to be used early in the development process and not as a final capacity planning tooling, as the implementation might have different characteristics as the generated benchmark code.

Liu/Kumaran/Luo (2001) show how LQN models can be used to support the capacity sizing for EJB applications. In a later work, Liu/Shen/Kumaran (2004) use LQN models to realize a capacity sizing tool for a business process integration middleware by taking different CPU configurations into account. The authors introduce a way to deal with different hardware environments in the context of LQN models. They implemented a model transformation tool which dynamically constructs LQN models from XML documents representing the application model on the one hand and the hardware configuration on the other hand. However, in their current implementation one is limited to only change the processing speed of the CPU of a server and one is not able to change the hardware environment more radically, for example from a one server deployment to multiple servers.

Tiwari/Nair (2010) are using LQNs to predict the performance of two deployments of the same Java EE sample application. Similar to the approach in this work, they show how the SPEC CPU benchmark can be used to adapt a LQN model to a different hardware environment. However, as already discussed in section 8.2.3, LQN models do not allow to change the workload or the hardware environment without reconstructing the whole model. Their approach is thus less flexible than the one proposed in this work.

**Energy consumption of enterprise applications**

Capra et al. (2010) developed energy benchmarks for enterprise resource planning (ERP), customer relationship management (CRM) and database management system (DBMS) applications. They showed that under the same workload different applications with similar functionality have a significant divergent energy consumption. They concluded that energy efficiency is a quality metric that should be considered when buying or developing new software.

An overview of existing power models and metrics to describe the energy consumption of computer systems can be found in the work of Rivoire et al. (2007). The authors show several metrics that can be used to model the energy consumption of computer systems. This work is thus not focused on the energy consumption of a specific enterprise application but rather on the system level.

Johann et al. (2012) propose methods to measure the energy efficiency of software. The authors define energy efficiency as the ratio of useful work done relative to the energy required for performing the work. The authors suggest to calculate the energy efficiency of single methods or components throughout the software development process to create energy efficient applications. However, even though this is a very promising research direction it is challenging to really measure the efficiency if the system on which an application will be deployed has a different power profile than the one on which the application is developed.

Jwo et al. (2011) propose an energy consumption model for enterprise applications. The authors calculate the overall energy consumption by multiplying the time a transaction spends on a machine with the machine's mean power consumption. As the time spend on a machine and its power consumption is workload dependent, the resource profile approach is more flexible as it allows to include this perspective.

#### Combination of performance and energy prediction

One of the few examples that combines energy consumption and performance prediction approaches with a business perspective can be found in the work of Li/Casale/Ellahi (2010). The authors propose a sizing methodology for ERP systems which is based on closed queuing networks. Their methodology allows to optimize sizing decisions using multiple dimensions. They allow to perform Total Cost of Ownership (TCO) decisions that include hardware purchasing as well as energy consumption costs for new ERP systems. Unfortunately, their approach is limited to ERP systems with a predefined set of deployment options and is thus not transferable to other types of applications. However, their multi objective optimization (MOO) approach might be an interesting enhancement for the resource profile concept introduced in this work as resource profiles could be used as the input for such a MOO solver.

#### Relationships between EAV, EAU and EAH

The term resource profile was already used by Brandl/Bichler/Ströbel (2007) in their work on cost accounting for shared IT infrastructures. The authors introduce an approach to associate resource demands to specific IT services (e.g. email) and to store these resource demands for different user types in service-specific vectors (called resource profiles). Using these resource demand vectors they propose an approach to exactly bill the service consumers by the number of users and types of services they are using. Compared to the approach presented in this work, their resource profile concept is mainly intended to be used to allocate costs for existing applications and services more precisely. The approach presented in this work is intended for new applications and services that should be integrated into a data center. However, the data in our resource profile can also be used for the cost accounting approach presented by Brandl/Bichler/Ströbel (2007).

### 8.5 Conclusion & Future Work

This work introduced the concept of resource profiles for enterprise applications. Their main purpose is to simplify the integration of new enterprise applications into data centers with given performance requirements. To achieve this goal, resource profiles support the capacity planning process by allowing to evaluate energy consumption and performance for different workloads and hardware environments before an enterprise application is deployed.

The required information to specify a resource profile and the input parameters for the evaluation can be provided by different parties such as EAVs, EAUs and EAHs. This is achieved by leveraging PCM as meta-model to represent resource profiles. This meta-model

is enhanced to allow predictions of the energy consumption. Additionally, an approach is presented to adapt these enhanced PCM models to different hardware environments. The evaluation showed that a resource profile for the SPECjEnterprise2010 benchmark application predicts energy consumption and performance for two hardware environments with high accuracy.

Additional case studies are required to validate further use cases outlined in section 8.2.2. To make resource profiles better applicable in different scenarios, several extensions are required. First of all, the representation of external systems (e.g. CRM or ERP systems) reused by an enterprise application needs to be investigated. As resource profiles are intended to describe a specific enterprise application, approaches to represent external systems as black-box components which can be replaced by EAUs or EAHs need to be introduced.

Enterprise applications are often executed in runtime environments (e.g. Java EE servers) that are available by different vendors. It would be helpful for the capacity planning if these platforms were represented explicitly in a resource profile and could also be changed by EAUs or EAHs.

Further extensions are required in the underlying PCM meta-model to support the representation of varying workloads and to represent memory demands. The meta-model should also be enhanced to allow representations of the power consumption using nonlinear models. Additionally, the simulation engine EventSim needs to be enhanced to support more elements of the PCM meta-model.

Another area of future research is better automation to create resource profiles. Our evaluation shows how a resource profile can be created and adapted for Java EE applications. Other enterprise application frameworks need similar automation. Even though the underlying performance models could be created using different means such as static or dynamic analysis, our future research will focus on better dynamic analysis as these approaches tend to generate more accurate models from a resource demand perspective.

# Chapter 9

# Detecting Performance Change in Enterprise Application Versions Using Resource Profiles

| | |
|---|---|
| Authors | Brunnert, Andreas[1] (brunnert@fortiss.org)<br>Krcmar, Helmut[2] (krcmar@in.tum.de)<br><br>[1]fortiss GmbH, Guerickestraße 25, 80805 München, Germany<br>[2]Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany |
| Outlet | 8th International Conference on Performance Evaluation Methodologies and Tools (ValueTools, 2014) |
| Status | Accepted |
| Contribution of first author | Problem and scope definition, construction of the conceptual approach, prototype development, experiment design, execution and result analysis, paper writing, paper editing |

**Table 9.1:** *Fact sheet publication P6*

**Abstract** Performance characteristics (i.e., response time, throughput, resource utilization) of enterprise applications change for each version due to feature additions, bug fixes or configuration changes. Therefore, performance needs to be continuously evaluated to detect performance changes (i.e., improvements or regressions). This work proposes a performance change detection process by creating and versioning resource profiles for each application version that is being built. Resource profiles are models that describe the resource demand per transaction for each component of an enterprise application and their control flow. Combined with workload and hardware environment models, resource profiles can be used to predict performance. Performance changes can be identified by comparing the performance metrics resulting from predictions of different resource profile versions (e.g., by observing an increase or decrease of response time). The source of changes in the resulting performance metrics can be identified by comparing the profiles of different application versions. We propose and evaluate an integration of these capabilities into a deployment pipeline of a continuous delivery process.

## 9.1 Introduction

Performance characteristics of enterprise applications change whenever new features or fixes are introduced during software development (Humble/Farley, 2010). Evaluating the performance impact of such changes nowadays requires a performance test which consumes a lot of time and resources. Due to the associated effort and cost, performance tests are often not executed for each version. To improve the feedback cycle during software development, we propose a performance change detection process for each application version that is being built.

This work proposes the use of resource profiles to realize the performance change detection process. The term resource profile is used to describe the resource demand per transaction of an enterprise application version (Brandl/Bichler/Ströbel, 2007; King, 2004). It typically includes central processing unit (CPU) usage, disk IO traffic, memory consumption, and network bandwidth for each component of a software system (Brandl/Bichler/Ströbel, 2007; King, 2004). In Brunnert/Wischer/Krcmar (2014) we introduced a concept to use sub-models of an architecture-level performance meta-model as resource profiles. These model-based resource profiles provide a better separation of transaction resource demands from the workload and hardware environment compared to the traditional way of specifying resource profiles using vectors (Brandl/Bichler/Ströbel, 2007). They specify the resource demand of different transactions, but also the components involved in the transaction processing as well as their control flow. It is furthermore possible to predict performance (i.e., response time, resource utilization, and throughput) using these model-based resource profiles by extending them with workload and hardware environment models (Koziolek, 2010).

Following the definition of Cherkasova et al. (2009), performance changes are defined as an increase or decrease of transaction processing time. Using predictions of different resource profile versions for the same workload and hardware environment model(s), performance changes can be identified by comparing the resulting response time prediction results. If a change is detected, resource profiles can be compared with each other to support the identification of the root cause. To realize such a performance change detection process, a resource profile version needs to exist for each enterprise application version that is being built. Due to this reason, we propose to integrate this change detection process into a deployment pipeline of a continuous delivery process as outlined in the next section.

## 9.2 Detecting Performance Change within a Deployment Pipeline

Continuous integration (CI) systems integrate independently developed code provided by different teams and build the overall system composed of all components involved. CI systems help to keep the development teams in sync and to create deployable artifacts automatically. CI systems such as Jenkins[1] are very popular nowadays as code changes that break the build or cause problems in automated unit tests can be identified immediately.

[1] http://jenkins-ci.org/

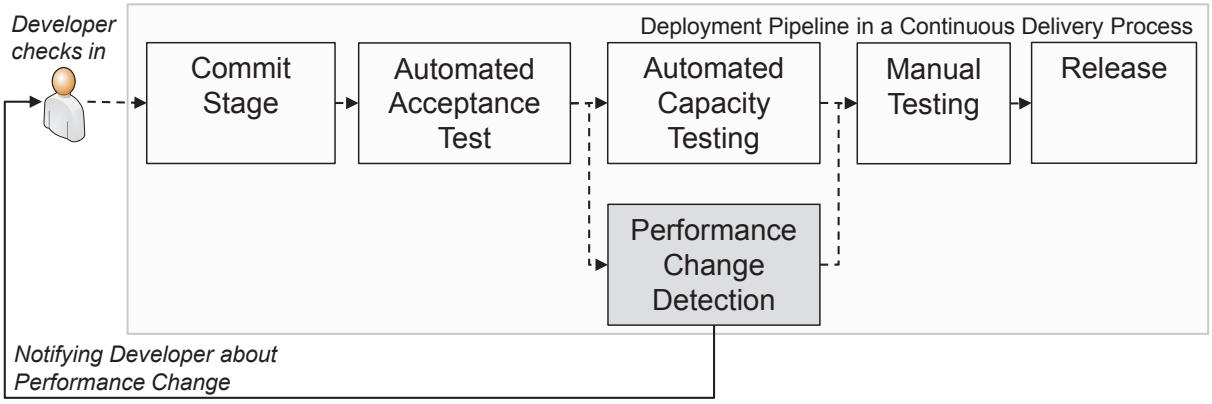



**Figure 9.1:** *Detecting performance change within a deployment pipeline (adapted from Humble/Farley (2010))*

This immediate feedback of potential problems increases the awareness of everyone involved in the development process regarding the impact of a change on the overall system.

As compiling and building a software system is only the first step to get a system in a state so that it can be used by its end users, a new discipline called continuous delivery (CD) emerged in recent years (Humble/Farley, 2010). CD is defined by Humble/Farley (2010) as an extension of CI principles to the "last mile" to operations. A key element in their definition of CD is a so called deployment pipeline. A deployment pipeline describes the steps it takes from a deployable version until a program version exists that can be used as a release candidate.

Figure 9.1 depicts an extended version of the deployment pipeline defined in Humble/Farley (2010). As a first step in this deployment pipeline, a build is triggered by one or multiple developer check-ins. These check-ins are used by a CI system in a so called commit stage to build a new version of an application and to execute a set of predefined tests. These tests focus on low-level application programming interface (API) tests from a developer perspective. The next stage, called automated acceptance test, evaluates if an application that is being built really delivers value to a customer. A set of regression tests is being executed to evaluate the functionality from an end-user perspective. Humble/Farley (2010) suggest that afterwards an automated capacity test should occur. If the capacity test results are acceptable, the new version is being tested manually before it is released as candidate for production deployment. It is important to note, that continuous delivery does not imply continuous deployment, and does not automatically deploy a release candidate to production.

This work proposes a performance change detection process using resource profiles as an alternative to the capacity testing step within such a deployment pipeline (see figure 9.1). Such an integration not only helps to ensure that a resource profile is created for every build but also to identify the cause of a performance change. As the source code changes included in a specific build are known, they can be specifically analyzed whenever a performance change is detected. The steps of the performance change detection process are outlined in the following section.

### *9.2.1 Performance Change Detection*

In their description of the capacity tests within the deployment pipeline, Humble/Farley (2010) define capacity as the maximum throughput a system can achieve with given response time thresholds for different transaction types. This definition implies that a system needs to be available during the automated capacity testing step which is comparable to the final production environment. Otherwise, results from capacity tests according to their capacity definition would not yield meaningful results. This precondition is often not given in development projects (Brunnert et al., 2014), because representative test environments are shared between projects to reduce cost. The immediate feedback that is promised by CD systems can, thus, often not be guaranteed using this approach, as only smaller scale systems are available. The capacity test results derived from such systems are hardly comparable to a production environment.

Using resource profiles, performance can be predicted for hardware environments with more resources (e.g., CPUs) than the ones that are available (Brunnert/Wischer/Krcmar, 2014). Furthermore, resource profiles can be adapted to a hardware environment which is more comparable to the production environment using an approach presented in Brunnert/Wischer/Krcmar (2014). These abilities speed up the feedback loop and allow for performance evaluations that would be otherwise impossible. Real capacity tests can then be executed later in the process or as a manual test step, when appropriate systems are available.

Detecting performance change using resource profiles requires the following steps: In a first step, a resource profile for the current application version needs to be created. This resource profile version must be put into a versioning repository to make it available for subsequent builds. Afterwards, performance is predicted using the current version and the results are compared with prediction results from a previous version to see if a performance change occurred. If a change is detected, a notification is sent to the development team(s). The next sections explain how these steps are realized and integrated into a continuous delivery process.

### *9.2.2 Creating Resource Profiles*

We propose the use of dynamic analysis to collect measurements during the automated acceptance test execution to create resource profiles. This approach has several advantages, apart from the fact that it saves time compared to a separate measurement run (according to Humble/Farley (2010), acceptance tests usually take several hours to complete): Transactions that are being tested and executed in the acceptance test stage are expected to be close to the behavior of the users (Humble/Farley, 2010). Using measurements from the acceptance tests also ensures that the workload stays relatively stable as the regression tests are executed for every build. The measurement results for each build are, therefore, comparable.

As mentioned in the introduction, a model-based resource profile of an enterprise application is represented using architecture-level performance models (Brunnert/Wischer/

Krcmar, 2014). We use the Palladio Component Model (PCM) as meta-model to represent resource profiles (Becker/Koziolek/Reussner, 2009). The PCM meta-model represents the performance-relevant aspects of a software system separately from the workload and the hardware environment. Performance-relevant aspects of a software system are represented in a so called repository model. This model contains components of a software system and their relationships. The control flow of a component operation, its resource demand and parametric dependencies are also specified in this model. Components are assembled in a system model to represent an application. We use the system model to group components by the deployment units they belong to, to simplify their use (Brunnert/Wischer/Krcmar, 2014). The workload on a system is described in a usage model. The remaining two model types in PCM describe the hardware environment: A resource environment model allows to specify available resource containers (i.e., servers) with their associated hardware resources (e.g., CPU or HDD). An allocation model specifies the mapping of system model elements on resource containers. A resource profile for an enterprise application can thus be represented by a repository together with a system model (Brunnert/Wischer/Krcmar, 2014).

Several ways to generate these two PCM model types either based on static (Becker, 2008) or dynamic analysis (Brunnert/Vögele/Krcmar, 2013) exist. We do not explain a specific model generation approach in this section to focus on the conceptual use of resource profiles for the purpose of detecting performance change[2]. However, for the purpose of identifying performance change, it is only feasible to use approaches that use measurement data from dynamic analysis. Otherwise, the required resource demand values for representing a resource profile would be missing (Spinner et al., 2014).

### *9.2.3 Versioning Resource Profiles*

Each reusable artifact that is created within the deployment pipeline is stored in a so called artifact repository (Humble/Farley, 2010). This is necessary to make each artifact available in different steps of the deployment pipeline. To detect performance change, we do not only need the resource profile of the current version but also the one of previous builds. Therefore, each resource profile version that is created in the deployment pipeline is stored in an artifact repository that allows to manage different resource profile versions.

As PCM is based on the Eclipse Modeling Framework (EMF)[3], all performance models conform not only to the PCM meta-model but also to the Ecore meta-model defined by EMF. We are leveraging this capability by using the EMFStore (Koegel/Helming, 2010), which already implements the required versioning features for models based on the Ecore meta-model.

The advantage of using EMFStore compared to other versioning systems is that it is especially designed to support the semantic versioning of models (Koegel/Helming, 2010). Instead of working with textual representations of the models in existing systems, EMFStore uses the Ecore model elements and their relationships to manage models stored in the

[2] An exemplary approach for generating resource profiles for Java Enterprise Edition (EE) applications is used in the evaluation section to validate this process.

[3] http://www.eclipse.org/modeling/emf/

repository. For example, instead of representing a structural change between two model versions as multiple lines in their textual representation, EMFStore directly stores the change in the Ecore model itself (Koegel/Helming, 2010). The only two PCM model layers that are currently stored and versioned in the EMFStore automatically are the repository and system models.

### *9.2.4 Predicting Performance*

Using PCM-based resource profiles, performance predictions can be made for different workload and hardware environment models. These predictions include results for the performance metrics response time, throughput and resource utilization. As performance change is defined as changes in the transaction processing time, response times in the prediction results are used as an indicator for change.

To enable comparable predictions in the deployment pipeline, workload and hardware environment models for performance predictions need to be statically defined. As explained in the previous section, the workload on a system can be represented using usage models. The hardware environment is represented using resource environment and allocation models. The artifact repository therefore needs to contain usage models as representation of different workloads, which use the external interfaces provided by the system model of a resource profile to specify the load. It furthermore needs to contain one or more resource environment models specifying the available servers and corresponding allocation models which define the mapping of deployment units specified in a resource profile to the servers.

Before performance can be predicted, a lookup needs to be made in the artifact repository to get the corresponding workload and hardware environment models. The workload and hardware environment models are combined with a resource profile to predict performance. If a hardware environment should be used with different hardware components than the one the resource profile was created on, the processing rates of the hardware resources need to be specified relative to the original ones. One approach to do that is to use benchmark results for the source and target hardware as shown in Brunnert/Wischer/Krcmar (2014).

### *9.2.5 Comparing Prediction Results*

To detect performance change, performance is predicted with the current resource profile version and a specified set of hardware environment and workload models. If a previous resource profile version for the same application is available, the same predictions are executed using the previous version. Afterwards, the prediction results are compared. The reason for executing the predictions with the previous resource profile version again is to ensure that the same workloads and hardware environments are used. This avoids situations in which changes in the workload or hardware environment models lead to incorrect results.

For each transaction, the relative predicted response time change between the current and the previous application version is calculated. As a prediction results in multiple response

time values over time, this set of transaction response time values can be represented in multiple ways (Jain, 1991). The most common ones are mean values including distances from the mean (e.g., standard deviation) as well as percentiles (e.g., 50th (median) or 90th percentile). Which one of these values represents a response time set best depends on the dispersion of the underlying distribution (Jain, 1991). The dispersion of the underlying distribution can change between the response time sets collected for the same transaction in different versions. To account for this fact while making the results comparable between versions, the relative change is always calculated for mean and median values as well as for 90th percentiles.

We propose to calculate the relative change ($rc$) as shown in equation 9.1. In this equation, the transaction response time ($rt$) predictions of the *current* and *previous* resource profile versions are compared with each other. The resulting change value indicates whether the response time is now higher than before if the value is positive or lower if the value is negative.

$$rc = \frac{rt_{current} - rt_{previous}}{rt_{previous}} \tag{9.1}$$

If the relative change for at least one transaction is higher than a specified threshold (e.g., above 20 % increase or decrease) a notification (e.g., as email from the build system) is being sent to the developer(s). If response time increased above a specified threshold the deployment pipeline needs to be stopped. It is important to stop the pipeline in this case, so that the application version is not marked as stable. This ensures, that the next build will be compared with a valid resource profile of a version with meaningful performance.

To enable a distinction between runs with and without changes, the corresponding resource profile versions need to be managed independently within the EMFStore and a resource profile should only be marked as usable for subsequent builds if the relative change is below the specified threshold.

To identify trends, a comparison can also be made against the resource profiles of more than one of the last successful builds. This avoids situations in which performance regressions develop slowly across multiple builds.

### *9.2.6 Comparing Resource Profiles*

Users can access, analyze and edit models in the EMFStore using a plugin for the PCM modeling environment[4]. Due to this reason, resource profile versions in the artifact repository are directly accessible to developers when they are notified about a performance change. The notification could also include a link to the corresponding resource profile versions. Each resource profile in the EMFStore can be analyzed over time to see which components, relationships or resource demands are associated with specific versions stored in the repository.

[4]http://www.palladio-simulator.com/

As resource profiles for normal application versions and for versions with performance changes are managed independently from each other in the EMFStore, one cannot easily analyze the differences between these versions. For this purpose, one can compare the different resource profile versions using the EMF Compare framework[5]. EMF Compare allows to automatically analyze and visualize the differences between models conforming to the Ecore meta-model.

The level of available detail depends on the resource profile generation approach, but we assume that the components a system is composed of, their operations and their relationships are represented in the model (Wu/Woodside, 2004), including corresponding resource demands. Therefore, the result of a comparison reveals changes in resource demands, control flows and component operations. Using this information, the developers can identify the sources for a performance change. Combined with the information, which changes have been performed during the check-ins that initiated the build, it should help the developer to identify the cause of a performance change.

## 9.3 Evaluation

This section evaluates the performance change detection process within a deployment pipeline as explained in section 9.2. Section 9.3.1 describes the setup of the build and test system for this evaluation. Afterwards, the evaluation steps using these systems are explained in section 9.3.2.

### *9.3.1 Build and Test System*

The evaluation in this paper uses a SPECjEnterprise2010[6] benchmark application called Orders domain as an exemplary enterprise application. The advantage of using a benchmark application is that the benchmark specifies a workload as well a dataset for test runs so that they can be easily repeated by others. The Orders domain is a Java EE web application that is composed of Servlet, JavaServer Pages (JSP) and Enterprise JavaBean (EJB) components. Users access this application using a web interface over the hypertext transfer protocol (HTTP) and can perform three different business transactions: browse, manage and purchase. These three business transactions are composed of several HTTP requests to the system.

The experiment setup consists of two virtual machines (VM). One VM, called build system, is used to execute the build and test tasks within the deployment pipeline. The other VM, called test system, is used to host the Orders domain deployment. These two virtual machines are mapped to two different hardware servers (IBM System X3755M3). The hardware servers are connected using a one gigabit-per-second network connection. Both

[5] http://www.eclipse.org/emf/compare/

[6] SPECjEnterprise is a trademark of the Standard Performance Evaluation Corp. (SPEC). The SPECjEnterprise2010 results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The official web site for SPECjEnterprise2010 is located at http://www.spec.org/jEnterprise2010.

virtual machines run openSuse 12.3 64bit as operating system and have four virtual CPU cores and 40 gigabytes of random-access memory.

The deployment pipeline on the build system is implemented as projects in the CI system Jenkins. A first project builds the Orders domain. If the build was successful, the new Orders domain version is deployed on the test system in a second project. The application is deployed on a GlassFish Application Server (AS) Open Source Edition 4.0 (build 89) in the Java EE 7.0 full profile. The database on the test system VM is an Apache Derby DB in version 10.9.1.0. The GlassFish AS and the Apache Derby DB are both executed in the same 64 bit Java OpenJDK VM (JVM version 1.7.0).

Once the deployment is completed successfully, a third project executes automated acceptance tests. For that purpose, test scripts are used that are provided by the benchmark. These test scripts describe the user interactions with the Orders domain and are implemented to run within the Faban harness[7]. Faban is a workload creation and execution framework. These tests are no acceptance tests in the traditional sense, but they exercise the system in a way a normal user would and are, thus, comparable.

If the acceptance tests complete successfully, a forth project in the CI system triggers the generation of a resource profile. An automatic performance model generation approach for Java EE applications introduced in Brunnert/Vögele/Krcmar (2013) is used to generate resource profiles for the Orders domain application. This approach uses runtime instrumentation to collect data for the model generation. Resource profiles are generated based on the data collected during the acceptance tests. These profiles are stored and versioned in an EMFStore server running on the build system. The last steps in the deployment pipeline shown in figure 9.1 are not automated.

### *9.3.2 Evaluation Steps*

**1.)** In a first step, the automated steps of the deployment pipeline are executed for the standard version of the Orders domain application.

**2.)** In a second step, the Orders domain application is modified and the automated steps of the deployment pipeline are triggered again. In this version, the performance characteristics of two application components are modified by increasing their resource consumption.

**3.)** Before we continue to identify performance change between both application versions, the prediction accuracy of both resource profiles is evaluated. To do that, prediction results of both resource profiles are compared with measurements of their corresponding application versions deployed on the test system.

**4.)** To identify performance change, we are using the resource profiles of both versions. They are used to predict response times for predefined workloads and one hardware environment. The expected result is, that an increase of response time and thus a

[7] http://java.net/projects/faban/

regression in performance can be observed. To identify the root cause, the resource profile versions are compared directly to see if the change introduced in the second application version is visible in this comparison.

### *9.3.3 Creating and Versioning Resource Profiles*

Once the standard version of the SPECjEnterprise2010 Orders domain application is being built and deployed on the instrumented test system, acceptance tests are executed. These tests are executed with 600 concurrent users for 20 minutes while data is only collected between a five minute ramp up and a five minute ramp down phase. All of the following test runs are executed using the same duration. Once the test is completed, a command line utility is triggered by the build system that implements the approach presented in Brunnert/Vögele/Krcmar (2013) and generates a resource profile for this application version and stores it in the EMFStore.

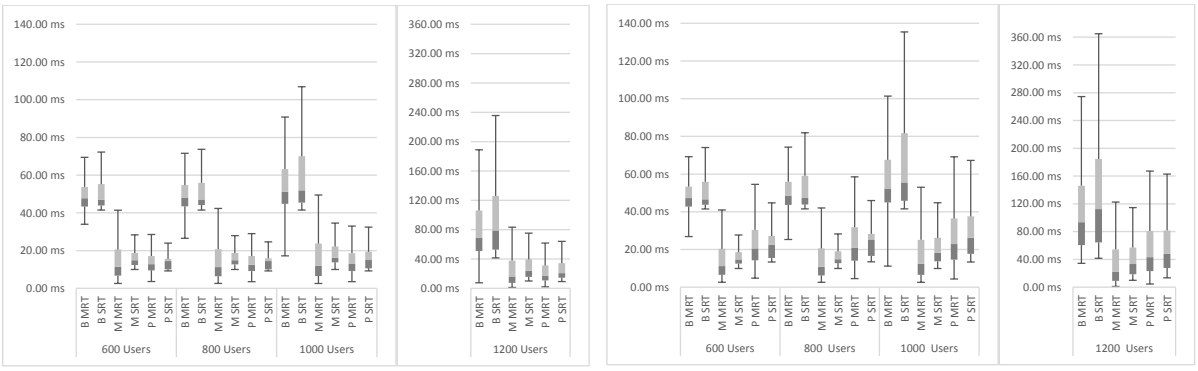
In a second step, an updated version of the Orders domain application is being built and deployed on the test system. The updated version is modified so that two components of the application consume more CPU time than in their original versions. Another test run with 600 concurrent users is then executed using the updated version. Afterwards, a new version of the resource profile is generated and stored in the EMFStore.

The resource profile for the original Orders domain application is hereafter called resource profile version one and the second resource profile for the modified application is hereafter called version two. To evaluate performance for both application versions, usage and hardware environment models are predefined. The usage model is created following the source code of the test scripts in the Faban harness. The hardware environment models represent the test system.

### *9.3.4 Evaluating the Accuracy of Resource Profile Predictions*

To evaluate the prediction accuracy of the generated resource profile versions (V), they are used to predict the performance of the corresponding application versions under different workload conditions. The same workloads are executed as test runs using the corresponding application versions. Afterwards, the measured results are compared with the predicted results. This comparison includes the response time and throughput of the business transactions (T) browse (B), manage (M) and purchase (P) as well as the CPU utilization of the test system.

The workload for this comparison is increased in steps of 200 concurrent users (U) from 600 to 1200 (~47 % to ~90 % CPU utilization). To predict the performance of the application versions, the corresponding resource profiles including the workload and hardware environment models are used as input for the simulation engine SimuCom (Becker/Koziolek/Reussner, 2009). SimuCom performs a model-to-text transformation to generate Java code based on PCM models. This Java code is afterwards executed to start a simulation. The simulation duration is set to 20 minutes while only data between




(a) Application and resource profile version one

(b) Application and resource profile version two

**Figure 9.2:** *Measured and simulated response times*

a five minute ramp up and five minute ramp down phase is used for the calculation of the simulation results.

The simulation results for the resource profile versions one and two are shown in boxplot diagrams[8] in figures 9.2(a), 9.2(b) and in table 9.2. Response time measurement and simulation results are described using mean and median values as well as measures of dispersion, namely the quartiles and the interquartile range (IQR). Variance and standard deviation are excluded from our investigation due to the skewness of the underlying distributions (Jain, 1991) of the response times of browse, manage and purchase. For each load condition specified by the number of users, figures 9.2(a) and 9.2(b) show the dispersion of the simulated response time (SRT) per business transaction. The simulated mean response times (SMRT), the simulated mean CPU utilization (SMCPU) and the simulated throughput (ST) can be found in table 9.2.

In the same way as the simulations, the tests run 20 minutes and the data for this comparison is collected in a steady state between a five minute ramp up and a five minute ramp down phase. The measured mean CPU utilization (MMCPU) of the test system is measured during this time using the system activity reporter. Only the measured throughput (MT) values are directly taken from the Faban harness. As the simulated values do not contain the network overhead between the build and the test system, the response time values of the business transactions are calculated based on measurements performed directly on the test system. For this purpose, the response times of HTTP requests executed by the Orders domain users on the test system are measured during the test execution using a Servlet filter (Brunnert/Vögele/Krcmar, 2013; Mordani, 2007). The response times of the single HTTP requests are then used to calculate the measured response time (MRT) values for the business transactions shown in the boxplot diagrams in figures 9.2(a) and 9.2(b). The measured mean response times (MMRT) are shown in table 9.2.

[8]Boxplot diagrams consist of a box whose bounds denote the first quartile $Q_1$ (lower bound) as well as the third quartile $Q_3$ (upper bound) of the underlying data sample. The quartiles are connected by vertical lines to form the box that indicates the IQR which is defined as $Q_3 - Q_1$. Furthermore, the median $Q_2$ is illustrated by a horizontal line within the box, thus separating it into two parts. Vertical lines outside the box (whiskers) indicate the range of possible outliers while their length is limited to 1.5 times the $IQR$.

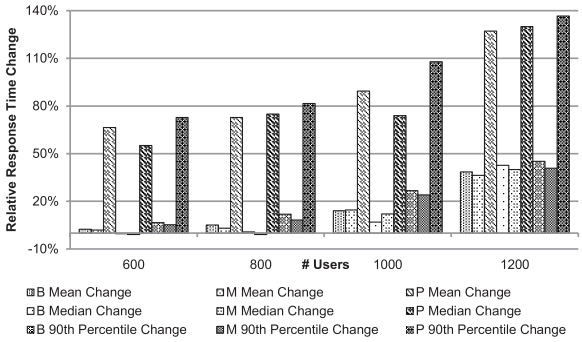
| V | U | T | MMRT | SMRT | RTPE | MT | ST | TPE | MMCPU | SMCPU | CPUPE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 600 | B | 49.95 ms | 53.13 ms | 6.38% | 18,027 | 18,328 | 1.67% | 47.48 % | 43.12 % | 9.18 % |
| | | M | 17.90 ms | 16.51 ms | 7.72% | 8,970 | 8,989 | 0.21% | | | |
| | | P | 14.03 ms | 14.53 ms | 3.56% | 8,946 | 9,275 | 3.68% | | | |
| | 800 | B | 51.82 ms | 54.05 ms | 4.31% | 24,332 | 24,524 | 0.79% | 59.81 % | 57.60 % | 3.70 % |
| | | M | 18.28 ms | 16.93 ms | 7.39% | 11,912 | 12,066 | 1.29% | | | |
| | | P | 14.25 ms | 14.76 ms | 3.60% | 11,972 | 12,194 | 1.85% | | | |
| | 1000 | B | 59.13 ms | 62.91 ms | 6.40% | 30,274 | 30,454 | 0.68% | 71.26 % | 71.76 % | 0.70 % |
| | | M | 21.66 ms | 19.64 ms | 9.33% | 15,081 | 15,281 | 1.33% | | | |
| | | P | 16.19 ms | 17.05 ms | 5.31% | 15,122 | 15,252 | 0.86% | | | |
| | 1200 | B | 88.57 ms | 98.78 ms | 11.53% | 36,389 | 36,497 | 0.30% | 81.17 % | 85.97 % | 5.92 % |
| | | M | 35.45 ms | 30.92 ms | 12.79% | 17,836 | 18,540 | 3.95% | | | |
| | | P | 27.38 ms | 26.97 ms | 1.51% | 18,119 | 17,975 | 0.79% | | | |
| 2 | 600 | B | 49.50 ms | 54.45 ms | 9.99% | 18,196 | 18,442 | 1.35% | 51.65 % | 45.90 % | 11.12 % |
| | | M | 17.80 ms | 16.83 ms | 5.45% | 9,078 | 9,124 | 0.51% | | | |
| | | P | 24.48 ms | 24.20 ms | 1.15% | 9,146 | 9,016 | 1.42% | | | |
| | 800 | B | 52.69 ms | 56.80 ms | 7.81% | 24,308 | 24,332 | 0.10% | 63.51 % | 61.01 % | 3.93 % |
| | | M | 18.41 ms | 17.46 ms | 5.15% | 12,118 | 12,165 | 0.39% | | | |
| | | P | 25.76 ms | 25.49 ms | 1.05% | 12,034 | 12,292 | 2.14% | | | |
| | 1000 | B | 61.85 ms | 71.74 ms | 15.99% | 30,379 | 30,326 | 0.17% | 73.98 % | 76.05 % | 2.81 % |
| | | M | 23.71 ms | 22.49 ms | 5.14% | 14,880 | 15,406 | 3.53% | | | |
| | | P | 29.74 ms | 32.31 ms | 8.62% | 15,021 | 15,186 | 1.10% | | | |
| | 1200 | B | 120.88 ms | 136.82 ms | 13.19% | 36,318 | 36,394 | 0.21% | 86.68 % | 90.98 % | 4.97 % |
| | | M | 49.18 ms | 42.16 ms | 14.27% | 18,010 | 18,106 | 0.53% | | | |
| | | P | 63.24 ms | 61.24 ms | 3.15% | 18,271 | 18,273 | 0.01% | | | |

**Table 9.2:** *Measured and simulated results for resource profile versions one and two*

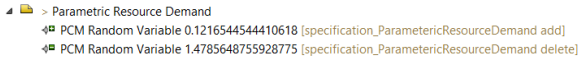
The comparison of the simulated and measured results shows that both resource profiles do represent their corresponding application versions very well. The highest relative response time prediction error (RTPE) for the mean response time values is 15.99 % for the browse transaction and a load of 1000 concurrent users. The median values support this result as the relative prediction error for the median values is at most 21 % for the browse and purchase transactions. For the manage transaction, the relative error of the median values goes up to 52 % for a load of 1200 concurrent users in the second resource profile version.

The skewness of a business transaction's underlying response time distribution can be determined considering the median's position between the quartiles $Q_1$ and $Q_3$. The results show that the skewness of the underlying distribution is not always correctly represented. This is especially the case for manage, as the first quartile $Q_1$ is predicted by the simulation with a relative error of up to 112 %, which is already caused by an absolute error of 7.32 milliseconds (ms). The first quartile $Q_1$ for the browse and purchase transactions is mostly represented with a relative error below 22 %, only for a load of 1200 users, the relative error for the purchase transaction in the predictions with the first resource profile version goes up to 32 %. The third quartile $Q_3$ is predicted with a relative error of at most 26 % for all transactions.

The relative throughput prediction error (TPE) is at most 3.95 % (see table 9.2). This validates the results but is expected, as the think time for each user is much higher (9.8 seconds) than the response time measurement and simulation results shown in figures 9.2(a) and 9.2(b). The impact of response time prediction errors on the throughput is thus very low.




(a) Relative change between both resource profile versions




(b) EMF Compare result

**Figure 9.3:** *Comparison results*

The relative CPU utilization prediction error (CPUPE) is at most 11.12 % (see table 9.2). The simulated CPU utilization is below the measured CPU utilization in low load levels (600 and 800 concurrent users), as the garbage collection overhead and other JVM activities are not represented in the resource profiles (Brunnert/Vögele/Krcmar, 2013; Brunnert/ Wischer/Krcmar, 2014). In high load conditions (1000 and 1200 concurrent users) the simulated values are slightly higher than the measured values. The data collection overhead included in the resource demands of the model elements (Brunnert/Vögele/Krcmar, 2013) thus seems to balance the influence of aspects not represented in the resource profiles in high load conditions.

### *9.3.5 Comparing Prediction Results and Resource Profile Versions*

The prediction results of the two resource profile versions are now compared with each other to identify performance change. As shown in figure 9.3(a), the response times of all transactions increased from version one to two, even though the median response time values of manage show a slight decrease for low load levels (600 and 800 users). It is furthermore visible that the purchase transaction response time increased in all load levels. The mean, median and 90th percentile values for this transaction increased by at least 66 % and up to 137 %. Therefore, the results show a clear regression for the purchase transaction.

Using a plugin in the PCM modeling environment we accessed the EMFStore that contains both resource profile versions and looked at their differences using EMF Compare. A screenshot of one comparison result can be found in figure 9.3(b). This screenshot shows, that the CPU resource demand of one component operation (1.48 ms) is about twelve times higher than the one of the previous version (0.12 ms). A similar result is visible in two components involved in the control flow of requests within the purchase transaction. Comparing the resource profiles, thus, helped to identify the places which caused the performance regression.

## 9.4 Related Work

Mi et al. (2008) and Cherkasova et al. (2008) have already identified the need to evaluate the performance changes in each enterprise application version. For this purpose, the authors propose to create so called application signatures and compare the performance characteristics of each application version using their corresponding signatures. Application signatures are a representation of transaction processing times relative to the resource utilization for a specific workload. The authors extend their approach in Cherkasova et al. (2009) by using performance modeling techniques to not only detect performance changes based on response times but also to evaluate performance anomalies in the CPU demand of transactions. Their work is focused on systems that are already in production. This work extends the idea of evaluating the change in performance of each application version to the development process. Furthermore, we propose the use of resource profiles instead of application signatures. Resource profiles allow for more flexible evaluations as they can be used to derive these metrics for different workloads and hardware environments. It is thus possible to derive more meaningful metrics with smaller systems as it would be possible using application signatures. This is important for performance evaluations during software development, as performance test environments might not be available (Brunnert et al., 2014).

An approach to detect performance regressions of different versions of a software system during development has been proposed by Nguyen et al. (2012). The authors propose the use of control charts to identify whether results of a performance test indicate a regression or not. Their approach focuses on the automatic detection of regressions based on performance test results. It could be used to evaluate prediction results based on resource profiles. However, their approach requires a real performance test and does not support the detection of possible problem causes as is possible by comparing resource profiles. It furthermore assumes a linear relationship between the resulting performance metrics and the load on a system, which might not always be true.

## 9.5 Conclusion and Future Work

Detecting performance change as part of a deployment pipeline provides immediate feedback to developers about the impact of a change on the performance of an overall enterprise application. The performance metrics and their relative change values indicate whether an application is on track to achieve the required performance goals. Compared to a performance evaluation solely at the end of the software development process, the suggested approach allows tracking and improving the performance along with the functionality in the process.

The evaluation results validate the performance change detection process within a deployment pipeline for a Java EE application. However, there are some limitations of the approach that need to be dealt with in future work. As of today, performance change can only be detected when the resource demand type which causes a performance change is also represented in a resource profile. As representing memory is currently not fully

supported by the underlying meta-model, changes caused by different memory consumption characteristics are not detectable yet.

An additional open challenge is that external interfaces of an application can change between enterprise application versions. If one resource profile version breaks the compatibility with the interfaces used by the workload specifications in the artifact repository, the change detection will stop working as long as the usage models are not modified. Even if a corresponding modification is made, the new usage models will no longer be compatible with the previous versions.

Another direction of future work is to automate the complete performance change detection process and to integrate it as a plugin in CI/CD systems. Furthermore, the steps of comparing different resource profile versions and identifying the problem causes should be handled automatically. It might also be interesting to use information about check-ins that are included in a build to perform static analysis to improve the search process for the reasons of a change.

# Chapter 10

# Continuous Performance Evaluation and Capacity Planning Using Resource Profiles for Enterprise Applications

| | |
|---|---|
| Authors | Brunnert, Andreas[1] (brunnert@fortiss.org)<br>Krcmar, Helmut[2] (krcmar@in.tum.de)<br><br>[1]fortiss GmbH, Guerickestrasse 25, 80805 München, Germany<br>[2]Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany |
| Outlet | Journal of Systems and Software (JSS) |
| Status | Accepted |
| Contribution of first author | Problem and scope definition, construction of conceptual approach, prototype development, scenario definition, experiment design, execution and analysis, paper writing, paper editing |

**Table 10.1:** *Fact sheet publication P7*

**Abstract** Continuous delivery (CD) is a software release process that helps to make features and bug fixes rapidly available in new enterprise application (EA) versions. Evaluating the performance of each EA version in a CD process requires a test environment comparable to a production system. Maintaining such systems is labor intensive and expensive. If multiple deployments of the same EA exist, it is often not feasible to maintain test instances for all of these systems. Furthermore, not all deployments are known at the time of a release (e.g., for off-the-shelf products). To address these challenges, this work proposes the use of resource profiles which describe the resource demand per transaction for each component of an EA and allow for performance predictions for different hardware environments and workloads without the need to own corresponding test environments. Within a CD process, resource profiles can be used to detect performance changes in EA versions. Once a version is released, resource profiles can be distributed along with the application binaries to support capacity planning for new deployments. Three integrated experiments for a representative EA provide validation for these capabilities.

## 10.1 Introduction

The need to continuously adapt enterprise applications (EA[1]) to changes in the business environment led to several modifications in the software development process in recent years. A tighter integration between development and operation teams as well as an increased frequency of releases of new EA versions are examples of such changes. The goal of these adjustments is to deliver new features or bug fixes to the users faster than traditional release cycles which release changes in larger batches in a few major versions.

In order to support rapid release cycles, concepts such as continuous delivery (CD) are applied in software development projects nowadays. A key element of CD is a so called deployment pipeline. The deployment pipeline describes the steps taken from creating a deployable EA version until a program version exists that can be used as a release candidate. Figure 10.1 depicts an extended version of the deployment pipeline defined by Humble/Farley (2010).

Because performance characteristics (i.e., response time, resource utilization, and throughput) of an EA change whenever new features or fixes are introduced in new EA versions (Humble/Farley, 2010), their performance needs to be continuously evaluated. For that purpose, Humble/Farley (2010) suggest to include a capacity testing step in a deployment pipeline as shown in figure 10.1. This step assumes that a test environment is available which is comparable to a production system. Maintaining a copy of a production system is an expensive and labor intensive undertaking (Brunnert et al., 2014). As multiple deployments of an EA version can exist, it is often not feasible to test all workloads and hardware environments in the capacity testing step. Furthermore, not all deployments are known at the time of a release (e.g., for off-the-shelf products).

In order to address these challenges, this work proposes extensions to the deployment pipeline to ensure continuous performance evaluation of each EA version. The extensions proposed in this work are realized using resource profiles. Resource profiles are models describing the resource demand per transaction of an EA and are introduced in section 10.3 of this paper.

In section 10.4, we introduce a performance change detection step using resource profiles as an alternative to the capacity testing step in a deployment pipeline (see figure 10.1). The proposed performance change detection approach allows for continuous performance evaluations of all EA versions without the need for expensive performance test environments. Section 10.5 introduces a concept to release the resource profile of an EA version along with the application binaries (packaged as deployment units, see figure 10.1). We outline

[1]The following non-standard abbreviations are used in this article:

| Abbreviation | Meaning | Abbreviation | Meaning | Abbreviation | Meaning |
|---|---|---|---|---|---|
| B | Browse | IC | Initial Cost | RP | Resource Profile |
| BR | Branch | M | Manage | RPC | Relative Power Consumption |
| C | Component | MMCPU | Measured Mean CPU Utilization | RT | Response Time |
| CD | Continuous Delivery | MMPC | Measured Mean Power Consumption | RU | Rack Unit |
| CF | Control Flow | MOO | Multi Objective Optimization | SC | Server Count |
| CPUPE | CPU Prediction Error | MRT | Measured Response Time | SMCPU | Simulated Mean CPU Utilization |
| D | (Resource) Demand | OP | (Component) Operation | SMPC | Simulated Mean Power Consumption |
| DML | Descartes Modeling Language | p | Probability | SRT | Simulated Response Time |
| DU | Deployment Unit | PD | Power Demand | T | Transaction |
| EA | Enterprise Application | PU | Purchase | TPM | Transactions Per Minute |
| EAH | Enterprise Application Host | RC | Relative Change | U | User |
| EAU | Enterprise Application User | RDSEFF | Resource Demanding Service Effect Specification | V | Version |
| EAV | Enterprise Application Vendor | | | Y | Year |

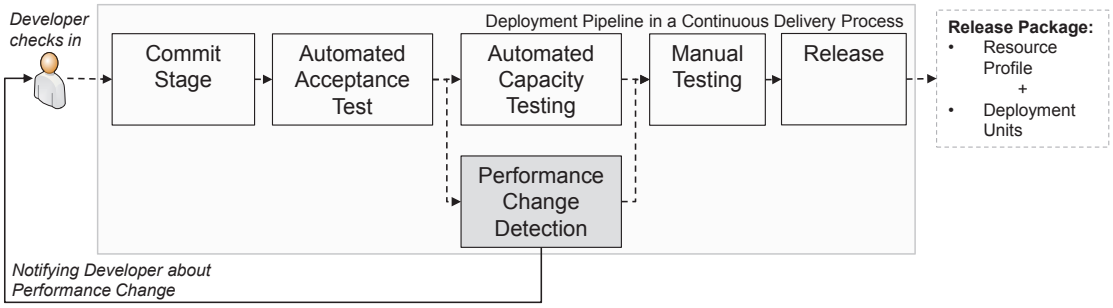



**Figure 10.1:** *Extended deployment pipeline (adapted from Humble/Farley (2010))*

how these resource profiles can be used to support capacity planning scenarios for new deployments of an EA version once it is released.

The performance change detection and capacity planning capabilities are evaluated using a representative enterprise application (section 10.6). Section 10.7 presents this work in the context of related work, and section 10.8 rounds off the concepts presented in this paper by outlining future research directions. Before the individual approaches are explained, the relationship of this work to previous work done by the authors is outlined in the following section 10.2.

## 10.2 A Comparison of the Present to Previous Work

This work enhances and integrates individual concepts presented in previously published articles (Brunnert/Vögele/Krcmar, 2013; Brunnert/Wischer/Krcmar, 2014; Brunnert/Neubig/Krcmar, 2014; Brunnert/Krcmar, 2014) to realize the performance evaluation and capacity planning capabilities outlined in the introduction. The relationship of the articles to the sections in this work is described in the following paragraphs.

Section 10.3 describes the idea of resource profiles as introduced in articles Brunnert/Wischer/Krcmar (2014) and Brunnert/Krcmar (2014). In comparison to the conference articles, this section includes a mathematical description of the resource profile content, whereas the articles only describe the content on an abstract, application-oriented, level.

Section 10.3.3 deals with the representation of resource profiles as performance models. While this section is based on our work in articles Brunnert/Vögele/Krcmar (2013); Brunnert/Wischer/Krcmar (2014); Brunnert/Neubig/Krcmar (2014) it describes the transformation technology independent and includes additional resource types such as network and HDD. The previous works described this transformation process only for one technology including central processing unit (CPU) and memory resources.

The continuous performance evaluation approach in section 10.4 is based on our previous work in Brunnert/Krcmar (2014). The content has been restructured and adapted to fit into the overall story line of this work. Furthermore, the performance change detection is now extended to use additional statistical techniques in order to evaluate if a change is statistically significant.

Section 10.5 on capacity planning using resource profiles is based on Brunnert/Wischer/Krcmar (2014). The content of this article is shortened and put into the context of the work introduced in section 10.4. For that purpose, a capacity planning process using resource profiles is introduced in section 10.5.2 and its individual steps are outlined in sections 10.5.3 and 10.5.4.

The three integrated controlled experiments in section 10.6 use resource profiles created during the change detection step in a deployment pipeline for capacity planning purposes and also take workload changes and network demands into account; validations which were not made in our previous works.

## 10.3 Resource Profiles

The term resource profile is used to describe the resource demand per transaction of an enterprise application version (Brandl/Bichler/Ströbel, 2007; King, 2004) and typically includes CPU usage, disk IO traffic as well as memory and network consumption (Brandl/Bichler/Ströbel, 2007; King, 2004). In order to derive performance metrics (i.e., response time, throughput, resource utilization) from these descriptions, it is necessary to transform them into models that can be used to predict performance.

Existing resource profile descriptions and corresponding approaches to derive performance metrics do not take software architectural information into account (King, 2004; Brandl/Bichler/Ströbel, 2007). This makes it impossible to evaluate performance on the level of single software components or to evaluate deployment changes (e.g., a different allocation of components). This work extends the existing resource profile definition in order to take this information into account. Furthermore, it introduces a transformation of these extended resource profiles into architecture-level performance models. These model-based resource profiles can be used for performance predictions by extending them with workload and hardware environment models.

Architecture-level performance modeling languages provide flexible meta-models to represent a software system (Koziolek, 2010). This flexibility leads to a broad interpretation of what is exactly represented in a performance model. The uncertainty makes it difficult to identify what kind of performance predictions can be made with a "performance model" as it depends on the content of a model. In order to address this uncertainty, resource profiles have a specific structure to make them easily usable for the purposes of assessing the current state of an EA's performance and for capacity planning. This structure and its transformation into a specific architecture-level performance modeling language are explained below.

### *10.3.1 Content and Structure*

A resource profile ($rp$) according to Brandl/Bichler/Ströbel (2007) and King (2004) is a set of vectors (i.e., $rp_{T_n}$) that describe the resource demand ($d$) for individual transactions ($T$, numbered from *1* to $n$) for a specific workload and a certain set of servers ($S$, numbered

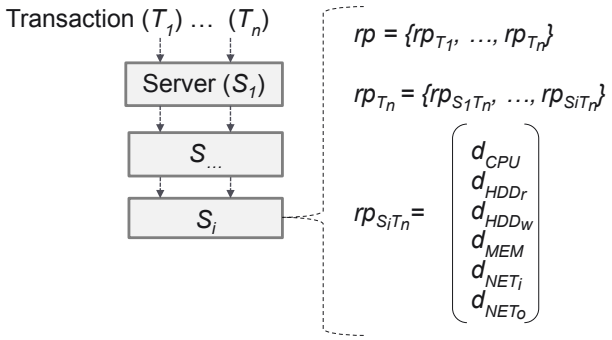



**Figure 10.2:** *Resource profile following the definition of Brandl/Bichler/Ströbel (2007) and King (2004)*

from *1* to $i$) as shown in figure 10.2. Depending on the type of hardware resource, resource demands either describe the time a transaction spends obtaining service from a resource (e.g., processing time for CPU) or the amount of work placed on a resource with limited capacity (i.e., memory consumption, data sent over a network or the amount of data written/read from disk). Resource profiles for enterprise applications only contain resource demands for the following resource types: CPU ($d_{CPU}$), HDD (differentiated by read $d_{HDD_r}$ and write $d_{HDD_w}$ operations), memory ($d_{MEM}$), and network (differentiated by incoming $d_{NET_i}$ and outgoing $d_{NET_o}$ traffic). If resource profiles should be used for other types of applications (e.g., mobile applications), further resource demand types (e.g., sensors) need to be represented (Willnecker/Brunnert/Krcmar, 2014b).

Following the basic idea of Brandl/Bichler/Ströbel (2007) and King (2004), resource profiles comprise sets of resource demand vectors for individual transactions. However, instead of representing transactions using vectors for individual servers in distributed application architectures as shown in Brandl/Bichler/Ströbel (2007) and King (2004), resource profiles are represented from a software perspective. Therefore, resource demands are now associated with component ($C$, numbered from *1* to $p$) operations ($OP$, numbered from *1* to $r$) of an EA. A transaction (i.e., $rp_{T_n}$) is therefore represented by a set of resource demand descriptions for all component operations (i.e., $rp_{C_p\,OP_r\,T_n}$) that are involved in its control flow as shown in figure 10.3. This change in perspective is necessary to allow for resource profiles that represent single enterprise applications independently from their deployment topology.

The resource demand of a component operation within the control flow of a transaction is specified by a set of vectors (i.e., $rp_{C_p\,OP_r\,T_n}$). These vectors (i.e., $rp_{C_p\,OP_r\,T_n\,BR_m}$) describe the resource demands for single component operation control flows, called branches ($BR$, numbered from *1* to $m$). Branches represent control flow variations of a component operation within the transaction processing and their corresponding resource demands. As parametric dependencies are not depicted by the chosen granularity level of single transactions, it is important to differentiate these control flows taking into account slightly modified parameters or users of a system. The control flows are differentiated depending on the subsequently called component operations. Therefore, each branch vector contains a control flow set ($cf$) that includes the resource demand descriptions of subsequently called component operations within the transaction control flow (e.g., $cf$ of $rp_{C_p\,OP_r\,T_n\,BR_m}$

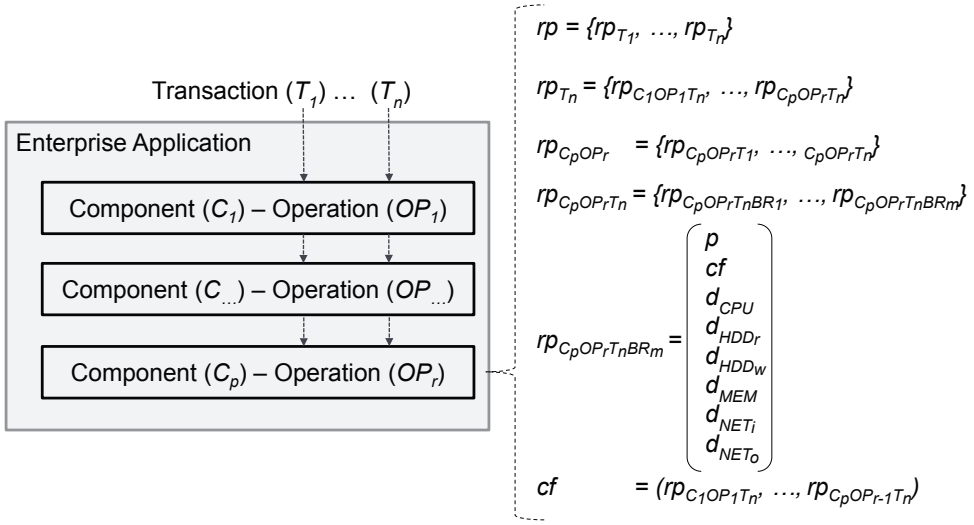



**Figure 10.3:** *Resource demand vectors on transaction and component operation level*

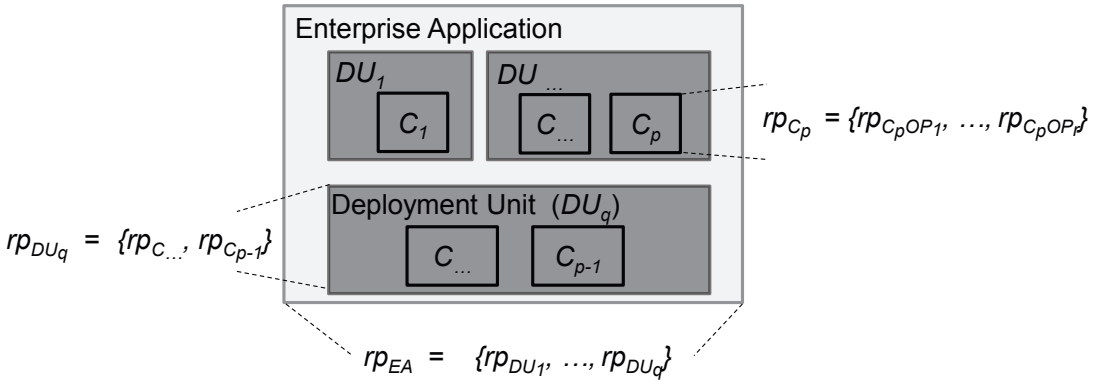



**Figure 10.4:** *Resource demand sets on component, deployment unit and enterprise application level*

contains $rp_{C_p OP_{r-1} T_n}$ in figure 10.3). Furthermore, the probability ($p$) of its occurrence is specified in a branch-specific vector (i.e., $rp_{C_p OP_r T_n BR_m}$).

The overall control flow of a transaction can be reconstructed by creating a graph based on the data in the control flow sets of all component operations involved in the transaction processing. The starting point of a transaction ensues after choosing those branch-specific vectors of a component operation (i.e., $rp_{C_p OP_r T_n}$) that are not contained within a control flow set (i.e., $cf$) of another component operation. The control flows within a component operation are therefore assumed to be independent of those of their predecessor and successor component operations within the same transaction. This assumption reduces accuracy as invalid execution flows may occur. However, representing each individual execution flow of a transaction separately while still maintaining the component operation structure would result in an inordinate amount of difficult to manage data. Further, this approach eases the collection of data if an EA is distributed across multiple servers and technologies, as only the part of transaction processing executed within the current server and/or technology needs to be considered. Using this approach, less data needs to be exchanged across different technologies and servers.

To further depict the software architecture, the resource demand descriptions for individual transactions (i.e., $rp_{C_p\,OP_r\,T_n}$) are aggregated per component operation (i.e., $rp_{C_p\,OP_r}$) (figure 10.3). If no resource demand set is available for a transaction, the operation is not involved in the control flow of a transaction and, thus, consumes no resources whenever the transaction is invoked. A next step to reconstruct the software architecture is to aggregate the resource demand sets of component operations (i.e., $rp_{C_p\,OP_r}$) for individual components (i.e., $rp_{C_p}$)(figure 10.4).

The next level of aggregation are deployment units ($DU$, numbered from *1* to $q$) as shown in figure 10.4. The reason for introducing this level of aggregation is that this is the level of granularity on which enterprise applications are deployed on a system (Liu/Smith, 2006). It is therefore possible to estimate the additional load placed on a hardware environment as soon as an individual deployment unit of an application is deployed on it. The type of components included in a $DU$ set can either represent binary packages for middleware components (Liu/Smith, 2006) or just bundles of scripts for dynamic web applications and stored procedures for databases. The resource profile for an overall EA (i.e., $rp_{EA}$) is then described by the set of profiles for its individual deployment units (figure 10.4).

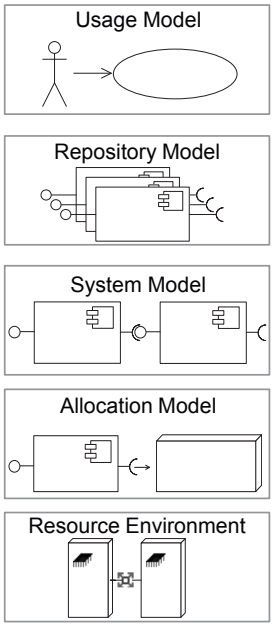
### *10.3.2 Representing Resource Profiles as Architecture-Level Performance Models*

In order to derive performance metrics based on resource profiles it is not only important to be able to specify the expected workload on an application but also its hardware environment and deployment topology (i.e., the distribution of deployment units to the available servers). Resource profiles are therefore represented using a specific performance model type, called architecture-level performance models. In this section, we explain the reason for choosing this model type and a specific modeling notation.

Numerous performance modeling approaches have been proposed in literature (Balsamo et al., 2004; Koziolek, 2010). Using these models as input for analytical solvers or simulation engines allows the prediction of response time, throughput and resource utilization for a modeled software system. A performance model of an EA typically contains performance-relevant aspects of an application architecture, the hardware environment and the workload. Conventional performance modeling approaches (Balsamo et al., 2004) such as Queuing Networks, Layered Queuing Networks (LQN) or Queuing Petri Nets depict all these aspects nested in one single monolithic performance model. It is therefore hard to change a single aspect, such as the hardware environment or the workload, without needing to substantially change the whole performance model. Architecture-level performance models (Koziolek, 2010) try to separate these aspects to simplify the modeling process.

Resource profiles represent a way to describe the performance-relevant aspects of an EA's architecture based on the probabilistic behavior of transactions. Using architecture-level performance models is therefore the only way to represent the resource profile data in a way that allows the user to add the workload and hardware environment later on.

Several architecture-level performance modeling notations have been proposed in literature (OMG, 2005; Smith et al., 2005; Grassi/Mirandola/Sabetta, 2007; Petriu/Woodside, 2007; OMG, 2011; Reussner et al., 2011; Kounev/Brosig/Huber, 2014). A comprehensive overview




- Workload
- Components, interfaces, relationships, control flows, resource demands
- System composed of components within the repository model
- Mapping of system components to available servers
- Available servers and their network connections

**Figure 10.5:** *The layers of the Palladio Component Model (PCM) (adapted from Becker/ Koziolek/ Reussner (2009))*

of existing architecture-level modeling notations can be found in the survey of Koziolek (2010). In order to decide on a modeling notation for the resource profile representation, we primarily looked at the maturity of the notations and their associated tooling. The maturity was measured by the number of case studies published using the approach as already done by Koziolek (2010). According to these aspects, we decided to use the Palladio Component Model (PCM) (Reussner et al., 2011) due to the maturity of the notation and its continuously maintained tooling infrastructure. However, a transformation into more recently introduced architecture-level performance modeling notations such as the Descartes Modeling Language (DML) (Brosig/Huber/Kounev, 2014; Kounev/Brosig/ Huber, 2014), might be implemented in the future as these approaches mature.

PCM is described by Reussner et al. (2011) as a software component model for business information systems to enable model-driven quality of service (QoS, i.e. performance) predictions. The PCM meta-model comprises several model layers referencing each other as shown in figure 10.5. The key layers for representing the resource profile information are the repository and system models. The repository model contains components of a software system, their relationships and resource demands. Components from the repository model are combined within a system model to represent an EA. The workload for an EA, represented by a system model, is specified in a usage model. The hardware environment on which an EA can be deployed is defined in a resource environment model. The allocation of EA components to servers within the hardware environment is specified in an allocation model.

In order to support the use of the PCM modeling notation, there exists an extensive set of tooling[2] to create PCM models and to derive performance predictions. Even though there exists a wide variety of options to derive performance predictions, this work focuses on

[2] `http://www.palladio-simulator.com/`

the default simulation engine of PCM called SimuCom (Becker/Koziolek/Reussner, 2009). The reason for this focus is that SimuCom supports all aspects of the PCM meta-model whereas other solvers often neglect specific features.

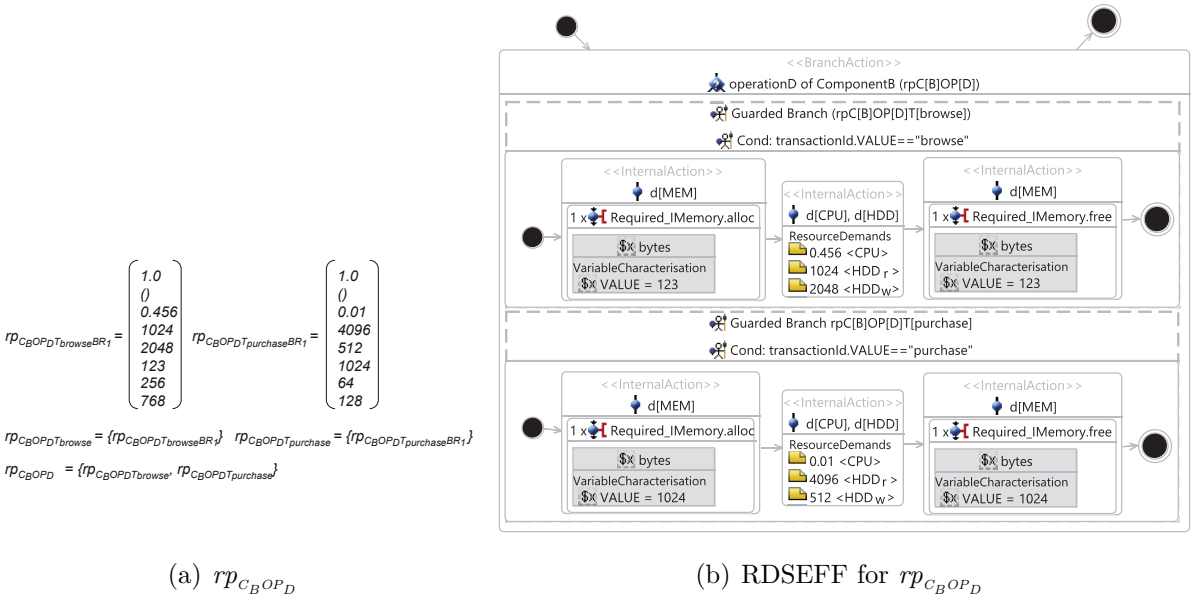
### *10.3.3 Transforming Resource Profiles into PCM Models*

The basic requirement for creating a resource profile with the content and structure outlined in section 10.3.1 is the collection of the required data to create the input vectors and sets. We assume that the resource demand values included in the input vectors are mean values. A prototype exists which allows the automatic transformation of input data available in this type of data format into PCM models (Brunnert/Vögele/Krcmar, 2013; Brunnert/Neubig/Krcmar, 2014). The basic concepts for the transformation are outlined in this section.

The required input data for the prototype can either be collected in a database as outlined in Brunnert/Vögele/Krcmar (2013) or within a specific Java-based data structure as shown in Brunnert/Neubig/Krcmar (2014). Furthermore, several monitoring solutions exist to collect the required information and we have shown how to use them as an input source in Willnecker et al. (2015a). Technologies also exist which are capable of transforming measurements that are not on the level of resource demands into a corresponding representation (Spinner et al., 2014).

The resource profile data are used to create the repository and system model layers of PCM. The components within the repository model and their operation behavior are created based on resource demand sets that represent their individual behavior for single transactions (i.e., using the content of $rp_{C_p OP_r}$). The behavior of single component operations is represented in Resource Demanding Service Effect Specifications (RDSEFFs). RDSEFFs are behavior descriptors similar to activity diagrams within the Unified Modeling Language (UML). The component operation behavior description for each individual transaction is represented according to the corresponding resource profile data (e.g., $rp_{C_p OP_r T_n}$). An example of a RDSEFF for the *operationD* of *ComponentB* based on the resource profile data shown in figure 10.6(a) is depicted in figure 10.6(b). This RDSEFF comprises two guarded branches containing the condition that the transaction in which the component is involved in needs to be either *browse* or *purchase*.

The RDSEFF in figure 10.6(b) furthermore contains resource demand data from the corresponding resource demand descriptions (i.e., $rp_{C_B OP_D T_{browse}}$ and $rp_{C_B OP_D T_{purchase}}$) shown in figure 10.6(a). Within each guarded branch, the CPU resource demands ($d_{CPU}$) are specified as millisecond (ms) values. In this example, the component operation consumes 0.456 ms CPU time when it is called within the context of the *browse* transaction, whereas the same component operation consumes 0.01 ms CPU time when it is called as part of the control flow of the *purchase* transaction. The HDD resource demands are specified as bytes written to or read from a disk. The component operation reads 1024 bytes from and writes 2048 bytes to disk when it is called within the context of the *browse* transaction, whereas it reads 4096 bytes and writes 512 bytes when it is called as part of the control flow of the *purchase* transaction.



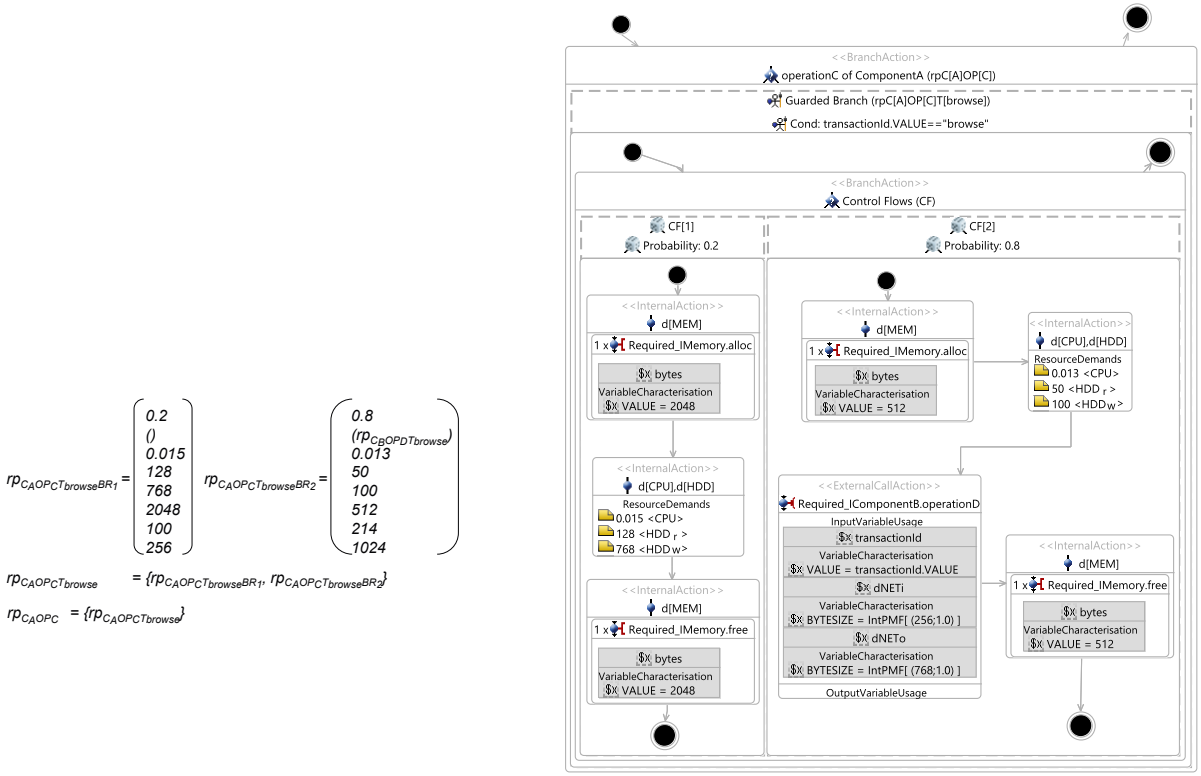

(a) $rp_{C_BOP_D}$ (b) RDSEFF for $rp_{C_BOP_D}$

**Figure 10.6:** *PCM RDSEFF representation of a simple resource demand description of a component operation*

The RDSEFF in figure 10.6(b) also contains the memory demand ($d_{MEM}$) of each control flow within the guarded branches. Because representing memory is not directly supported by the PCM meta-model, a *Memory* component is introduced in each PCM-based resource profile (see figure 10.8(b)) representing the available memory capacity of a system in bytes and allowing other components to consume memory (Brunnert/Neubig/Krcmar, 2014). The *Memory* component offers two operations: *alloc* and *free*. These operations are used within the RDSEFFs for component operations to allocate and deallocate memory. In the example in figure 10.6(b), the operation control flow for the *browse* transaction consumes 123 bytes of memory, whereas the operation consumes 1024 bytes of memory within the control flow of the *purchase* transaction. This model follows the application programming interface (API) for applications written in the programming language C. However, this model is a simplification and is not accurate for modern runtime environments such as Java or .NET as it does not reflect their automatic memory management and garbage collection capabilities. It does, however, allow for an approximation of the memory demand of the application for a specific workload.

The example in figure 10.6 contains exactly one control flow for each transaction in which the component operation is involved in. This is the case because no other external operations are called by *operationD* of *ComponentB*. An example with two control flows is shown in figure 10.7. The *operationC* of *ComponentA* is only involved in the transaction processing of the *browse* transaction although it contains two different control flows. Control flow one (*CF[1]*) is executed with 20 % probability whereas control flow two (*CF[2]*) is executed with 80 % probability. Control flow two also contains an example of a dependency between two component operations: *operationC* of *ComponentA* calls *operationD* of *ComponentB*. Such dependencies are derived from the ordered sets that describe each transaction control flow within each component operation (i.e., *cf*).

If the same component operation is contained multiple times directly one after another in an ordered control flow set of a resource demand column (e.g., if the second control




(a) $rp_{C_A OP_C}$ (b) RDSEFF for $rp_{C_A OP_C}$

**Figure 10.7:** *PCM RDSEFF representation of resource profile data with external operation calls*

flow of $rp_{C_A OP_C T_{browse}}$ in figure 10.7(a) would contain $rp_{C_B OP_D T_{browse}}$ twice), then the PCM RDSEFF representation would contain a loop around the external operational call with the corresponding loop count. The transformation also takes care of combining control flow columns that only differ by the amount of times an external component operation is called in a row, so that these differences are only depicted using a specification of the probabilities for certain call loop counts.

The call from *operationC* of *ComponentA* to *operationD* of *ComponentB* contains multiple variable characterization elements. These elements specify parameters that are transferred between both operations. In this case, the *transactionId* that has been used to call *operationC* is passed on to *operationD* to differentiate the operation behavior as shown in figure 10.6. Furthermore, two parameters (*dNETi* and *dNETo*) are specified to depict the network demand of this call. The network demand is specified by the *BYTESIZE* values. The values of these parameters are derived from the network demand values ($d_{NET_i}$=*256* and $d_{NET_o}$=*768*) from the resource demand description of the *operationD* for the transaction *browse* shown in figure 10.6(a). In this example, only one control flow is available as shown in figure 10.6. Therefore, the probability that 256 and 768 bytes are transferred over the network is always one (see Integer Probability Mass Function (IntPMF) in figure 10.7). If more control flows for the current transaction were to exist, then the IntPMF functions would contain entries for each control flow defined in the resource profile set of the corresponding operation specifying the network demand and its probability (i.e., $p$). It is important to note that the default simulation engine for the PCM meta-model (SimuCom, (Becker/Koziolek/Reussner, 2009)) needs to be enhanced to

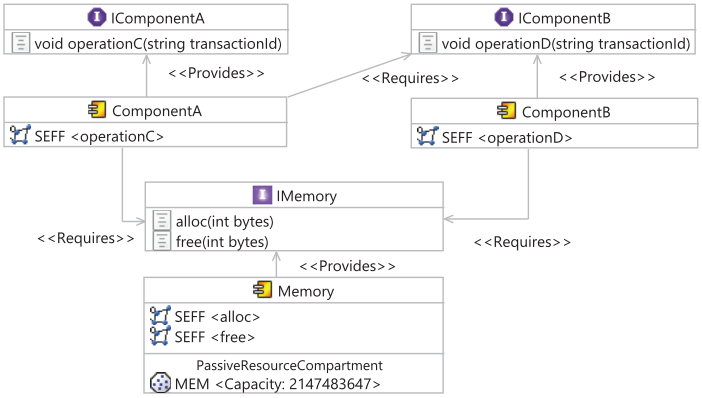
take these different network demands into account[3]. The default implementation assumes that the same network demands are used for the incoming and outgoing traffic and uses the BYTESIZE of an input parameter as the network demand of the return value. The SimuCom extension now uses the dNETi and dNETo BYTESIZE values so that the incoming and outgoing traffic is properly handled during the simulation.

An overview of the entire repository model for the examples in figures 10.6 and 10.7 is shown in figure 10.8. The model includes the components *A* and *B* and defines their dependency required for the call from *operationC* of *ComponentA* to *operationD* of *ComponentB* shown in figure 10.7. This dependency is specified by a *requires* relationship between *ComponentA* and an interface *IComponentB*. Interfaces in this model specify the available operations. The interfaces and operations are created based on available data for each component (e.g., $rp_{C_A}$ and $rp_{C_B}$ as shown in figure 10.8(a)).

$$rp_{C_A} = \{rp_{C_A OP_C}\}$$

$$rp_{C_B} = \{rp_{C_B OP_D}\}$$

(a) $rp_{C_A}$ and $rp_{C_B}$



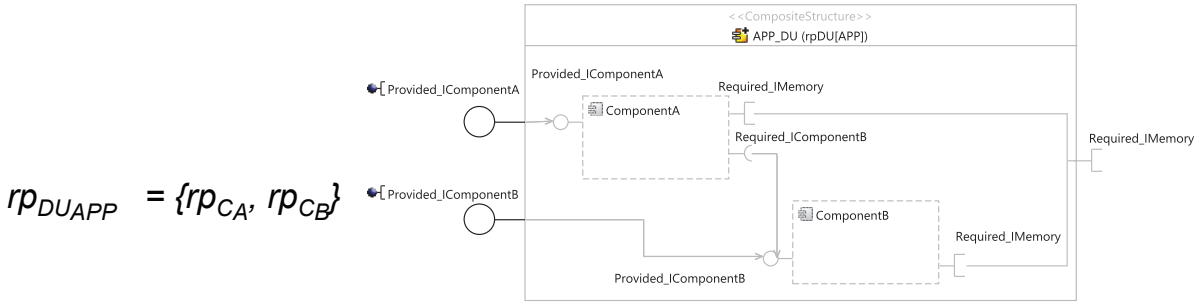

(b) Repository model representation of $rp_{C_A}$ and $rp_{C_B}$

**Figure 10.8:** *PCM repository model example for the representation of application components*

The data in the sets for deployment units (e.g., $rp_{DU_{APP}}$ shown in figure 10.9(a)) are used in a next step to group individual components in so-called composite components of the PCM meta-model. An example for this type of grouping can be found in figure 10.9(b). The deployment unit contains both components of the application. These composite components are the only enterprise application artifacts represented within a PCM system model. With this representation, resource profile users are not able to evaluate deployment options which are not supported by the creator of a resource profile (e.g., a software vendor). In the example in figure 10.9(b), the deployment unit specification requires only one external interface, namely the *Memory* component. This way of representation is necessary to share the available memory between multiple deployment units on the same server.

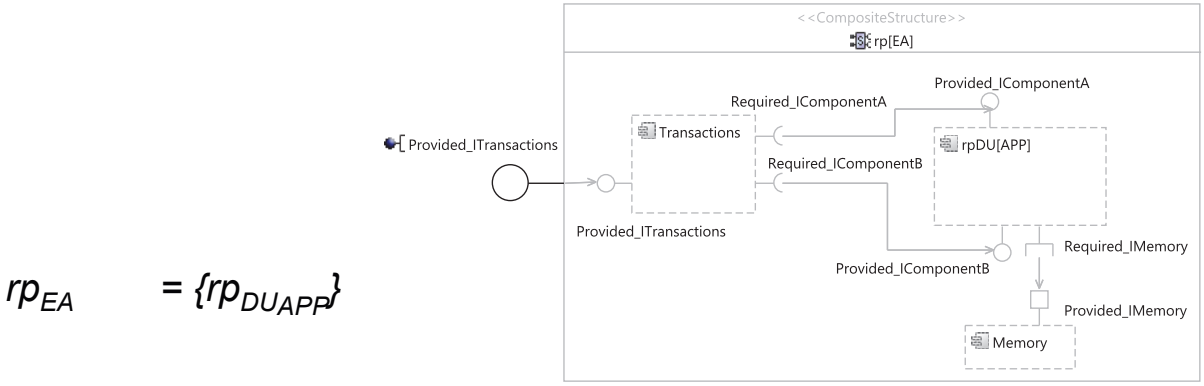
Figure 10.10 shows an example of a system model containing only deployment units ($DU$) in addition to two common components that are always created by the transformation (the *Memory* and *Transactions* components). The advantage of hiding everything else within the system model is a reduction in the complexity of the PCM-based resource profiles

[3]The corresponding SimuCom extension is available in the supplemental material to this article: `http://pmw.fortiss.org/research/JSS`




(a) $rp_{DU_{APP}}$ (b) PCM representation of $rp_{DU_{APP}}$

**Figure 10.9:** *Example for a PCM-based representation of a deployment unit*




(a) $rp_{EA}$ (b) PCM representation of $rp_{EA}$

**Figure 10.10:** *Example for a PCM-based representation of a resource profile for an enterprise application*

for users. The users only need to interact with the *DUs* and map them on the available servers in the hardware environment as shown in figure 10.12(b).

To make it similarly easy for users to specify the workload on the system, a *Transaction* component is introduced that abstracts the transactions from the component operations. This component provides one common interface for all transactions represented in a resource profile so that a user of a resource profile does not need to interact with individual component interfaces to specify the workload as shown in figures 10.10(b) and 10.11. As mentioned earlier, the starting point of a transaction ensues after choosing those branch-specific vectors of a component operation that are not contained within a control flow set of another component operation. The operations within the transaction component therefore call the RDSEFFs representing these control flow sets when they are invoked.

### *10.3.4 Predicting Performance Using PCM-based Resource Profiles*

In order to predict performance using a resource profile represented in the PCM meta-model, it is necessary to create workload and hardware environment models. Figure 10.11 shows an example of a workload specification for the system shown in figure 10.10(b). Figure 10.11 depicts a usage behavior in which half of the users execute only the browse

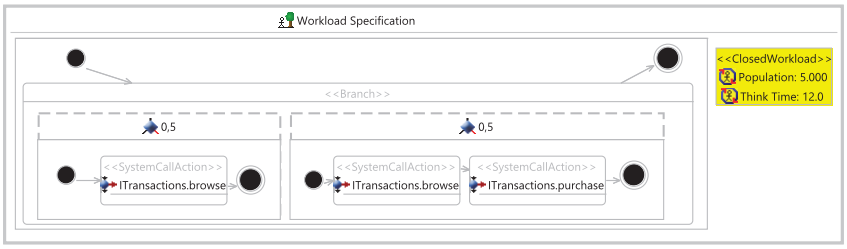



**Figure 10.11:** *Example for a workload specification in a usage model for the resource profile example*

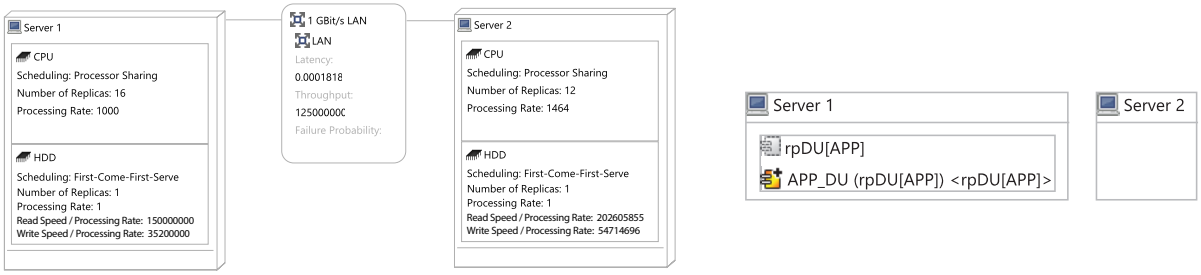
transaction whereas the other half also purchase something afterwards. It is important to note that the variability on the level of user behavior is limited to different distributions between the transactions represented in a resource profile, further variations (e.g., different parameters) are not possible. However, this is not a limitation of the PCM meta-model but rather of the resource profile content, as only resource demands and probabilities for specific transactions are represented. As the user behavior is only one aspect of a workload, the usage model in figure 10.11 furthermore specifies how many users are interacting with the system and their ”think time” between two interactions. The figure contains an example for a closed workload with 5000 users and a think time of 12 seconds. Instead of using a closed workload, it is also possible to specify an open workload with interarrival times instead of fixed user counts and think times.

An example for a hardware environment is shown in figure 10.12(a). The main elements in this hardware environment from a resource demand perspective are the processing rate of resources and the throughput specified in the linking resource. All values included in a resource profile are specified relative to these values. For example, a processing rate of 1000 means that 1000 units of work can be executed per simulated time frame. For resource profiles, we assume a simulated time frame to be one second. Therefore, CPU demand values specified in RDSEFFs are interpreted as ms.

For the HDD resource, it is not only necessary to specify the processing rate but also the write and read speed in bytes per simulated time interval. As the processing rate for the HDD resources is set to 1, the speed is defined in bytes per second. These speed definitions are used to transform the byte values defined in the component operations into consumed time on the HDD resource during the simulation. As such a differentiation is not supported by the current PCM and SimuCom versions, we needed to extend the meta-model and the simulation engine in order to support this feature[4]. We have leveraged the fact that PCM already provides a way to differentiate read and write calls for HDD resource demands. Therefore, our extension primarily focused on specifying the read and write speeds for the HDD resource in the resource environment model and extending SimuCom to calculate the consumed time on demand using the provided data in the resource environment and repository models.

The linking resource that specifies a one gigabit-per-second (GBit/s) local area network in this figure, specifies a latency and a throughput. The throughput is specified in bytes per

[4]The corresponding meta-model and SimuCom extensions are available in the supplemental material to this article: `http://pmw.fortiss.org/research/JSS`




(a) Example for a resource environment model (b) Example for an allocation model

**Figure 10.12:** *Hardware environment models*

second, so that the network demand values in the external operation calls are properly processed by the simulation engine. The latency is specified in seconds.

Whenever a resource profile is intended to be used to predict performance for a hardware environment which was not the one used to create it, either the resource profile values or the processing rates and throughput/latency values need to be adapted. For example, in order to show that server two includes a faster processor than server one in figure 10.12(a), the processing rate is increased from 1000 to 1464. The details of this process are outlined in section 10.5.3. Please note that we do not explicitly model additional infrastructure elements such as a virtualization layer as shown in the work of Huber et al. (2011) and Brosig et al. (2013). The reason for this is that the processing rates in the resource environment models should already describe the resource speed available to an EA directly. For example, the processing rate would already depict a different value for a virtualized CPU compared to a non-virtualized one.

An allocation of the deployment unit (figure 10.9) to the available hardware environment is shown in figure 10.12(b). In this figure, the deployment unit is placed on server one. Using the PCM models shown in figures 10.10(b), 10.11 and 10.12(b) as input for the PCM simulation engine SimuCom (Becker/Koziolek/Reussner, 2009), allows the prediction of the performance for the specified workload and hardware environment based on the specifications in a resource profile.

## 10.4 Continuous Performance Evaluation Using Resource Profiles

The key goal of performance evaluations during the software development process is to ensure that performance requirements (e.g., maximum response time targets) can be met (Brunnert et al., 2014). For this purpose, the performance of an EA needs to be continuously evaluated in order to analyze the impact of new features or bug fixes on performance. Therefore, the performance change detection process introduced in this section allows detection of performance changes in each EA version being built.

To ensure that the performance change detection is executed for each EA version, this work proposes its integration into a deployment pipeline as an alternative to capacity tests as shown in figure 10.1. In their description of the capacity tests within the deployment pipeline, Humble/Farley (2010) define capacity as the maximum throughput a system

can achieve with given response time thresholds for different transaction types. This definition implies that a system comparable to the final production environment needs to be available during the automated capacity testing step. If this is not the case, results from capacity tests according to their capacity definition would not yield meaningful results. As outlined in the introduction, this precondition does not often exist in development projects (Brunnert et al., 2014). The immediate feedback promised by CD systems can, thus, often not be guaranteed using this approach, as only smaller scale systems are available. The capacity test results derived from such systems are hardly comparable to a production environment.

Using resource profiles, performance can be predicted for hardware environments with more resources (e.g., CPUs) than those that are available (Brunnert/Wischer/Krcmar, 2014). Furthermore, resource profiles can be adapted to a hardware environment which is more comparable to the production environment using an approach presented later in section 10.5.3. These abilities speed up the feedback loop and allow for performance evaluations that would be otherwise impossible. Real capacity tests can then be executed later in the process or as a manual test step when appropriate systems are available.

Following the definition of Cherkasova et al. (2009), performance changes are defined as an increase or decrease of transaction processing time. Detecting performance changes using resource profiles requires a series of steps: In a first step, a resource profile for the current application version needs to be created. This resource profile version must be put into a versioning repository to make it available for subsequent builds. Afterwards, performance is predicted using the current version and the results are compared to prediction results from a previous version to see if a performance change occurred. If a change is detected, a notification is sent to the development team(s). The next sections explain the realization of these steps and their integration into a continuous delivery process.

### *10.4.1 Creating Resource Profiles*

In order to decide when a resource profile can be created within a deployment pipeline, it is important to understand the two steps before the performance change detection should take place: As a first step in the deployment pipeline (figure 10.1), a build is triggered by one or multiple developer check-ins. These check-ins are used in a so-called commit stage to build a new version of an application and to execute a set of predefined tests. These tests focus on low-level API tests from a developer perspective. In a second step, automated acceptance tests evaluate if an application that is being built truly delivers value to a customer. A set of regression tests is then executed to evaluate the functionality from an end-user perspective.

We propose the use of dynamic analysis to collect measurements during the automated acceptance test execution to create resource profiles. This approach has several advantages, apart from the fact that it saves time compared to a separate measurement run (according to Humble/Farley (2010), acceptance tests usually take several hours to complete). Transactions that are being tested and executed in the acceptance test stage are expected to be close to the behavior of the users (Humble/Farley, 2010). Using measurements from the acceptance tests also ensures that the workload stays relatively stable as the regression

tests are executed for every build. The measurement results for each build are, therefore, comparable.

### *10.4.2 Versioning Resource Profiles*

Each reusable artifact created within the deployment pipeline is stored in an artifact repository (Humble/Farley, 2010). This is necessary to make each artifact available in different steps of the deployment pipeline. To detect performance change, we require the resource profile of the current version as well as the profiles of previous builds. Therefore, each resource profile version created in the deployment pipeline is stored in an artifact repository thereby allowing the management of different resource profile versions.

As PCM is based on the Eclipse Modeling Framework (EMF)[5], resource profile representations in this meta-model conform not only to the PCM meta-model but also to the Ecore meta-model defined by EMF. We leverage this capability by using the EMFStore (Koegel/Helming, 2010) which implements the required versioning features for models based on the Ecore meta-model.

The advantage of using EMFStore rather than other versioning systems is that it is especially designed to support the semantic versioning of models (Koegel/Helming, 2010). Instead of working with textual representations of models, EMFStore uses the Ecore model elements and their relationships to manage models stored in the repository. Thus, rather than representing a structural change between two model versions as multiple lines in their textual representation, EMFStore directly stores the change in the Ecore model itself (Koegel/Helming, 2010). The only two PCM model layers automatically stored and versioned in the EMFStore are the PCM repository and system models that represent a resource profile.

### *10.4.3 Predicting Performance*

Using PCM-based resource profiles, performance predictions can be made for different workload and hardware environment models described in section 10.3.4. These predictions include results for the performance metrics response time, throughput and resource utilization. As performance change is defined as changes in the transaction processing time, response times in the prediction results are used as an indicator for change.

To enable comparable predictions in the deployment pipeline, workload and hardware environment models for performance predictions need to be statically defined. As explained in section 10.3.4, the workload on a system can be represented using usage models. The hardware environment is represented using the resource environment and allocation models. The artifact repository, therefore, needs to contain usage models as representation of different workloads which use the external interfaces provided by the system model of a resource profile to specify the load. The artifact repository should also contain one or more resource environment models specifying the available servers and corresponding

[5] http://www.eclipse.org/modeling/emf/

allocation models which define the mapping of deployment units specified in a resource profile to the servers.

Before performance can be predicted, a lookup needs to be made in the artifact repository to obtain the corresponding workload and hardware environment models. These models are combined with a resource profile to predict performance. If a hardware environment should be used different than the one the resource profile was created on, the processing rates of the hardware resources need to be specified relative to the original resources. Applying benchmark results for both the source and target hardware resources, as outlined in section 10.5.3, is one approach that can be used for that purpose.

### *10.4.4 Comparing Prediction Results*

To detect a change in performance, performance is predicted with the current resource profile version and a specified set of hardware environment and workload models. If a previous resource profile version for the same application is available, the same predictions are executed using the previous version. The prediction results are then compared with one another. The reason for executing the predictions with the previous resource profile version again is to ensure that the same workloads and hardware environments are used. This avoids situations in which changes in the workload or hardware environment models could possibly lead to incorrect results.

For each transaction, an evaluation of any statistically significant change is made. The dispersion of the underlying distribution can change between the response time sets collected for the same transaction in different versions. To account for this, the two-sample Kolmogorov-Smirnov test (Conover, 1971) is used in a first step to identify if two response time distributions differ significantly. This test is used because it makes no assumptions about the underlying distributions. In order to apply this test, the discrete continuous set of predicted response times for a transaction of the current and previous application versions is used as input. The test derives a distance between both response time sets as shown in equation 10.1. In this equation, $F_{current,n}(x)$ and $F_{previous,n'}(x)$ denote the empirical distribution functions of the response times for a transaction of a current and previous application version. The supremum ($\sup_x$) denotes that the distance is represented by the largest difference between the empirical distribution functions. Whether this distance indicates a significant difference between both response times sets with size $n$ and $n‘$ respectively can be calculated as shown in equation 10.2. In this equation, $c(\alpha)$ denotes a coefficient statically defined for sufficiently large $n$ and $n‘$ values and specific significance levels (Smirnov, 1948). For a significance level ($\alpha$) of 0.05, $c(\alpha)$ is 1.36 (Smirnov, 1948). This significance level and corresponding $c(\alpha)$ value is used for the performance change detection.

$$D_{n,n‘} = \sup_x |F_{current,n}(x) - F_{previous,n‘}(x)| \tag{10.1}$$

$$D_{n,n‘} > c(\alpha)\sqrt{\frac{n + n‘}{n * n‘}} \tag{10.2}$$

If the Kolmogorov-Smirnov two-sample test indicates a significant difference, the relative predicted response time change between the current and the previous application version

is calculated. As a prediction results in multiple response time values over time, this set of transaction response time values can be represented in multiple ways (Jain, 1991; Brüseke/ Engels/Becker, 2013). The most common methods to represent these values are mean values including distances from the mean (e.g., standard deviation) and percentiles (e.g., 50th (median) or 90th percentile). Which of these values best represents a response time set depends on the dispersion of the underlying distribution (Jain, 1991). As the response time distribution can change between versions, the relative change is always calculated for mean, median values and 90th percentiles to make the results comparable. We propose to calculate the relative change for a transaction ($T_{rc}$) as shown in equation 10.3. In this equation, the transaction response time ($T_{rt}$) predictions of the *current* and *previous* resource profile versions are compared with each other.

The resulting change values for mean, median and the 90th percentile indicate whether the response time is now higher (positive value) or lower (negative value) than previously. The user can specify separate thresholds for each of these indicators. If at least one relative change value for a transaction is higher than a specified threshold (e.g., a greater than 20 % increase or decrease) a notification (e.g., as email from the build system) is sent to the developer(s) providing a list of transactions with performance regressions and the corresponding relative change values. If response time increases above a specified threshold, the deployment pipeline needs to be stopped. It is important to stop the pipeline in this case to avoid marking the application version as stable which ensures that the next build will be compared with a valid resource profile of a version with meaningful performance.

$$T_{rc} = \frac{T_{rt_{current}} - T_{rt_{previous}}}{T_{rt_{previous}}} \tag{10.3}$$

To enable a distinction between runs with and without changes, the corresponding resource profile versions need to be managed independently within the EMFStore; a resource profile should only be marked as usable for subsequent builds if the relative change is below the specified threshold. In order to allow for situations in which a new feature is introduced that causes a performance regression and this regression is acceptable, a developer can connect to the EMFStore and mark the resource profile of the failed build as successful so that the next build to include this feature will be regarded as stable.

To identify trends, a comparison can also be made against the resource profiles of more than one of the last successful builds. This avoids situations in which performance regressions develop slowly across multiple builds.

### *10.4.5 Comparing Resource Profiles*

Users can access, analyze and edit models in the EMFStore using a plugin for the PCM modeling environment[6]. As a result, resource profile versions in the artifact repository are directly accessible to developers after they are notified about a performance change. The notification could also include a link to the corresponding resource profile versions. Each

[6]http://www.palladio-simulator.com/

resource profile in the EMFStore can be analyzed over time to see which components, relationships or resource demands are associated with specific versions stored in the repository.

Because resource profiles for normal application versions and for versions with performance changes are managed independently from each other in the EMFStore, one cannot easily analyze the differences between these versions. For this purpose, a comparison can be made between the different resource profile versions using the EMF Compare framework[7]. EMF Compare allows differences between models conforming the Ecore meta-model to be automatically analyzed and visualized.

The level of available detail depends on the data collection approach for the resource profiles. We assume that the components of a system, their operations and their relationships are represented in the resulting PCM-based models (Wu/Woodside, 2004), including corresponding resource demands. Therefore, the result of a comparison reveals changes in resource demands, control flows and component operations. Using this information and information related to which changes were performed during the check-ins that initiated the build, the developers can identify the sources for a performance change.

## 10.5 Capacity Planning Using Resource Profiles

Once an EA version has been released (figure 10.1), the primary use case of a resource profile is to support capacity planning. This support is important in cases when multiple deployments of the same EA exist or when not all deployments are known at the time of a release (e.g., for off-the-shelf products).

### *10.5.1 Use Cases*

Although a lot of work on capacity planning using performance models exists, these works often focus on a technical perspective and explain modeling and solution techniques from a software system life cycle perspective (Menascé/Almeida, 2002). The organizational (IT governance) perspective is often neglected (Brunnert et al., 2014). One of the few examples is PCM which explicitly takes different organizational roles into account (Becker/Koziolek/Reussner, 2009). However, this perspective is still limited to that of one development organization. This section takes a cross-organizational perspective and explains use cases in which the transferable nature of resource profiles helps to simplify the relationships of enterprise application vendors (EAV, i.e., software or consulting companies), users (EAU, i.e., companies who source software from EAVs) and hosts (EAH, i.e., data center providers).

The knowledge required for constructing a resource profile and for specifying the input variables for the evaluation is often distributed between different parties (i.e., EAV, EAU and EAH, see figure 10.13). Resource profiles are therefore meant to be used differently

[7] http://www.eclipse.org/emf/compare/

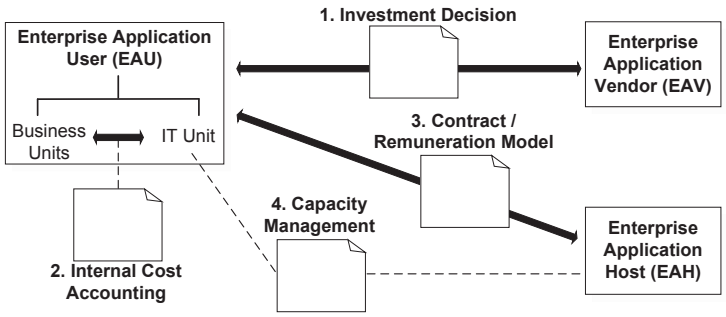



**Figure 10.13:** *Use cases for a resource profile once an EA is released (adapted from Brunnert/ Wischer/ Krcmar (2014))*

depending on the available information. An EAV should create and distribute resource profiles for all enterprise applications sold (off-the-shelf and custom developments), which then can be adapted by EAUs and EAHs for their specific needs. EAUs and EAHs can modify the workload and the hardware environment but reuse the specifications provided by an EAV.

If an EAU is interested in a new enterprise application, he could use the corresponding resource profile as one component in an overall *investment decision* (see 1. in figure 10.13). The EAU can specify the expected amount of users, their behavior and the existing hardware environment to evaluate if this hardware would be sufficient to run the application for his needs. At the same time, the EAU could evaluate the impact of this particular application on its operation costs. If new hardware is needed for the application, the resource profile helps to compare different hardware configurations in terms of their impact on performance and operational costs. Resource profiles can also be used to choose between different off-the-shelf software products with similar functionality with regard to the above mentioned criteria.

When software is purchased and hosted by an EAU internally, a resource profile supports the *internal cost accounting* between different business units and the IT unit (Brandl/ Bichler/Ströbel, 2007) (see 2. in figure 10.13). Using resource profiles, the hardware resource consumption and operational costs of an enterprise application can be broken down to user profile levels or transaction classes. Brandl/Bichler/Ströbel (2007) showed how a breakdown of the resource consumption on the level of transaction classes can be used to allocate costs to different business units according to their workload.

If an EAU does not want to host an application himself, resource profiles can be used as a basis for a *contract* between an EAU and an EAH (e.g., cloud providers) (see 3. in figure 10.13). As cloud computing is gaining popularity, the demand for usage-based costing will increase (Brandl/Bichler/Ströbel, 2007). Similar to the internal cost accounting approach explained above, EAUs and EAHs could agree on a *remuneration model* which is directly dependent on the resource consumption and operational costs of the hosted application (Li/Casale/Ellahi, 2010). Resource profiles help both parties to better estimate their costs in such a scenario.

If an enterprise application is already running in a production environment, resource profiles help in the *capacity management* process (King, 2004) (see 4. in figure 10.13). For

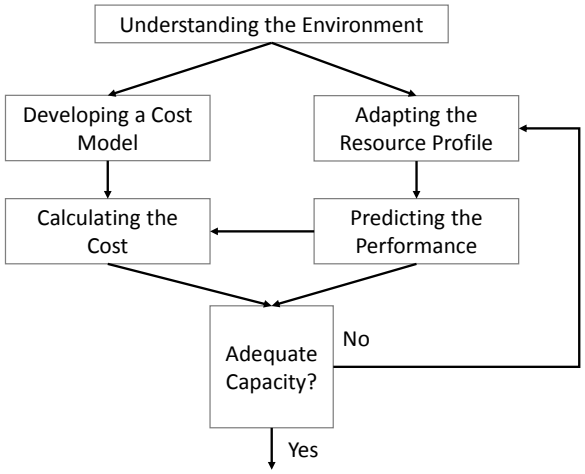



**Figure 10.14:** *Capacity planning process (adapted from Menascé/Almeida* (2002) *p. 177-179)*

example, the impact of an increased user load on performance and operational costs of an application can be examined beforehand to draw appropriate conclusions.

### *10.5.2 Capacity Planning Process*

A simplified capacity planning process adapted from Menascé/Almeida (2002) using resource profiles for the purpose of capacity planning is depicted in figure 10.14. This process assumes that a resource profile is available for an EA for which capacity planning needs to be performed. This assumption is based on the fact that resource profiles for each EA version are already created as outlined in section 10.4. Once an EA version is released, these resource profiles are assumed to be distributed along with the application binaries as shown in figures 10.1 and 10.13.

As a first step in this capacity planning process, it is necessary to understand the environment in which the capacity planning needs to take place to define the corresponding constraints. To describe these constraints, the following parameters for a capacity planning process can be set, namely (Menascé/Almeida, 2002):

- Expected workload (i.e., user behavior and count)
- Performance requirements (e.g., service-level agreements)
- Technology constraints (especially regarding the hardware environment)
- Cost constraints

Afterwards, two steps can be performed in parallel: the resource profile can be adapted to the target environment and a cost model can be created. In a next step, predictions based on the resource profile can be used to estimate if the performance constraints for the given workload and hardware environment can be met. Once the performance results

are available and the performance-related constraints are met, a cost model can be used to determine if the cost constraints are met as well. These steps are outlined in the next sections.

### *10.5.3 Adapting Resource Profiles to Different Hardware Environments*

In order to be able to predict performance for a different hardware environment than the one from which the resource profile was created, a hardware environment model needs to be adapted from the initial to the target one. In order to adapt the resource profile from one environment to another, the processing rate of resources must be scaled according to the performance of the hardware resources.

Following Menascé/Almeida (2002), the processing rate of a resource is scaled according to the hardware benchmark results of the initial and target hardware resources. After performing a suitable benchmark on the initial and the target server, we assume to get two benchmark scores of the investigated hardware resource. We furthermore assume that these benchmark scores are numerical and that higher benchmark score values indicate better performance. The benchmark score of the initial ($b_{initial}$) and target ($b_{target}$) server and the initial processing rate ($r_{initial}$) allow a calculation of the new processing rate ($r_{target}$) for the target server's resource as follows:

$$r_{target} = \frac{b_{target}}{b_{initial}} * r_{initial} \tag{10.4}$$

For CPU benchmarks it is important that the benchmark can evaluate the performance of a single core; otherwise, it is more difficult to adapt the resource environment model from one server to another if the amount of CPU cores needs to be modified. If standardized benchmarks are used for this purpose, the benchmarks must not necessarily be performed by the user of a resource profile as results for common hardware systems are often available on the web sites of the benchmark providers. Nevertheless, processing rates and benchmark scores of hardware resources used to derive resource demands need to be available to the user of a resource profile.

This approach assumes that all resource demands in the RDSEFFs of a repository model are initially derived from measurements on the same hardware types. Otherwise, it would be necessary to adapt the resource demands in an RDSEFF individually if a component is moved from one server to another in the allocation model. Similarly, the network traffic between all components needs to be represented in a resource profile even if the profile is created on a single machine. Without this information, it would not be possible to distribute components in a resource profile to different machines connected by a network.

Another key assumption of this approach is that the execution platform of an EA does not change between two deployments (Kuperberg, 2010). For example, if the resource demands in a resource profile are measured on a specific middleware platform, it is not possible to easily predict the impact of a second or different middleware platform.

### *10.5.4 Developing Cost Models and Calculating the Cost*

After a hardware environment is found that meets the performance goals, the next step is to estimate associated cost. According to Menascé et al. (2004), the costs of an EA deployment include hardware costs (e.g., server machines, routers), software costs (e.g., operating system (OS) or middleware licenses), facility costs (e.g., leasing of physical space, energy consumption) and downtime costs (e.g., lost income from missed business opportunities). Furthermore, penalties for violations of service-level agreements (SLA) might occur. Therefore, not only initial costs but also the total cost of ownership (TCO) in the long term need to be considered.

Apart from the downtime costs and the SLA penalty charges, total costs can be determined by specifying a function relative to the amount of hardware resources required to achieve performance goals. For example, if two servers with eight CPU cores each and a network router to connect them are required to handle the workload, their individual price tags can be used to calculate the hardware costs. The corresponding software costs can be determined depending on the license model: e.g., if license fees need to be paid per CPU core and year, a simple multiplication of the fee per core by sixteen would result in the software costs per year. Management and support costs can also be allocated per server or a set of servers in a rack as outlined by Patel/Shah (2005). A simple way to also distribute facilities cost would also be to take the space requirements per server or rack into account. More advanced cost assessments may include the exact size of a server or their power consumption.

If an EA is intended to be hosted externally, the cost model might be less complicated from an EAU perspective, as some of these costs are abstracted by an EAH. Therefore, hardware, facility and parts of the software and management costs might be represented by a simple monetary value per rented server instance. However, the basic idea is the same and an EAH still faces the task of calculating the required capacity for an EA that is to be hosted in his environment.

Therefore, as soon as a minimal hardware environment is found by using resource profile predictions, and it guarantees the performance requirements in order to avoid SLA penalty charges or other problems, its content in terms of servers and other hardware resources can be used as input for the corresponding cost model as shown by Koziolek (2013).

Because the cost of the energy consumption is one of the key challenges in enterprise data centers (Poess/Nambiar, 2008), we extended the PCM resource environment in our work (Willnecker/Brunnert/Krcmar, 2014b) to allow for energy consumption predictions. Using this extension during predictions allows to approximate the energy consumption of an EA along with the other performance metrics for a given hardware environment. The predicted energy consumption can therefore be used as additional input for a cost model.

## 10.6 Evaluation

This section assesses the continuous performance evaluation and capacity planning approaches using resource profiles introduced in this work. The evaluation of these approaches and their associated claims is performed in three integrated controlled experiments (Hevner et al., 2004); each experiment validates an approach or claim presented in this work.

All three experiments are conducted with an application provided by the SPECjEnterprise2010[8] benchmark called Orders domain; used here as a representative enterprise application. The advantage of using a benchmark application is that SPECjEnterprise2010 specifies a workload and a dataset for tests using this application. The results are therefore reproducible for others[9]. The Orders domain application is a Java Enterprise Edition (EE) web application and comprises Servlet, JavaServer Pages (JSP) and Enterprise JavaBean (EJB) components. The Orders domain application is used by automobile dealers to sell and order cars. The dealers interact with the Orders domain application over the hypertext transfer protocol (HTTP). In the following, we refer to the automobile dealers as users.

The first experiment evaluates the feasibility of the continuous performance evaluation approach. For this purpose, a continuous delivery pipeline for building this application is extended by integrating the performance change detection process proposed in section 10.4. Two application versions are built, the second one contains a performance regression. By executing the performance change detection process during the delivery pipeline, we expect to detect the regression as well as the reason for this regression.

In a second experiment, we evaluate the claim that the workload for a resource profile can be modified independently of the resource profile content and the hardware environment. For this purpose, we use the resource profile for the Orders domain application version without a regression as built in the first experiment. The resource profile and hardware environment is not changed but its workload is modified. Predictions using the modified workload are compared against corresponding measurements.

The capacity planning approach is validated in a third experiment. We assume that the resource profile for the Orders domain application version without a regression is released along with the corresponding application binaries. This resource profile is then used to support the capacity planning process for a hardware purchasing scenario. In this scenario, a decision needs to be made as to which one of two different hardware environments should be purchased for the Orders domain application. For that purpose, we follow the capacity planning process outlined in section 10.5. The existing resource profile for the Orders domain application created within the deployment pipeline is adapted to both environments in question and used to predict performance and energy consumption. Additionally, a cost

[8]SPECjEnterprise is a trademark of the Standard Performance Evaluation Corp. (SPEC). The SPECjEnterprise2010 results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The results in this publication should thus be seen as estimates as the benchmark execution might deviate from official run rules. The official web site for SPECjEnterprise2010 is located at `http://www.spec.org/jEnterprise2010`.

[9]The raw data of the evaluation results are available for download on the following website: `http://pmw.fortiss.org/research/jss/`

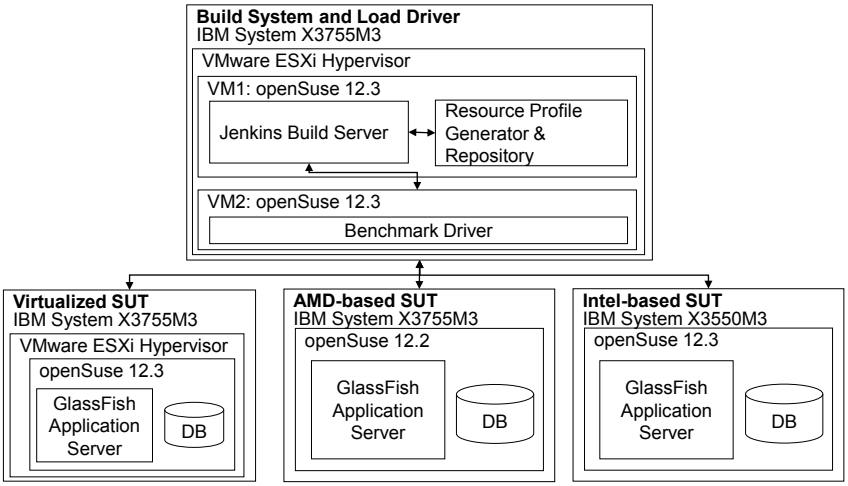



**Figure 10.15:** *Experiment setup*

model is created and used to calculate the cost of the hardware environments based on resource profile predictions. As a result, the resource profile capabilities to support capacity planning and the claim that the hardware environment can be modified independently from the resource profile and workload are validated.

The following section 10.6.1 describes the experiment setup used for this evaluation. Section 10.6.2 introduces a data collection approach that allows the collection of required data for generating PCM-based resource profiles for Java EE applications. Section 10.6.3 describes the evaluation of the performance change detection within a deployment pipeline. Lastly, section 10.6.4 evaluates the feasibility of modifying the workload of a resource profile and section 10.6.5 evaluates the capacity planning approach outlined in this work.

### *10.6.1 Experiment Setup*

The experiment setup used for the evaluation in this work is shown in figure 10.15. The setup comprises three deployments of the Orders domain application and a system to execute the tasks within a deployment pipeline. The application is deployed on three different hardware environments: a virtualized environment, an AMD-based hardware environment and an Intel-based hardware environment. We call these systems under test (SUT) virtualized SUT, AMD-based SUT and Intel-based SUT. The exact hardware and software configurations for these three SUTs are shown in table 10.2.

An additional system called ”build system and load driver” is part of the experiment setup. A key element of this system is a Jenkins build server that implements a deployment pipeline for the Orders domain application according to the one shown in figure 10.1. In a first step, the Jenkins server automatically builds the Orders domain application. Afterwards, automated acceptance tests are executed. Once the acceptance tests are completed, a resource profile for the Orders domain application is created based on data collected during the tests and then stored in an artifact repository. The other steps of the deployment pipeline are not automated.

| **Hardware Environment** | **Virtualized SUT** | **AMD-based SUT** | **Intel-based SUT** |
|---|---|---|---|
| **Application** | SPECjEnterprise2010 (version 1.03) Orders domain | | |
| **Application Server** | GlassFish 4.0 (build 89) | | |
| **Database** | Apache Derby DB | | |
| **Java Virtual Machine** | 64 bit Java HotSpot Virtual Machine (JVM) version 1.7.0 (build 24.71-b01) | | |
| **Operating System (OS)** | openSuse 12.2 | | openSuse 12.3 |
| **Virtualization** | VMware ESXi (5.1.0)<br>4 Cores and 40 GB RAM | | |
| **CPU Cores** | 48 x 2.1 GHz | 6 x 2.1 GHz | 4 x 2.4 GHz |
| **CPU Sockets** | 4 x AMD Opteron 6172 | | 2 x Intel Xeon E5645 |
| **Random Access Memory (RAM)** | 256 GB | | 96 GB |
| **Hardware System** | IBM System X3755M3 | | IBM System X3550M3 |
| **Network** | 1 gigabit-per-second (GBit/s) | | |

**Table 10.2:** *Software and hardware configuration of the systems under tests*

For the purpose of executing acceptance tests, the Jenkins build server uses a so called benchmark driver. This driver is provided by the SPECjEnterprise2010 benchmark and implements the behavior of the users of the Orders domain application. They, in turn, execute several HTTP requests on the system which are combined into three business transactions: browse, manage and purchase. These tests are not acceptance tests in the traditional sense; they exercise the system in a way a normal user would and are, thus, comparable.

Additionally, the build system orchestrates a resource profile generation component that implements the transformation process introduced in section 10.3.2 and a corresponding resource profile repository based on the EMFStore as outlined in section 10.4.2. The Jenkins build server and the benchmark driver for the SPECjEnterprise2010 deployments are deployed individually on different virtual machines (VM). The VM for the Jenkins server is called VM1, whereas the VM for the benchmark driver is called VM2[10]. Both VMs are deployed on a hardware server similar to the one of the virtualized SUT. This server is again connected via one gigabit-per-second network connections to the SUTs.

To collect the required data for generating a resource profile for a Java EE application, we use a data collection approach introduced in our previous works (Brunnert/Vögele/Krcmar, 2013; Brunnert/Neubig/Krcmar, 2014). We decided not to rely on external measurement tools for this purpose as most of the tools do not provide sufficient detail for the creation of a resource profile or prohibit the publication of instrumentation overhead evaluations.

### *10.6.2 Collecting Resource Profile Data for Java EE Applications*

The Java EE specification (Shannon, 2006) defines the component types included in an application and a runtime environment for hosting Java EE applications that is consistently available across Java EE server products. Leveraging this standardization, the suggested data collection approach is designed so that it can be applied for Java EE applications running on all Java EE server products compliant with the specification. An overview of the places and technologies used to collect the required data for creating resource profiles for Java EE applications is shown in figure 10.16.

[10]In order to be able to execute the evaluation in parallel three copies of these VMs on similar hardware servers are used.

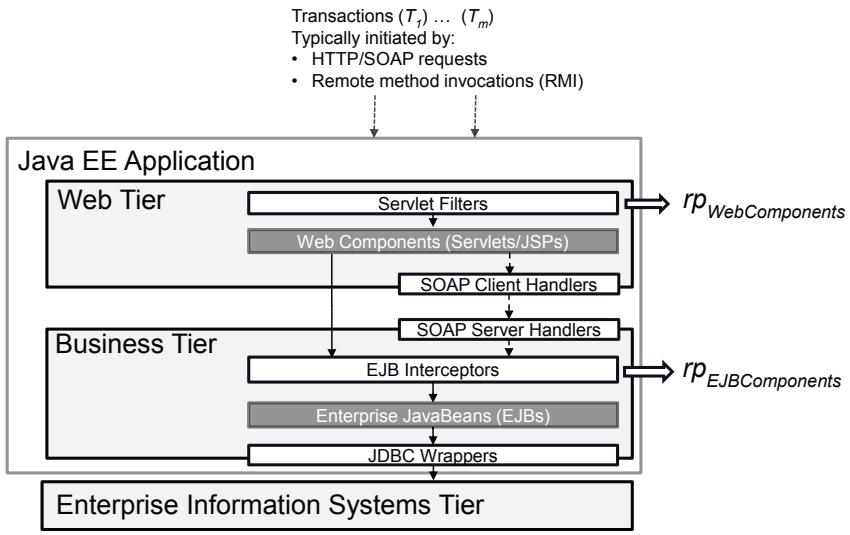



**Figure 10.16:** *Java EE data collection - transaction processing interception techniques*

The main Java EE application component types are Applets, Application Clients, Enterprise JavaBeans (EJB) and web components (i.e., JavaServer Pages (JSP) and Servlets) (Shannon, 2006). As Applets and Application Clients are external processes which do not run within a Java EE server runtime, this paper focuses on EJB and web components. To collect the required data for creating resource profiles using the structure outlined in section 10.3.1, several standardized technologies are used that are capable of intercepting the control flow of transactions within a Java EE system. Starting from top to the bottom, these are: ServletFilters to collect the required data for web components, EJB interceptors for EJBs and Java database connectivity (JDBC) Wrappers to transfer information about the current transaction to external databases. SOAP Handlers are used to intercept calls to EJB components that are used as web services.

The interception technologies for web and EJB components basically use the same approach to collect the required data. These interception technologies are executed before and after a component operation and are able to collect the resource demand for the current thread executing a transaction. The simplified code for these interception techniques is shown in listing 10.1.

As a first step in listing 10.1, the code retrieves an existing *transactionID* or creates a new one. A key requirement is the ability to uniquely identify a transaction for each data sample collected: if this does not occur, the resource demands for single component operations cannot be associated with a specific transaction. Therefore, the *transactionID* is not only propagated within a Java Virtual Machine (JVM) but also across JVMs or to external systems. This propagation is the main purpose of the JDBC Wrappers and SOAP handlers. The SOAP handlers can resume transactions if they receive a *transactionID* in the header of a SOAP message which allows the tracking of transactions across different systems in a distributed architecture.

After the transaction context is established through receipt of the correct *transactionID*, the resource demand values are collected. First, the request is analyzed to calculate the size of the request received that led to the component invocation. If a *Content-Length* parameter is set in the request, the request size can be read directly and does not need to

be calculated. The resulting value is used to calculate the incoming network demand for the current component operation. The calculation of the network demand is possible for web components and EJB components called through SOAP handlers.

**Listing 10.1:** *Basic interception logic (adapted from Brunnert/ Vögele/ Krcmar (2013))*

```
public class PerformanceMonitoringFilter implements Filter {
 public void doFilter(request, response, chain){
  String transactionID = getTransactionID();
  ResourceDemand dNETi = getRequestSize(request);
  ResponseWrapper responseWrapper = new ResponseWrapper(response);
  ResourceDemand startRD = getCurrentThreadResourceDemand();
  chain.doFilter(request, responseWrapper);
  ResourceDemand stopRD = getCurrentThreadResourceDemand();
  ResourceDemand dNETo = responseWrapper.getResponseSize();
  storeDemand(transactionID, startRD, stopRD,dNETi,dNETo);
 }
}
```

The CPU time, HDD demand and memory consumption in bytes for a component operation are collected by calling the *getCurrentThreadResourceDemand()* method before and after a component operation. Internally, the CPU time and memory consumption are measured using the *ThreadMXBean* interface of the Java Virtual Machine (JVM)[11]. HDD demand measurements are only supported on Linux and the implementation accesses the proc file system[12] to collect the required data for the current thread. The accuracy of the CPU demand measurements using the *ThreadMXBean* are dependent on the timer accuracy of the underlying operating system as discussed by Kuperberg (2010). Even though most Unix-based operating systems such as Linux provide an accuracy on nanosecond level, the Windows operating system only provides an accuracy of 15 millisecond intervals. In light of this restriction, we are leveraging the *QueryThreadCycleTime* operating system interface on Windows to collect more accurate measurements[13].

The response is then analyzed to calculate the outgoing network demand. The response size is calculated using a wrapper around the response object, already inserted before the request processing starts and counts all bytes written to the response during the request processing. The *storeDemand()* method at the end of the listing stores all values and associates them with the component and operation name as well as its deployment unit.

The data collected by these interception technologies can be stored in different ways. In our previous works, we used files (Brunnert/Vögele/Krcmar, 2013) and a Java-specific technology called MBeans (Brunnert/Neubig/Krcmar, 2014) which we will use for storing the data for this evaluation as we are in a Java-only environment.

If resource profile data needs to be collected for non-Java EE systems such as databases or web service providers based on different technology stacks, additional data collection approaches are required. To avoid the need to describe additional data collection approaches

[11] https://docs.oracle.com/javase/7/docs/jre/api/management/extension/com/sun/management/ThreadMXBean.html

[12] https://www.kernel.org/doc/Documentation/filesystems/proc.txt

[13] https://msdn.microsoft.com/en-us/library/windows/desktop/ms684943%28v=vs.85%29.aspx

and to simplify the setup, the database in the experiment setup outlined in section 10.6.1 is included in the Java EE runtime. Therefore, resource demands of database transactions are already included in the measurements of other Java EE components.

The evaluations in the following sections focus on network and CPU demands of the SPECjEnterprise2010 Orders domain application deployments as collecting memory consumption and HDD demand data would produce too much overhead for the HTTP requests issued by the benchmark driver with a response time of less than ten ms in most of the cases as shown in our previous work (Brunnert/Neubig/Krcmar, 2014). A workaround to incorporate these demands would be to separate measurement runs for CPU timings and network as well as for memory and HDD demands. However, the prototype implementing the transformation of the collected resource profile data into PCM models does not currently support this workaround.

### *10.6.3 Evaluating the Continuous Performance Evaluation*

This section evaluates the performance change detection process within a deployment pipeline as explained in section 10.4. For executing the tests in this evaluation, only the virtualized SUT within the experiment setup outlined in section 10.6.1 is used. We chose a virtualized environment for this evaluation because it is more likely that such systems are used for acceptance tests rather than fully equipped production systems such as the AMD- and Intel-based SUTs used in the evaluation of the capacity planning capabilities.

The Jenkins server on the build and test system automates the first steps of the deployment pipeline shown in figure 10.1, namely, the building of a new application version, the execution of an acceptance test and the creation and versioning of a resource profile. All other steps are not automated. The evaluation steps using this system are as follows:

1. The automated steps of the deployment pipeline are executed for the standard version of the Orders domain application as outlined in section 10.6.3.1.
2. The Orders domain application is modified and the automated steps of the deployment pipeline are triggered again (section 10.6.3.1). In this version, the performance characteristics of an application component are modified by increasing its resource consumption.
3. Before performance changes between both application versions are identified in section 10.6.3.3, the prediction accuracy of both resource profiles is evaluated in section 10.6.3.2. To do that, prediction results of both resource profiles are compared with measurements of their corresponding application versions.
4. We use the resource profiles of both versions to identify performance change in section 10.6.3.3. These resource profiles are used to predict response times for predefined workloads and one hardware environment. The expected result is an observed increase of response time and thus a regression in performance. To identify the root cause of increase in response time, the resource profile versions are compared directly to see if the change introduced in the second application version is visible in this comparison.

#### 10.6.3.1 Creating and Versioning Resource Profiles

Once the standard version of the SPECjEnterprise2010 Orders domain application is being built and deployed on the instrumented test system, acceptance tests are executed with 500 concurrent users for sixteen minutes whereby data is only collected between a five minute ramp up and a one minute ramp down phase. Once the test is completed, a command line utility is triggered by the build system which creates a PCM-based resource profile representation for this application version based on the collected data (see section 10.3.2) and stores it in the EMFStore.

In a second step, an updated version of the Orders domain application is being built and deployed on the test system. The updated version is modified so that one component of the application consumes more CPU time and sends more data over the network than its original version. Another test run with 500 concurrent users is then executed using the updated version. Afterwards, a new version of the resource profile is generated and stored in the EMFStore.

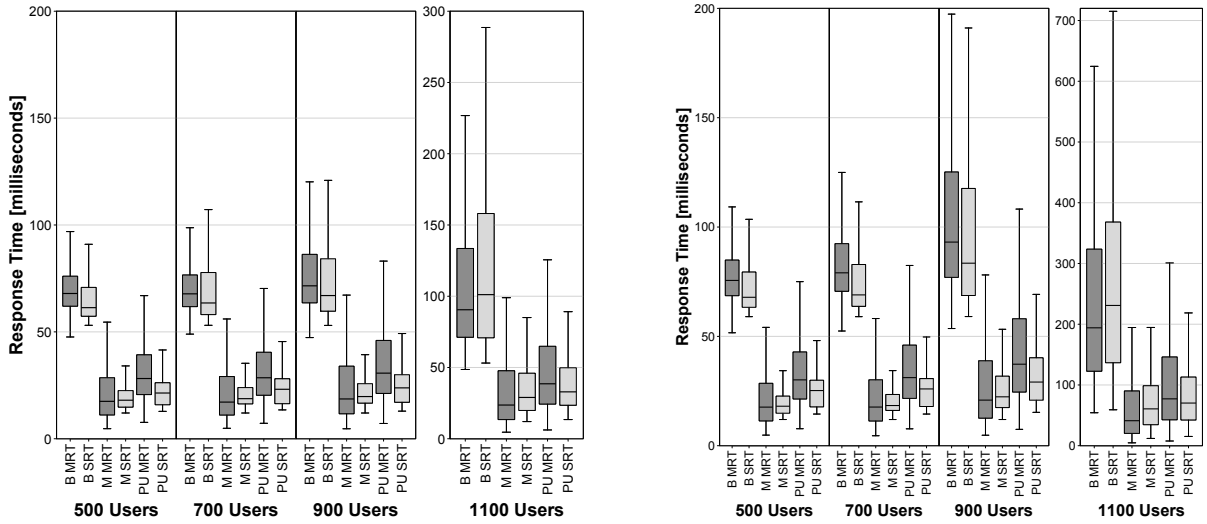
The resource profile for the original Orders domain application is hereafter called resource profile version one and the second resource profile for the modified application is hereafter called version two. To evaluate performance for both application versions, usage and hardware environment models are predefined. The usage model is created following the source code of the test scripts in the benchmark driver. The hardware environment models represent the virtualized SUT and the benchmark driver. In order to represent the network connection between the benchmark driver and the SUT, the lmbench[14] benchmark suite is used to derive the required network latency and bandwidth values for parameterizing the resource environment (see section 10.3.4). The bandwidth is measured using the *bw_tcp* benchmark of the lmbench suite, whereas the latency is calculated as an average of the *lat_connect* and *lat_tcp* results. The combination of the two latency values is necessary to reflect the latency for transmitting data and establishing a connection in the one latency value specified in a PCM linking resource. The latency is subsequently used by the PCM simulation engine SimuCom for incoming and outgoing data transmissions of the same connection; as a result, the latency value is divided by two.

#### 10.6.3.2 Evaluating the Accuracy of Resource Profile Predictions

To evaluate the prediction accuracy of the generated resource profile versions, they are used to predict the performance of the corresponding application versions under different workload conditions. The same workloads are executed as test runs using the corresponding application versions. Afterwards, the measured results are compared with the predicted results. This comparison includes the response time and throughput of the business transactions browse (B), manage (M) and purchase (PU) as well as the CPU utilization of the virtualized SUT.

The workload for this comparison is increased from 500 to 1100 concurrent users (~40 % to ~96 % CPU utilization). To predict the performance of the application versions, the

[14] `http://www.bitmover.com/lmbench/`




(a) Application and resource profile version one

(b) Application and resource profile version two

**Figure 10.17:** *Measured and simulated response times*

corresponding resource profiles including the workload and hardware environment models are used as input for the simulation engine SimuCom (Becker/Koziolek/Reussner, 2009). SimuCom performs a model-to-text transformation to generate Java code based on PCM models. This Java code is afterwards executed to start a simulation. The simulation duration is set to sixteen minutes whereby only data between a five minute ramp up and one minute ramp down phase are used for the calculation of the simulation results.

The simulation results for the resource profile versions one and two are shown in boxplot diagrams[15] in figure 10.17 and in tables 10.3 and 10.4. Response time measurement and simulation results are described using median values as well as measures of dispersion, namely the quartiles and the interquartile range (IQR). Mean and standard deviation values are excluded from our investigation due to the skewness of the underlying distributions (Jain, 1991) of the response times of browse, manage and purchase. For each load condition specified by the number of users, figures 10.17(a) and 10.17(b) show the dispersion of the simulated response time (SRT) per business transaction. The simulated throughput (depicted as transactions per minute (TPM)) and the simulated mean CPU utilization (SMCPU) can be found in tables 10.3 and 10.4 respectively.

In the same way as the simulations, the tests run sixteen minutes and the data for this comparison are collected in a steady state between a five minute ramp up and a one minute ramp down phase. The measured mean CPU utilization (MMCPU) of the virtualized SUT is measured using a Java client that continuously collects the CPU utilization of the GlassFish process on the SUT during a test run. The measured response time and throughput values are directly taken from measurements on the benchmark driver. The driver has been modified to enable measurements of individual transactions as it only provides aggregated measurements in its default form. These measured response time

[15]Boxplot diagrams comprise a box whose bounds denote the first quartile $Q_1$ (lower bound) as well as the third quartile $Q_3$ (upper bound) of the underlying data sample. The quartiles are connected by vertical lines to form the box indicating the IQR which is defined as $Q_3 - Q_1$. Furthermore, the median $Q_2$ is illustrated by a horizontal line within the box, thus separating it into two parts. Vertical lines outside the box (whiskers) indicate the range of possible outliers; their length is limited to 1.5 times the $IQR$.

| | | Application and resource profile version one | | | Application and resource profile version two | | |
|---|---|---|---|---|---|---|---|
| **Users** | **T** | **Measured TPM** | **Simulated TPM** | **Prediction error** | **Measured TPM** | **Simulated TPM** | **Prediction error** |
| 500 | B | 1515.70 | 1514.80 | 0.06 % | 1515.40 | 1524.80 | 0.62 % |
| | M | 763.30 | 762.10 | 0.16 % | 749.80 | 760.60 | 1.44 % |
| | PU | 748.40 | 769.80 | 2.86 % | 753.60 | 757.20 | 0.48 % |
| 700 | B | 2095.10 | 2117.60 | 1.07 % | 2155.40 | 2136.40 | 0.88 % |
| | M | 1067.00 | 1082.00 | 1.41 % | 1043.30 | 1055.80 | 1.20 % |
| | PU | 1062.60 | 1065.80 | 0.30 % | 1053.10 | 1072.40 | 1.83 % |
| 900 | B | 2716.60 | 2701.20 | 0.57 % | 2722.30 | 2721.20 | 0.04 % |
| | M | 1342.20 | 1382.20 | 2.98 % | 1354.10 | 1377.70 | 1.74 % |
| | PU | 1350.40 | 1398.50 | 3.56 % | 1366.90 | 1372.60 | 0.42 % |
| 1100 | B | 3315.20 | 3357.00 | 1.26 % | 3306.10 | 3322.20 | 0.49 % |
| | M | 1661.10 | 1659.40 | 0.10 % | 1656.00 | 1633.40 | 1.36 % |
| | PU | 1652.80 | 1661.60 | 0.53 % | 1650.90 | 1656.70 | 0.35 % |

**Table 10.3:** *Measured and simulated throughput*

| | Application and resource profile version one | | | Application and resource profile version two | | |
|---|---|---|---|---|---|---|
| **Users** | **MMCPU** | **SMCPU** | **CPUPE** | **MMCPU** | **SMCPU** | **CPUPE** |
| 500 | 44.75 % | 39.29 % | 12.18 % | 48.71 % | 43.81 % | 10.05 % |
| 700 | 55.92 % | 54.96 % | 1.72 % | 63.33 % | 61.41 % | 3.03 % |
| 900 | 67.35 % | 70.34 % | 4.45 % | 78.15 % | 78.44 % | 1.67 % |
| 1100 | 80.14 % | 86.65 % | 8.12 % | 92.62 % | 95.40 % | 3.00 % |

**Table 10.4:** *Measured and simulated CPU utilization*

(MRT) values for the business transactions are shown in the boxplot diagrams in figures 10.17(a) and 10.17(b). The measured throughput and the MMCPU values can be found in tables 10.3 and 10.4 respectively.

The comparison of the simulated and measured results shows that both resource profiles do represent their corresponding application versions very well. The relative response time prediction error for the median response time values is at most 24.07 % for both resource profile versions. The prediction error for the median increases to 47.38 % only for the manage transaction in the second resource profile version and a load of 1100 concurrent users. The reason for this difference is that the low absolute median values for the manage transaction are quite sensitive to changes. As the load level of 1100 users in the second resource profile causes a CPU utilization of 95.40 % in the simulation compared to 92.62 % on the real system (see table 10.4), additional queuing effects increased the manage median response time by 19 ms thus causing this difference.

The skewness of a business transaction's underlying response time distribution can be determined considering the median's position between the quartiles $Q_1$ and $Q_3$. The results show that the skewness of the underlying distribution is not always correctly represented. This is especially the case for manage, as the first quartile $Q_1$ is predicted by the simulation with a relative error of up to 71 %, which is already caused by an absolute error of 14.4 ms. The first quartile $Q_1$ for the browse and purchase transactions is represented with a relative error of at most 23 %. The third quartile $Q_3$ is predicted with a relative error of largely below 30 % for all transactions.

The relative throughput prediction error is at most 3.56 % (see table 10.3). This validates the results but is expected as the think time for each user is much higher (9.8 seconds) than the response time measurement and simulation results shown in figures 10.17(a) and 10.17(b). The impact of response time prediction errors on the throughput is thus very low.

The relative CPU utilization prediction error (CPUPE) is at most 12.18 % (see table 10.4). The simulated CPU utilization is below the measured CPU utilization in low load levels (500 and 700 concurrent users), as the garbage collection overhead and other JVM background activities are not represented in the resource profiles (Brunnert/Vögele/Krcmar, 2013; Brunnert/Wischer/Krcmar, 2014). The resource demands for the background activities are missing because resource profiles only contain data for threads processing user transactions and not for other threads in the system dealing with the aforementioned activities. In high load conditions (900 and 1100 concurrent users) the simulated values are slightly higher than the measured values. The data collection overhead included in the resource demands of the model elements (Brunnert/Vögele/Krcmar, 2013) thus seems to balance the influence of aspects not represented in the resource profiles in high load conditions.

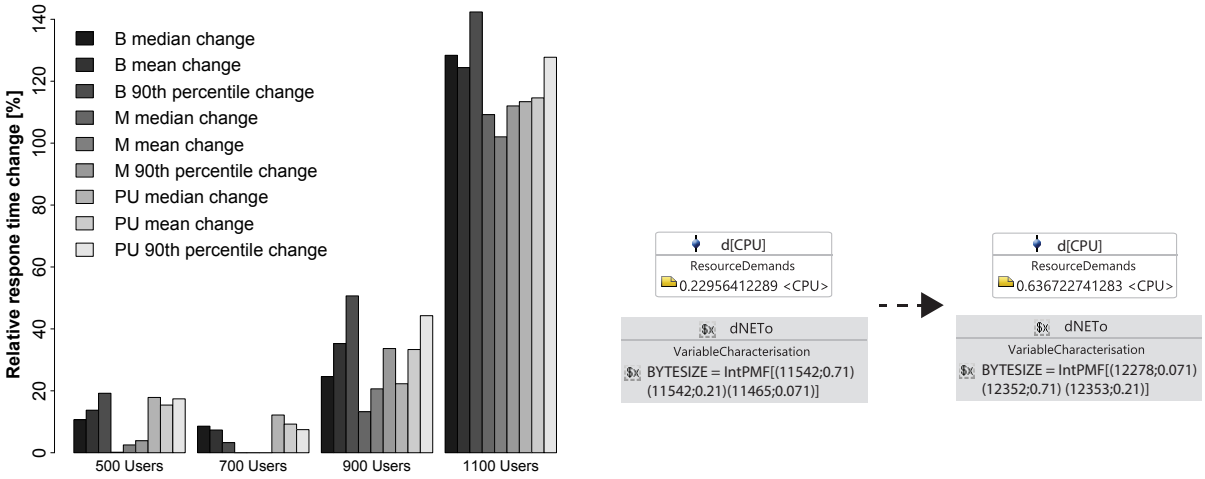
#### 10.6.3.3 Comparing Prediction Results and Resource Profile Versions

The prediction results of the two resource profile versions are now compared with each other to identify performance change. As a first step, the two-sample Kolmogorov-Smirnov test is executed for each business transaction in order to identify if a change is statistically significant. Except for the manage transaction in the case of 700 concurrent users, a statistically significant change in response time is identified for all transactions and load levels between both versions.

The response times of all investigated transactions increased from version one to two (figure 10.18(a)). Furthermore, the browse and purchase transaction response time increased considerably in all load levels. The mean, median and 90th percentile values for these transactions increased up to 140 %. Therefore, the results show a clear regression for these transactions. This is expected, as the component that was changed is involved in the control flow of both transactions.

The comparison results (figure 10.18(a)) also indicate that the change in the new EA version leads to a tremendous increase of response times for all transactions at the highest load level in the case of 1100 users. The reason for this increase is the CPU utilization of 95.40 % at this load level (see table 10.4). The CPU is constantly busy and is now a major bottleneck, although the same amount of CPU cores could handle this load in the previous application version. Therefore, the change included in this EA version causes not only a performance regression in terms of transaction response times but also reduces the scalability of the overall EA.

In order to identify the reason for this change, we accessed the EMFStore containing both resource profile versions and looked at their differences using EMF Compare. Two main comparison results are shown in figure 10.18(b). This figure shows, that the CPU resource demand of one component operation (0.64 ms) is nearly three times higher than the one of the previous version (0.23 ms). Additionally, the network demand of the same component operation increased slightly, by about seven percent. As this component operation is involved in the control flow of the browse and purchase transactions and all other resource demands did not change considerably between the two resource profile versions, this




(a) Relative change between both resource profile versions

(b) EMF Compare results

**Figure 10.18:** *Comparisons of prediction results and resource profile versions*

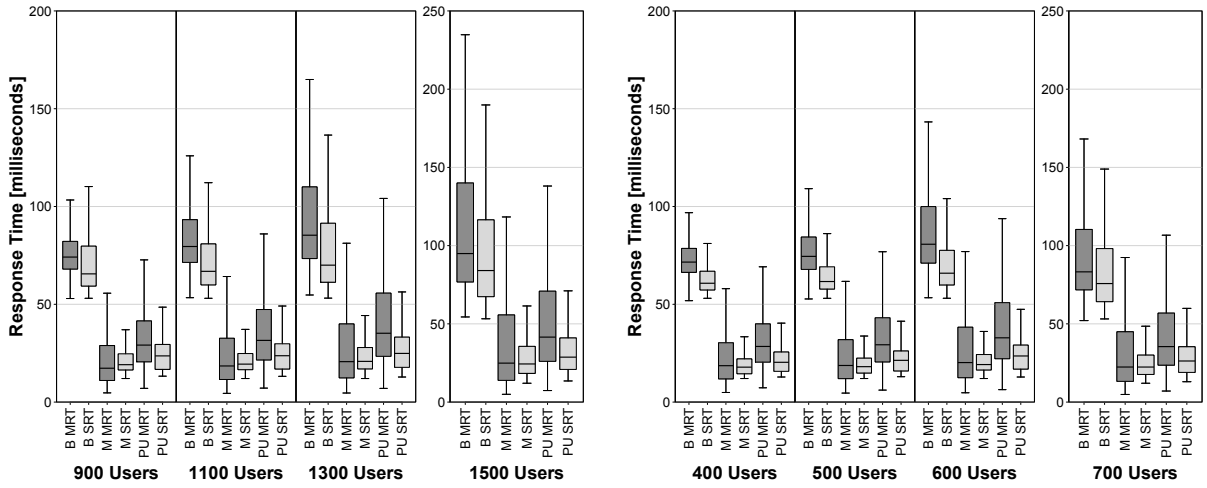
component is obviously the source of the change. Thus, a comparison of the resource profiles helped to identify the places causing the performance regression.

### *10.6.4 Evaluating Workload Changes*

In the previous section we evaluated the ability to detect performance changes in a deployment pipeline using resource profiles. During this evaluation, we have only used an increased number of users as a change in workload. As one of the claims of our resource profile definition is that we can change the workload and hardware environment independently from the resource profile content, we evaluate additional workload changes before we evaluate hardware changes as discussed in the next section. For the evaluations in this section we use the resource profile version one of the previous evaluation. The hardware environment representing the virtualized SUT is not changed.

As a first step, we change the user behavior by changing the transaction distribution from 50 % browse, and 25 % purchase and manage respectively to a distribution where a user executes browse only in 20 % of the cases and purchase and manage in 40 % respectively. This user behavior change is called workload change one in the following paragraphs, figures and tables. As this user behavior produces less load on the system, we are able to increase the number of users further to 1500 concurrent users on the virtualized SUT. In order to evaluate the accuracy of resource profile predictions, simulation results based on this modified workload are compared with corresponding measurements. As in the previous evaluation, only measurement and simulation results during a ten minute steady phase between a five minute ramp-up and a one minute ramp-down phase are considered. The measurements are also executed similarly and are compared with the simulation results afterwards.

The evaluation results are shown in figure 10.19(a), displaying response time, table 10.5 representing throughput and table 10.6 showing the CPU utilization prediction accuracy.




(a) Workload change one

(b) Workload change two

**Figure 10.19:** *Measured and simulated response times*

The median response times are predicted with a relative error of at most 30.84 % compared to the measurement results. The first and third quartiles are also predicted with an error of largely below 30 %. The notable exceptions are the first quartile for the manage transaction which is predicted with relative errors of up to 48.96 % (for a load of 900 users) and the third quartile of the purchase transaction with an error of 42.20 % (for a load of 1500 users). However, even the greatest relative error is caused by an absolute difference of 5.41 ms and is thus not very high. The response time prediction accuracy is slightly lower as in the previous evaluation. We attribute this difference to the fact that manage and purchase are transactions that write into a database. As the transaction isolation level is set to repeatable read, the increased number of database locks not represented in a resource profile are likely to cause this difference. The throughput is predicted with an error of at most 3.56 % and the CPU utilization with an error of at most 16.09 %.

As a second workload change, the think time of the users for the new user behavior is also modified. In this way, we evaluated the capability of evaluating different transaction distributions, think times and user counts for a resource profile created using one workload in a deployment pipeline. In this second workload change scenario, the think time is reduced by five seconds. As each user now executes nearly twice the amount of requests in the same time period, the amount of users that can be handled by the virtualized SUT is lower than in the other evaluations. In this scenario we evaluate a range of 400 to 700 users. This workload leads to a similar utilization of the SUT resources as the previous workload with up to 1500 users.

The results are shown in figure 10.19(b), displaying response time, table 10.5 representing throughput and table 10.6 showing the CPU utilization prediction accuracy. The response time prediction is still reasonable accurate as the median response times are predicted with a relative error of at most 28.50 % compared to the measurement results. The first and third quartiles are also predicted with an error of largely below 30 %. The most notable exception is the third quartile for the purchase transaction which is predicted with relative errors of up to 42.71 % (for a load of 600 users). The throughput is predicted with an error of at most 8.17 % and the CPU utilization with an error of at most 16.50 %. The

| Workload change one | | | | | Workload change two | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Users** | **T** | **Measured TPM** | **Simulated TPM** | **Prediction error** | **Users** | **T** | **Measured TPM** | **Simulated TPM** | **Prediction error** |
| 900 | B | 1082.70 | 1114.50 | 2.94 % | 400 | B | 959.70 | 1038.10 | 8.17 % |
| | M | 2197.30 | 2251.30 | 2.46 % | | M | 1926.90 | 2037.20 | 5.72 % |
| | PU | 2164.20 | 2241.10 | 3.55 % | | PU | 1953.00 | 1999.90 | 2.40 % |
| 1100 | B | 1344.10 | 1365.50 | 1.59 % | 500 | B | 1207.40 | 1252.40 | 3.73 % |
| | M | 2653.70 | 2739.90 | 3.25 % | | M | 2409.50 | 2536.90 | 5.30 % |
| | PU | 2648.40 | 2736.60 | 3.33 % | | PU | 2418.30 | 2554.80 | 5.64 % |
| 1300 | B | 2716.60 | 2701.20 | 0.57 % | 600 | B | 1441.50 | 1539.10 | 6.77 % |
| | M | 1342.20 | 1382.20 | 2.98 % | | M | 2898.90 | 3023.00 | 4.30 % |
| | PU | 1350.40 | 1398.50 | 3.56 % | | PU | 2885.10 | 3042.00 | 5.44 % |
| 1500 | B | 1574.80 | 1617.50 | 2.71 % | 700 | B | 1704.30 | 1738.80 | 2.02 % |
| | M | 3163.50 | 3249.20 | 2.68 % | | M | 3423.30 | 3561.80 | 4.05 % |
| | PU | 3124.10 | 3222.80 | 3.16 % | | PU | 3374.30 | 3561.80 | 5.56 % |

**Table 10.5:** *Measured and simulated throughput*

| Workload change one | | | | Workload change two | | | |
|---|---|---|---|---|---|---|---|
| **Users** | **MMCPU** | **SMCPU** | **CPUPE** | **Users** | **MMCPU** | **SMCPU** | **CPUPE** |
| 900 | 57.36 % | 48.13 % | 16.09 % | 400 | 52.74 % | 44.04 % | 16.50 % |
| 1100 | 67.79 % | 58.80 % | 13.26 % | 500 | 62.31 % | 54.40 % | 12.70 % |
| 1300 | 75.73 % | 69.65 % | 8.03 % | 600 | 71.71 % | 65.70 % | 8.38 % |
| 1500 | 83.18 % | 80.27 % | 3.50 % | 700 | 78.39 % | 75.88 % | 3.20 % |

**Table 10.6:** *Measured and simulated CPU utilization*

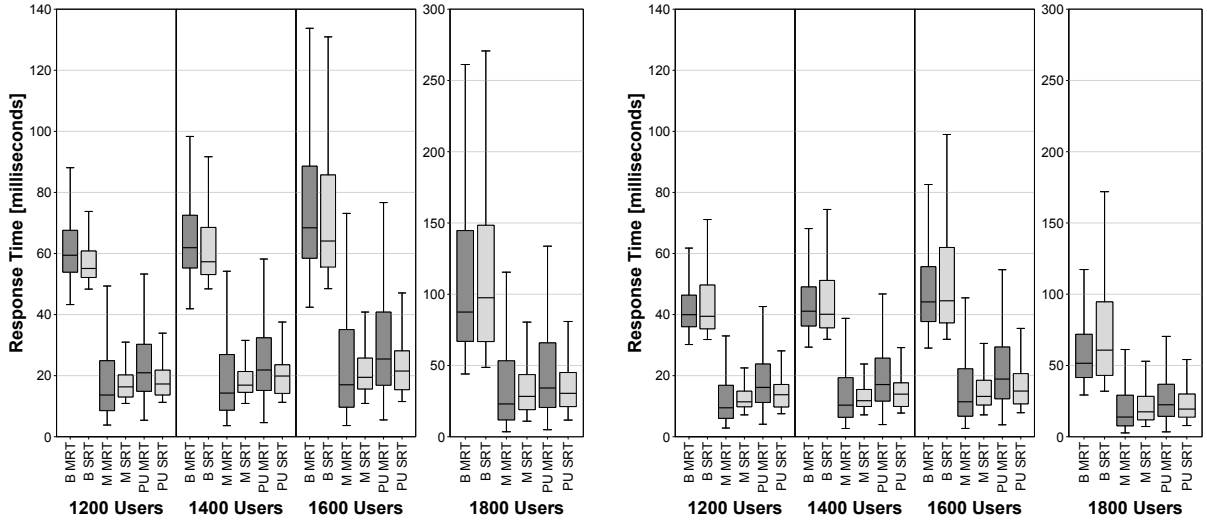
higher error for the throughput prediction is most likely caused by the benchmark driver allowing only to specify a cycle time whereas a think time is specified in the workload models. A cycle time specifies that a request is performed every n seconds, whereas a think time specifies a wait time of n seconds between two requests. As we simply reduced the think time by five seconds, this definition deviated slightly from the driver behavior. The behavior difference of think time versus cycle time therefore had a smaller impact when the think time was higher as was the case in the previous evaluations.

The two workload change evaluations in this section validate the claim that the workload can be changed independently from a resource profile and its hardware environment. The results show that predictions using modified workloads, still result in accurate results for response time, throughput and resource utilization.

### *10.6.5 Evaluating Capacity Planning Using Resource Profiles*

To evaluate the capacity planning capabilities using resource profiles outlined in section 10.5 and the claim that the hardware environment of a resource profile can be changed independently from the workload, we evaluate a hardware purchasing scenario for the SPECjEnterprise2010 Orders domain application. The scenario investigated is whether the AMD- or the Intel-based SUT in the experiment setup shown in figure 10.15 should be used for a production deployment of the Orders domain application. More specifically, which of these systems provides the required capacity in a more economic way.

Following the capacity planning process shown in figure 10.14 and outlined in section 10.5.2, our first step is to evaluate which performance can be achieved with the AMD- and Intel-based SUTs for the SPECjEnterprise2010 Orders domain application. For this purpose, we use the resource profile of the Orders domain application without a regression (version one) created in the deployment pipeline as explained in section 10.6.3. Afterwards, we




(a) AMD-based SUT (b) Intel-based SUT

**Figure 10.20:** *Measured and simulated response times*

develop a cost model for both systems and calculate the cost for each deployment. The results indicate which of these systems provides sufficient capacity in a more economic fashion taking hardware and energy costs as well as space requirements in a data center into account.

#### 10.6.5.1 Adapting the Resource Profile and Predicting Performance

As explained in section 10.5.3, resource profiles created using measurements contain resource demand values relative to the hardware environments on which they have been created. For this evaluation, they contain values specific for the virtualized SUT. To adapt this information for the AMD- and Intel-based SUTs, the processing rate of CPU cores represented in the resource environment model needs to be changed. Additionally, the number of CPU cores needs to be adapted (see table 10.2). SPEC CPU2006[16] is used to benchmark the CPU cores of all three SUTs. The benchmark comprises an integer (SPECint) and a floating point (SPECfp) benchmark. To calculate the adapted processing rate, the SPECint benchmark is executed on the virtualized, AMD- and Intel-based SUTs. The virtualized SUT achieved a benchmark score of 11.4, the AMD-based SUT a score of 12.0 and the Intel-based SUT a score of 18.91 for the SPECint_base2006 metric. By using equation 10.4, a processing rate of 1052.63 is calculated for CPU cores in the adapted resource environment model of the AMD-based SUT and a processing rate of 1658.77 for the Intel-based SUT. In order to adapt the network connection between the benchmark driver and both SUTs, we again used the lmbench to derive network latency and bandwidth values and adapted the resource environments accordingly.

[16]The SPEC CPU2006 results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The results in this publication should thus be seen as estimates as the benchmark execution might deviate from official run rules. The official web site for SPEC CPU2006 is located at `http://www.spec.org/cpu2006`.

| | | AMD-based SUT | | | Intel-based SUT | | |
|---|---|---|---|---|---|---|---|
| **Users** | **T** | **Measured TPM** | **Simulated TPM** | **Prediction error** | **Measured TPM** | **Simulated TPM** | **Prediction error** |
| 1200 | B | 3624.40 | 3666.90 | 1.17 % | 3632.20 | 3675.80 | 1.20 % |
| | M | 1794.70 | 1846.80 | 2.90 % | 1808.50 | 1822.10 | 0.75 % |
| | PU | 1806.00 | 1803.70 | 0.13 % | 1798.30 | 1828.10 | 1.66 % |
| 1400 | B | 4242.50 | 4273.10 | 0.72 % | 4227.80 | 4290.90 | 1.49 % |
| | M | 2086.70 | 2112.20 | 1.22 % | 2098.80 | 2131.20 | 1.54 % |
| | PU | 2098.20 | 2150.30 | 2.48 % | 2144.70 | 2123.00 | 1.01 % |
| 1600 | B | 4831.10 | 4881.50 | 1.04 % | 4880.00 | 4869.70 | 0.21 % |
| | M | 2405.60 | 2438.60 | 1.37 % | 2400.40 | 2454.70 | 2.26 % |
| | PU | 2413.60 | 2426.20 | 0.52 % | 2428.40 | 2439.20 | 0.44 % |
| 1800 | B | 5430.70 | 5491.60 | 1.12 % | 5443.30 | 5503.00 | 1.10 % |
| | M | 2704.80 | 2746.10 | 1.53 % | 2718.00 | 2710.50 | 0.28 % |
| | PU | 2737.00 | 2699.70 | 1.36 % | 2720.50 | 2752.50 | 1.18 % |

**Table 10.7:** *Measured and simulated throughput*

| | AMD-based SUT | | | Intel-based SUT | | |
|---|---|---|---|---|---|---|
| **Users** | **MMCPU** | **SMCPU** | **CPUPE** | **MMCPU** | **SMCPU** | **CPUPE** |
| 1200 | 59.29 % | 60.02 % | 1.23 % | 55.85 % | 57.26 % | 2.53 % |
| 1400 | 66.00 % | 69.98 % | 6.03 % | 62.98 % | 66.78 % | 6.03 % |
| 1600 | 74.94 % | 79.95 % | 6.68 % | 70.72 % | 76.08 % | 7.58 % |
| 1800 | 84.06 % | 89.83 % | 6.85 % | 79.41 % | 85.69 % | 7.90 % |

**Table 10.8:** *Measured and simulated CPU utilization*

The adapted resource profiles for the AMD- and Intel-based SUTs are now used to predict the application performance on the corresponding hardware environments. Compared to the evaluations performed in the previous sections, we are now using higher scale production workloads of 1200 to 1800 concurrent users for the evaluation. The user count is again increased in increments of 200 users. The same workloads are executed on the AMD- and Intel-based SUTs using the benchmark driver on the build system (see VM2 in figure 10.15). Similar to the evaluations in the previous sections, only measurement and simulation results during a ten minute steady phase between a five minute ramp-up and a one minute ramp-down phase are considered. The measurements are also executed the same way as in the previous evaluations and are compared with each other afterwards. Response time results are shown in figures 10.20(a) and 10.20(b), and the throughput and CPU utilization results are presented in tables 10.7 and 10.8, respectively.

The median response times are predicted with a relative error of at most 26.54 % compared to the measurement results. The first and third quartiles are predicted with an error of largely below 30 %, except for the manage transaction on both SUTs where the first quartile is predicted with errors of up to 67.32 %. However, even the greatest relative error of the first quartiles is caused by an absolute difference of 5.86 ms and is thus not very high. Similar to the first evaluation, response time prediction errors do not have a considerable impact on throughput, as the prediction error for the throughput remains below 3 % (see table 10.7). The CPUPE is at most 7.90 % for a load of 1800 concurrent users on the Intel-based SUT but stays below that value in all other cases. According to these results, an adapted resource profile can predict the performance for hardware environments that are not the ones with which it was created with sufficient accuracy for capacity planning purposes (Menascé et al., 2004).

#### 10.6.5.2 Developing a Cost Model and Calculating the Cost

As outlined in section 10.5, one of the core steps after performance is predicted for a given workload and hardware environment, is to calculate the costs for this environment. To do

| | AMD-based SUT | | | Intel-based SUT | | |
|---|---|---|---|---|---|---|
| **Users** | **MMPC AMD** | **SMPC AMD** | **Prediction error** | **MMPC Intel** | **SMPC Intel** | **Prediction error** |
| 1200 | 300.00 W | 268.51 W | 10.50 % | 192.88 W | 185.73 W | 3.71 % |
| 1400 | 297.12 W | 280.20 W | 5.70 % | 199.15 W | 188.24 W | 5.48 % |
| 1600 | 303.90 W | 291.90 W | 3.95 % | 200.00 W | 190.70 W | 4.65 % |
| 1800 | 315.73 W | 303.50 W | 3.87 % | 201.86 W | 193.23 W | 4.28 % |

**Table 10.9:** *Measured and simulated power consumption for the AMD- and Intel-based SUTs*

that, we establish a cost model (see equation 10.7) that considers the hardware and energy costs of each hardware environment as well as space requirements within the data center.

In the following, we do not use absolute prices for the full servers as CPU cores have been disabled in this evaluation. Therefore, we just reflect that the Intel-based SUT costs 2.9 % less than the AMD-based SUT relative to the amount of enabled cores used for this evaluation[17]. This relative value is depicted by the initial cost ($IC$) variable in equation 10.7.

Equation 10.7 also includes the relative power consumption ($RPC$) of the AMD- and Intel-based SUTs for the SPECjEnterprise Orders domain application. To evaluate their energy consumption for the application and given workloads, we used an extension to the PCM meta-model and simulation environment introduced in our previous works (Brunnert/Wischer/Krcmar, 2014; Willnecker/Brunnert/Krcmar, 2014b). This extension allows the prediction of energy consumption of a hardware environment during simulations. To use this extension, we established so called power consumption models for the AMD- and Intel-based servers shown in equations 10.5 and 10.6. These models describe a server's power demand ($PD$) relative to the utilization of the CPU resource ($u_{CPU}$). These models are derived from measurements on the AMD- and Intel-based systems as shown in Brunnert/Wischer/Krcmar (2014).

$$PD_{AMD} = 198.042 + 1.174 * u_{CPU} \tag{10.5}$$

$$PD_{Intel} = 170.6339 + 0.2637 * u_{CPU} \tag{10.6}$$

The PCM extension using these power consumption models for the corresponding hardware environments was used during the performance predictions for the AMD- and Intel-based SUTs. The simulated mean power consumption (SMPC) results are shown in table 10.9. During the measurement runs on the Intel- and AMD-based SUTs, the measured mean power consumption (MMPC) was collected using hardware sensors in the corresponding servers accessible via the Intelligent Platform Management Interface (IPMI)[18]. The power consumption prediction error for each load level and system is shown in table 10.9. The SMPC values match their MMPC values with high accuracy thus confirming the results for the prediction of the CPU utilization shown in table 10.8 as they are derived from the same data sets.

For the cost model it is important to note that the Intel-based SUT consumes on average only 66 % of the power for the same SPECjEnterprise2010 Orders domain workload as the

[17]The prices for deriving this price ratio have been calculated based on the full server price divided by the total amount of cores and multiplied by the amount of enabled cores.

[18]`http://www.intel.com/design/servers/ipmi/`

| Server Count (SC) | Years (Y) | TCOfactor AMD-based SUT | TCOfactor Intel-based SUT |
|---|---|---|---|
| 1 | 1 | 1 | 0.9 |
| 1 | 5 | 1.4 | 1.2 |
| 2 | 1 | 2 | 1.8 |
| 2 | 5 | 2.8 | 2.3 |

**Table 10.10:** *Calculated total cost of ownership (TCO) factor for the AMD- and Intel-based SUTs*

AMD-based SUT (see table 10.9). This relative value is represented by the RPC variable in the cost model. To account for the energy bill over several years, the cost model also includes a factor years ($Y$) depicting the amount of years for which the energy costs should be calculated.

Software licenses are not considered in our calculations as the SPECjEnterprise2010 license fees are not bound to specific servers and the application servers and operating systems used in this evaluation are free of charge.

To consider space requirements for the different hardware environments in a data center, we include their rack units ($RU$) in the cost model. Both servers must be built into a rack and a rack can only contain a designated amount of servers depending on its size (a typical full rack provides space for 42 RU). Each additional rack requires more space in a data center as well as connections to the network and power systems. The AMD-based SUT requires two rack units[19] whereas the Intel-based SUT requires one rack unit[20]. The relative $RU$ value for the Intel-based SUT is 0.5 whereas it is 1 for the AMD-based SUT.

As the dimensions included in the cost model have different values and scales, weighted, relative values instead of absolute values are used to calculate a relative total cost of ownership factor ($TCOfactor$) for each hardware environment. The weighting is introduced to ensure that factors representing higher absolute values (such as the initial cost) are properly considered. For our evaluation, we calculate the $TCOfactor$ by weighting the initial purchasing costs with 80 % of the $TCOfactor$, while power and space are equally weighted with 10 %. This weighting is used in our case as German energy prices have a considerable impact on the operation cost of the data center used in the experiment and space in the data center is also limited. Other data center operators might use different weighting schemes. The overall relative cost factor for one server (calculated using the weighted, relative $IC$, $RPC$ and $RU$ values) is multiplied by the server count ($SC$) to account for the amount of servers that need to be purchased.

$$TCOfactor = SC * [(0.8 * IC) + 0.1 * (Y * RPC) + 0.1 * RU] \tag{10.7}$$

To calculate the resulting cost for the AMD-based SUT and the Intel-based SUT, the $TCOfactor$ is calculated for different numbers of servers and years (table 10.10). The results indicate a cost advantage using Intel-based servers for this enterprise application and workload, not only because their initial cost is slightly lower than the cost of the AMD-based servers. Due to its lower energy consumption, the $TCOfactor$ gets even better

[19] `http://www-03.ibm.com/systems/x/hardware/rack/x3755m3/`
[20] `http://www.redbooks.ibm.com/technotes/tips0804.pdf`

for each year the system is in operation. The lower space requirement is also an advantage of the Intel-based systems especially when the number of servers is increased. Hence, it appears to be economically more efficient to operate the Orders domain application on the Intel-based machines.

## 10.7 Related Work

The resource profile concept and its application areas outlined in this work relate to several existing areas of research. This section is therefore structured according to research areas of relevance to our work. First, we review approaches to support the continuous performance evaluation using performance change detection techniques (section 10.7.1). We then look at capacity planning for enterprise applications using performance models (section 10.7.2). The review of related work continues with approaches to estimate resource demands for software systems (section 10.7.3). In a forth step, methods for TCO calculations for enterprise applications based on energy consumption and performance predictions are presented (section 10.7.4). The review of related work concludes with approaches to improve the relationships of EAVs, EAUs and EAHs using resource demand data (section 10.7.5).

### *10.7.1 Continuous Performance Evaluation and Change Detection*

An approach to detect performance regressions early in the software development process has been proposed by Heger/Happe/Farahbod (2013). The authors propose to continuously conduct performance unit tests and analyze the results; once a performance regression is detected, the approach tries to detect the root cause of the regression. In order to do so, the authors combine response times gathered during performance unit tests with source code changes included in a version control system. Compared to the approach presented in this work, Heger/Happe/Farahbod (2013) focus on evaluations on the level of single component operations and response time data derived from small scale measurements (e.g., single user unit tests). Therefore, effects that might occur due to an increased resource consumption of a component might not be visible due to the low load. However, their approach could be combined with the performance change detection approach presented in this work by simply measuring the resource demand during the performance unit tests and using this data for model creation and predictions for different workloads and hardware environments.

A benchmark-based approach for detecting performance regressions has been proposed in Bulej/Kalibera/Tůma (2005) and Kalibera/Bulej/Tůma (2005a). These authors also outline the necessity to detect performance changes in different versions of a software system. Their work is evaluated using two enterprise application benchmarks (Bulej/Kalibera/Tůma, 2005). Similar to the change detection evaluation in this work, these authors show that they can detect a performance bug introduced between two benchmark executions. In a later work, the authors try to address the non-determinism of results in individual benchmark runs due to different initial states (Kalibera/Bulej/Tůma, 2005b; Kalibera/Tůma, 2006). For this purpose, the authors propose the calculation of an impact factor to

describe the non-determinism of the initial state of a specific deployment of a software system which is taken into account to determine the required number of measurements for detecting performance change with high confidence. The approach presented in this work extends their work by using model-based performance predictions instead of benchmark executions; it would, however, be an interesting enhancement to also consider such an impact factor for the change detection based on resource profiles.

Mi et al. (2008) and Cherkasova et al. (2008) propose creation of so-called application signatures and compare the performance characteristics of each application version using their corresponding signatures. Application signatures are a representation of transaction processing times relative to the resource utilization for a specific workload. The authors extend their approach in Cherkasova et al. (2009) by using performance modeling techniques to not only detect performance changes based on response times but also to evaluate performance anomalies in the CPU demand of transactions. Their work focuses on systems that are already in production. This work extends the idea of evaluating the change in performance of each application version to the development process. Furthermore, we propose the use of resource profiles instead of application signatures: resource profiles allow for more flexible evaluations as they can be used to derive these metrics for different workloads and hardware environments. It is also possible to derive these metrics with smaller systems as it would be possible using application signatures.

An approach to detect performance regressions of different versions of a software system during development has been proposed by Nguyen et al. (2012). The authors propose the use of control charts to identify whether or not results of a performance test indicate a regression. Their approach focuses on the automatic detection of regressions based on performance test results and it could be used to evaluate prediction results based on resource profiles. However, their approach requires a real performance test and does not support the detection of possible problem causes as is feasible by comparing resource profiles. Their approach also assumes a linear relationship between the resulting performance metrics and the load on a system, which might not always be true.

An approach to diagnose detected performance changes is proposed by Sambasivan et al. (2011) who suggest a comparison of request flow traces in order to identify the cause of a change. Their approach relies on runtime traces and defines algorithms to rank changes in the control flow or in the response time of single components of an application. This approach could be used to enhance the search for a root cause of a performance change in resource profiles. This is possible because resource profiles contain the structural data required for their approach and prediction results provide response times for single components during a transaction processing. Their approach could be enhanced using resource profiles to take resource demands into account.

An interesting approach to detect and visualize the cause of a performance change using PCM models is proposed by Brüseke et al. (2014). The authors employ the Kolmogorov-Smirnov test to detect changes in the response times of two simulation results and propose to create performance reports including flame-graphs to visualize operations within a control flow that contribute most to the response time of an overall transaction. Once a performance change has been detected using resource profiles, the flame-graphs could be used to help developers analyzing the reasons for a change.

An approach that not only tries to detect performance changes between one or more application versions has been proposed by Wert/Happe/Happe (2013). The authors propose to automatically detect performance antipatterns using performance measurements. They show how the throughput of a benchmark application can be increased by over 50 % by removing antipatterns detect using their approach. It might prove useful to work on a combination of the performance change detection approach proposed in this work and the antipattern detection approach presented by Wert/Happe/Happe (2013). This integration would allow developers to be automatically informed about improvement opportunities although no change is detected.

### *10.7.2 Capacity Planning Using Performance Models*

Liu/Kumaran/Luo (2001) show how layered queuing network (LQN) models can be used to support the capacity sizing for EJB applications. In a later work, Liu/Shen/Kumaran (2004) use LQN models to realize a capacity sizing tool for a business process integration middleware by taking different CPU configurations into account. The authors introduce a way to deal with different hardware environments in the context of LQN models. They implemented a model transformation tool which dynamically constructs LQN models from extensible markup language documents representing the application model on the one hand and the hardware configuration on the other. However, using their current implementation only the processing speed of the CPU of a server can be changed and it is not possible to change the hardware environment more radically, for example from a one server deployment to multiple servers.

King (2004) describes an approach to use resource profiles to calculate resource utilization values and to derive hardware recommendations from these results. For that purpose, the author multiplies the resource demand values with the amount of users per transaction and divides the result by the available processing rate. However, the author did not calculate other metrics (i.e., throughput, response time) based on this information and only takes queuing into account by adding a static contingency number and limiting the utilization levels of a system. This approach does not allow for any modifications in terms of the deployment topology.

A model-driven capacity planning tool suite for component- and web service-based applications is proposed by Zhu et al. (2007). The tool suite can be used in early software design phases to support the performance evaluation and comprises tools to transform existing design models into performance models and benchmark drivers to derive resource demands for the performance models. These performance models and benchmarks can then be used to support capacity planning tasks. Their tooling is intended to be used early in the development process and not as a final capacity planning tooling, as the implementation might have different characteristics than the generated benchmark code.

Tiwari/Nair (2010) use LQNs to predict the performance of two deployments of the same Java EE sample application. Similar to the approach in this work, they show how the SPEC CPU benchmark can be used to adapt a LQN model to a different hardware environment. However, LQN models do not allow a change in workload or the hardware environment

without reconstructing the whole model. Their approach is thus less flexible than the one proposed in this work.

In the last couple of years, the focus of model-based capacity planning approaches shifted to online performance predictions. These approaches try to predict and adapt the required capacity during the runtime of an EA as opposed to beforehand. A recently released modeling language to support such runtime adaptations is the DML (Kounev/Brosig/Huber, 2014). One of the key challenges for such approaches is dealing with the virtualization layer as the provisioning and removal of resources needs to be done on this layer (Huber et al., 2011; Brosig et al., 2013). Compared to traditional capacity planning approaches, these online approaches often need to use very fast solution techniques which often leads to reduced accuracy. However, runtime models such as DML can also be solved using simulation approaches for more accurate estimations. It would be interesting to combine the resource profile contents with the advanced infrastructure representation capabilities of models such as DML to better support capacity planning activities in cloud scenarios.

### *10.7.3 Resource Demand Estimation*

Resource profiles contain resource demands that need to be derived from measurements. The existing work to derive resource demands can be categorized into two basic approaches: using direct measurements and using statistical methods to estimate resource demands from other metrics (Spinner et al., 2015). The approach for Java EE systems used in the evaluation for this work (Brunnert/Vögele/Krcmar, 2013; Brunnert/Neubig/Krcmar, 2014) is an approach that directly measures resource demands. Another tool-supported approach that allows an estimation of resource demands based on indirect measurements has been introduced by Spinner et al. (2014). Their tooling implements several approaches to derive resource demands based on throughput, response time and resource utilization metrics (Spinner et al., 2015). The advantage of these approaches is that they are technology independent and generate less overhead as no additional measurements are necessary. However, it is often challenging to use such approaches to derive the demand for more than one resource type. Furthermore, it is often necessary to measure all transactions that are executed on a system during the time a resource utilization is measured in order to avoid an overestimation of resource demands for specific transactions. Nonetheless, there are several use cases in which resource demand measurement and estimation techniques can be used in combination to collect comprehensive resource profile data for complex software systems that cannot be fully instrumented.

### *10.7.4 Combination of Performance and Energy Prediction*

One of the few examples of combining energy consumption and performance prediction approaches with a business perspective is the work of Li/Casale/Ellahi (2010). The authors propose a sizing methodology for enterprise resource planning (ERP) systems based on closed queuing networks. Their methodology allows optimization of sizing decisions using multiple dimensions. With this methodology, TCO decisions can be performed including hardware purchasing and energy consumption costs for new ERP systems. Unfortunately,

their approach is limited to ERP systems with a predefined set of deployment options and is thus not transferable to other types of applications. However, their multi objective optimization (MOO) approach might be an interesting enhancement for the resource profile concept introduced in this work as resource profiles could be used as the input for a MOO solver.

### *10.7.5 Relationships between EAV, EAU and EAH*

The term resource profile has been used by Brandl/Bichler/Ströbel (2007) in their work on cost accounting for shared IT infrastructures. The authors introduce an approach to associate resource demands to specific IT services (e.g., email) and to store these resource demands for different user types in service-specific vectors (called resource profiles). Using these resource demand vectors, they propose an approach to exactly bill the service consumers based on the number of users and types of services being used. Compared to the approach presented in this work, their resource profile concept is mainly intended to be used to allocate costs for existing applications and services more precisely. The approach presented in this work is intended for new applications and services that should be integrated into a data center. However, the data in our resource profile can also be used for the cost accounting approach presented by Brandl/Bichler/Ströbel (2007).

## 10.8 Conclusion and Future Work

This work introduced resource profiles represented as architecture-level performance models as a way of describing the resource demand of enterprise application versions. They describe an EA in its current state and allow for predicting its performance for different workloads and hardware environments. Two application areas for resource profiles have been introduced in this work, namely: the continuous performance evaluation during software development and capacity planning once an EA is released.

Continuous performance evaluation is realized by introducing a performance change detection step within a deployment pipeline. This additional step allows to automatically detect if feature additions or bug fixes introduced in new EA versions lead to performance changes of the overall EA. In case a change is detected, the resource profile of the corresponding EA version can be used together with the information about the developer check-ins that triggered a build to identify the source of a performance change.

Once an EA version is released, the resource profile of this EA version can be distributed along with the EA binaries by an EAV. The ability to specify the resource profile independently from the workload and hardware environment, helps to better estimate costs of EAUs and EAHs for using or hosting the EA by specifying their workload and hardware environment models then using the combined model as input for a simulation engine to derive performance and energy consumption predictions. A capacity planning process has been introduced that uses these prediction results as input for a cost model.

The capabilities introduced in this work have been evaluated in three integrated experiments for a representative enterprise application. In a first experiment, resource profiles of different versions of the same application were successfully applied to detect a performance change and its root cause. The second experiment validated the claim that the workload can be changed independently from the resource profile and hardware environment specification using a resource profile created during the change detection step. Afterwards, the same resource profile was used for a capacity planning scenario in order to decide which one of two hardware environments provided the required capacity in a more economical way. The predictions of performance and energy consumption for these evaluations matched measurements on real systems with high accuracy.

In future work, the applicability of the concepts needs to be evaluated for other technology stacks (e.g., .NET or ABAP) or systems with asynchronous communication patterns (e.g., message-oriented middleware). A key requirement for such an evaluation is the ability to collect the required data for creating resource profiles. Instead of relying on a custom solution to collect the required data, as was used for the evaluation in this work, industry standard measurement solutions should be used. This type of an extension is currently being developed as part of our work presented in Willnecker et al. (2015a).

Another direction of future work is to automate the complete performance change detection process and to integrate it as a plugin in CD systems. Furthermore, the steps of comparing different resource profile versions and identifying the problem causes should be handled automatically. It might also be interesting to use information about check-ins included in a build to perform static analysis to improve the search process for the reasons or causes of a change.

Additional extensions are required if an EA depends on existing systems within a data center. As resource profiles describe a specific enterprise application, approaches to represent external systems as black-box components need to be introduced. It would be helpful if EAUs and EAHs could replace default representations of external systems within a resource profile with resource profiles of corresponding systems used in their data centers.

# Part C

# Chapter 11

# Summary of Results

This chapter summarizes the results by outlining the individual results of the embedded publications in a first step and summarizing the overall results in a second step.

## 11.1 Results of Embedded Publications

An overview of the key results for the individual publications is given in table 11.1. These results are outlined in more detail in the following paragraphs.

### P1: Performance Management Work

The first publication uses informed arguments to outline open challenges in the fields of SPE and APM. It highlights the necessity to integrate SPE and APM activities due to the changes that occurred in the way EAs are delivered and maintained from the time SPE and APM were established up until today. First of all, the system life cycle becomes more and more integrated due to concepts such as DevOps, which lead to a high frequency of releases and a need for better communication between development (Dev) and operation (Ops) teams. The publication also argues that the changes in the IT architecture (from monolithic to system-of-systems) lead to a change in the IT governance structures. This is due to the fact that multiple organizations are responsible for different parts of an overall software system. As most of these changes can only be solved by integrating technical and organizational perspectives, it proposes the term Performance Management Work (PMW) as a process-oriented approach that integrates SPE and APM activities to address these challenges from both perspectives.

### P2: Integrating the Palladio-Bench into the Software Development Process of a SOA Project

The author has been involved in several industrial projects related to the topic of this dissertation. The second publication outlines one of them. It explains the project scope and the challenges while incorporating model-based performance evaluations into the development process of a large-scale development project. The scope of the project was to establish a continuous performance management that follows core concepts of PMW. For

this purpose, APM data collected during operation needs to be fed back to development in order to support SPE activities. The project was in an early phase at the time of writing this publication but several shortcomings of the PCM meta-model, its modeling environment (Palladio-Bench) and simulation engines could already be identified and are outlined in this publication. One of the key solutions for the integration of model-based performance evaluations into the project is the use of automatic performance generators. In the case of this project, the author did not rely on runtime instrumentations to create the models as in the other works included in this dissertation but on design models. Performance models derived from design documents are complemented with so called performance curves describing the response time behavior of external services in order to continuously evaluate the performance of an EA in a SOA environment.

**P3: Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications**

During the literature reviews performed while preparing the publications P1 and P2, one unresolved challenge of model-based performance evaluations became evident: the construction of a performance model. In order to create a performance model, several conceptual questions need to be solved regarding the granularity of the model content and the feasibility of collecting the required data for parameterizing the model. Publication P3 proposes a solution that answers the conceptual questions, allows to collect the required data and automatically generates performance models for EAs built upon the Java EE standard. The proposed solution is built so that it can be applied on all Java EE servers that are compliant with the specification. A prototype for the proposed solution is used to evaluate the accuracy of generated models in an experiment using a SPECjEnterprise2010 deployment.

**P4: Evaluating the Prediction Accuracy of Generated Performance Models in Up- and Downscaling Scenarios**

One of the key use cases of performance models is to evaluate performance in scenarios that cannot be tested on a real system. Scaling a system up and down in terms of the available hardware resources (e.g., CPU cores) are examples for such scenarios. They commonly occur in capacity planning processes. The results of an experiment outlined in publication P4 show that performance models generated with an improved solution based on the work in publication P3 can predict the performance of a representative EA in such scenarios with high accuracy.

**P5: Using Architecture-Level Performance Models as Resource Profiles for Enterprise Applications**

As the generated performance models showed promising results in the evaluation in publication P4, their use in real world capacity planning scenarios is outlined in publication P5. In a first step, P5 explains that there is a lack of a communication medium between EAVs, EAUs and EAHs. P5 proposes such a communication medium called resource profile. Resource profiles are represented using the PCM meta-model but have specific properties that differentiate them from regular performance models. The main difference is that resource profiles are meant to describe the current state of an EA and not to evaluate changes. Their structure is designed so that the communication between EAVs,

EAUs and EAHs can be properly supported. EAVs can create resource profiles for all applications sold, which allow EAUs to specify their workload and EAHs to specify the hardware environment on which an EA should be deployed. In order to make them easily usable, system specifics are hidden for EAUs and EAHs. EAUs can specify their workload on the level of business transactions, whereas EAHs can use predefined deployment units to map the EA on their hardware environment instead of individual components. This structure avoids wrong configurations not supported by an EAV.

As capacity planning not only considers initial hardware and software costs, it is important to incorporate other cost drivers. One of the major cost drivers in data centers nowadays is the energy consumption. This publication therefore proposes an extension to PCM that not only allows for predictions of performance but also of the energy consumption of an EA deployment.

The resource profile concept and the ability to predict energy consumption are evaluated in an experiment following a hardware migration scenario. A resource profile for a representative EA is created and evaluated on one hardware environment and then used to predict performance and energy consumption for a hardware environment that is not the one it was created with. In order to create resource profiles, an improved version of the model generation approach introduced in P3 and P4 is used that creates PCM models with the specific properties of resource profiles. The evaluation results validate the use of resource profiles for planning capacity for different hardware environments and furthermore the PCM extension to allow for energy consumption predictions.

**P6: Detecting Performance Change in Enterprise Application Versions Using Resource Profiles**

As outlined in publication P1, recent changes in the system life cycle lead to a continuous release of new EA versions. In order to be able to release EAs that are properly evaluated from a performance standpoint, this publication proposes a performance change detection process using resource profiles. This performance change detection process allows to identify if bug fixes or feature additions lead to an improvement or a regression of EA performance. Performance change is defined as a change in transaction response time as it is the key metric from the perspective of a single user. The performance change detection process is integrated into a deployment pipeline of a continuous delivery process. This integration ensures that the performance change detection is executed for every EA version that is being built. It furthermore supports the search for a reason of a performance change as the source code changes that are included in a build are known.

The overall change detection process is evaluated in an experiment that shows that a performance regression introduced in a new EA version is properly detected using resource profiles in a deployment pipeline. The deployment pipeline in this experiment is implemented using the continuous integration server Jenkins. Two versions of a representative EA are being built, from which the second one contains a performance regression. Resource profiles for both EA versions are automatically created using the generator introduced in publications P3, P4 and P5. The resource profiles for both EA versions are used to predict performance. Comparing the prediction results clearly identified the regression. A comparison of both resource profile versions furthermore revealed the reason for the change in performance.

| No. | Title | Key Result(s) |
|---|---|---|
| P1 | Performance Management Work | - Outlines the necessity to integrate SPE and APM activities and explains conceptual challenges for this integration |
| P2 | Integrating the Palladio-Bench into the Software Development Process of a SOA Project | - Outlines the challenges of integrating model-based performance evaluations into an industry project |
| P3 | Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications | - Introduces a solution to automatically create performance models |
| P4 | Evaluating the Prediction Accuracy of Generated Performance Models in Up- and Downscaling Scenarios | - Improves the solution to create performance models and evaluates it in typical capacity planning scenarios |
| P5 | Using Architecture-Level Performance Models as Resource Profiles for Enterprise Applications | - Introduces resource profiles that describe EAs in their current state and use cases for resource profiles in capacity planning scenarios<br>- Extends the PCM meta-model to allow for energy consumption predictions |
| P6 | Detecting Performance Change in Enterprise Application Versions Using Resource Profiles | - Introduces an approach to detect performance changes for EA versions within a deployment pipeline |
| P7 | Continuous Performance Evaluation and Capacity Planning Using Profiles for Enterprise Applications | - Outlines an integrated approach to create resource profiles during development, detect performance changes for each application version and distribute the resource profiles to support capacity planning activities |

**Table 11.1:** *Key results of embedded publications*

### P7: Continuous Performance Evaluation and Capacity Planning Using Resource Profiles for Enterprise Applications

Publication P7 enhances and integrates the results of publications P3 to P6 to present a continuous performance evaluation and capacity planning approach for EAs in rapid release cycles.

In a first step, publication P7 describes the idea of resource profiles from a theoretical perspective. Afterwards, it explains the representation of resource profiles as architecture-level performance models. P7 describes the transformation from a conceptual point of view and includes additional resource types such as network and HDD. The previous publications describe this transformation process only for the Java EE technology including CPU and memory resources.

The performance change detection and capacity planning approaches introduced in P5 and P6 are integrated in order to allow for continuous releases that are properly evaluated from a performance perspective. As not all workloads and hardware environments can be evaluated in a deployment pipeline as shown in publication P6, the distribution of resource profiles created in the deployment pipeline for capacity planning purposes as outlined in publication P5 is explained. For that purpose, a capacity planning process using resource profiles is introduced.

Three integrated experiments evaluate the performance evaluation and capacity planning capabilities. The first experiment is similar to the one in P6 and shows that a performance regression can be detected by using resource profiles in a deployment pipeline. As all three experiments take network demands into account, the performance regression is not only based on increased CPU demands as in P6 but also on increased network demands. The resource profile of an EA version without a regression is used as input for the next two experiments. The second experiment validates the claim that the workload for a resource profile can be modified independently and shows that predictions for two workload changes match measurements with high accuracy. The third experiment evaluates a hardware purchasing scenario. The resource profile for the EA version without a regression is adapted to two different hardware environments and used to predict their performance and energy consumption. The prediction results of the adapted resource profile for both hardware environments are evaluated against measurements of their real systems with high accuracy. The prediction results are then used as input for a cost model in order to evaluate the capacity planning process. The cost model takes initial hardware costs, energy costs and space requirements in a data center into account. A total cost of ownership value is calculated using the cost model that shows which of the two hardware environments provides the required capacity in a more economic way.

## 11.2 Overall Results

As outlined in the introduction, the overall goal of this dissertation is to support performance evaluations for EAs by integrating SPE and APM activities to better cope with recent changes in software development and operation processes.

In order to achieve this goal, so called resource profiles are introduced that describe the resource demand of EAs in their current state. Resource profiles are transformed into architecture-level performance models in order to allow for performance predictions for different workloads and hardware environments. Compared to traditional performance models, their purpose is not to evaluate changes but to describe the current state of an EA. Resource profiles furthermore have a specific structure to make them easily usable throughout the life cycle of an EA.

In a first step, this dissertation introduced conceptual and practical challenges for model-based performance evaluations. A key challenge of applying model-based performance evaluations is addressed by introducing a solution to automatically generate performance models for Java EE applications. This solution is continuously improved and evaluated throughout this dissertation to allow for the construction of resource profiles. Afterwards, two application areas for resource profiles are outlined: continuous performance evaluations in rapid release cycles and capacity planning once an EA version is released.

Continuous performance evaluation is realized by introducing a performance change detection step within a deployment pipeline. This additional step allows to automatically detect if feature additions or bug fixes introduced in new EA versions lead to performance changes of the overall EA. In case a change is detected, the resource profile of the corresponding EA version can be used together with the information about the developer check-ins that triggered a build to identify the source of a performance change.

Once an EA version is being released, the resource profile of this EA version can be distributed along with the EA binaries by an EAV. EAUs and EAHs can specify their workload and hardware environment models for the provided resource profile independently from each other and use the combined model as input for a simulation engine to derive performance and energy consumption predictions. A capacity planning process is introduced that uses these prediction results as input for a cost model.

Resource profiles, as a technical solution, also address some of the organizational challenges for a SPE and APM integration outlined in publication P1. They are created during the development process, increase the performance awareness of development teams, and serve as a communication medium between different parties involved in the operation of an EA once it is released. Even though a company can be in several roles at the same time (e.g., an EAU with an internal operation team that serves as an EAH), the roles EAV, EAH and EAU are usually performed by different organizational units. Therefore, providing such a communication medium makes the capacity planning processes more transparent as the necessary data is available to all parties involved.

The continuous performance evaluation and capacity planning capabilities introduced in this work were evaluated in multiple experiments using a representative EA. The experiment results validate the utility of these capabilities. Furthermore, the experiment results show that performance and energy consumption predictions based on resource profiles for different workloads and hardware environments match measurement results on real systems with high accuracy.

# Chapter 12

# Contribution and Limitations

## 12.1 Contribution to Research and Practice

Research in the field of EA performance is currently either focused on improving performance evaluations during development (SPE) or on improving performance during operation (APM). Both fields have been research areas for several decades. Yet, there is little work that tries to build a bridge between both worlds. This gap has not been addressed so far as the closer integration of development and operation processes is a phenomenon that can only be observed in recent years. The main contribution to research is therefore that this work closes the gap in between these two areas as it outlines how resource profiles, as artifacts created and used during development, can be used in the transition to and during operation in order to support performance evaluations throughout the life cycle of an EA. As the required data to generate resource profiles can also be collected during operation, it also supports the communication in the other direction as developers can use the data of EAs that are already in operation to support performance evaluations in early phases of the development process as shown in publication P2. In addition to the individual contributions outlined in section 11.2, some of the model-based performance evaluations are in a scale that is seldom found in the community. Systems were evaluated with up to 4300 concurrent users by comparing measurement and simulation results.

As contribution to practice, this dissertation improves the applicability of model-based performance evaluations. A lot of the performance evaluation approaches suggested under the umbrella of SPE can only be applied once a model of a system exists. The ability to automatically generate such models makes it possible to better apply existing model-based performance evaluation techniques in practice. Furthermore, two new application areas of such generated models are introduced. The capabilities introduced in this work allow for ensuring that all EA versions that are released are properly evaluated from a performance perspective. They furthermore support EAVs and EAHs during the capacity planning process for new or existing deployments of an EA.

The improved applicability of performance models furthermore helps to reduce the costs for performance evaluations. This is due to the fact that measurement-based performance evaluations applied in practice require a test environment that is comparable to a production system. This requirement introduces a lot of cost for companies as maintaining such

systems involves hardware costs, software license costs, operating costs (e.g., space and energy cost) and a lot of manual labor. As multiple deployments of an EA can exist it is often not feasible to maintain test instances for each deployment. Furthermore, not all deployments are known at the time of a release (e.g., for off-the-shelf-products). Model-based performance evaluations avoid the need for maintaining such expensive test environments for all deployments and, thus, reduce the cost for performance evaluations considerably. They furthermore support situations in which deployments are not known at the time of EA release.

## 12.2 Limitations of the Results

The continuous performance evaluation and capacity planning capabilities introduced in this work can be applied for different types of EAs. However, this work evaluates these capabilities only for EAs built upon the Java EE standard. Future work could improve the applicability of the resource profile concept for other types of EAs.

Even for Java EE applications, several important aspects with an impact on performance are not considered in the embedded publications. One of the main missing factors is the memory management within Java Virtual Machines (JVM). The memory management is realized by garbage collectors (GC) that can stop the execution of application threads in a JVM at certain points in time depending on the GC strategy. The impact of a specific type of GC strategy is not considered.

It is furthermore important to note that most of the embedded publications use the SPECjEnterprise2010 benchmark application to evaluate solutions introduced in this dissertation. Even though using a benchmark application has the advantage of reproducibility due to the predefined EA, dataset and workload that need to be used, the generalizability of the results is limited. The benchmark application is moreover built upon the features provided by the Java EE 5.0 standard and Java EE has been developed further. It would thus be beneficial to evaluate the solutions for an application that also makes use of new features added to the Java EE standard.

Additional limitations arise from the resource types covered by resource profiles used in the experiments. Even though memory demands can be measured and represented in a resource profile on an appropriate granularity level using the approaches presented in this work, the accuracy of prediction results for this resource type is questionable. Furthermore, the measurement of memory demands increases the instrumentation overhead of the data collection approach. This overhead made it impossible to collect memory demand data for the evaluations in the experiments as shown in publication P4. Additionally, the data collection approach is currently not able to measure HDD demand on an appropriate granularity level on multiple platforms. Due to the aforementioned reasons, the evaluations in this work focus on the demand of software on CPU and network.

Another key limitation is the representation of interacting EAs. As one resource profile only represents one EA, concepts are missing to integrate resource profiles of multiple applications in order to evaluate their combined performance.

# Chapter 13

# Future Research Directions

Several areas that are covered by this dissertation can be complemented with additional research efforts. In the following, four possible future research directions and current efforts in these directions are outlined.

### Further Integration of SPE and APM

First of all, a lot of research opportunity lies in the further integration of SPE and APM activities to better support the performance evaluation throughout the life cycle of EAs. As this is a huge area with a lot of possibilities, the author has co-founded the DevOps Performance Working Group within the research group of the Standard Performance Evaluation Corporation (SPEC)[1]. This working group provides an international platform for researchers and practitioners to collaborate on the challenges of integrating performance evaluations within development (SPE) and operation (APM).

One of the challenges for the integration of model-based SPE and measurement-based APM activities addressed in this dissertation is the automatic generation of performance models. However, the existing prototype for the model generation relies on a measurement infrastructure that has been developed by the author. This infrastructure is not yet robust enough for large-scale industrial use, it would thus be beneficial to rely on existing measurement infrastructures for the model generation instead. The use of existing measurement infrastructures would also simplify the model generation for different technology stacks. A currently ongoing effort to address this challenge is the integration of the model generator developed as part of this dissertation with the dynaTrace[2] monitoring solution as outlined in Willnecker et al. (2015a).

### Improving Resource Profile Representation and Simulation Capabilities

The meta-model for the representation of resource profiles and its simulation engines need to be extended to better support the representation of memory consumption. It would be interesting to analyze if common garbage collection behaviors for managed runtime environments such as JVMs could be predefined in the meta-model similar to scheduling mechanisms for other resources as suggested in the work of Libič/Tůma/Bulej (2009).

[1] `http://research.spec.org/working-groups/devops-performance-working-group.html`
[2] `http://www.dynatrace.com/`

Publication P1 outlined three major dimensions that lead to changing requirements for SPE and APM activities: the system life cycle, the IT governance and the system architecture. The results of this dissertation address the system life cycle and IT governance perspectives, but do not properly address changes from the system architecture perspective as only one EA (system) is represented in a resource profile as of today. It would thus be beneficial to extend the resource profile representation in a way that if an EA interacts heavily with another application, the resource profiles of both EAs can be combined and exchanged easily.

**Diagnosing Reasons for Performance Changes**

The performance change detection process outlined in this dissertation can also be complemented with additional research efforts. A key challenge that needs to be addressed is the automatic detection of the source of a performance change. This would allow to automatically indicate in which component (or even operation) of an EA a change occurred. An additional step in the direction of diagnosing the reason of a performance change could be to apply static code analysis on the check-ins that led to a build. The combined knowledge of changes in the resource demand of a component operation and the code changes could be used for an automatic detection of the root cause for a performance change.

**Energy Efficiency of Enterprise Applications**

The capacity requirements of EAs are continuously rising as they need to handle an ever increasing amount of features, users and data. Therefore, data centers tend to continuously grow. This growth contributes not only to the operation cost of an EAH but also to the energy consumption and carbon dioxide production of mankind. As the way an EA is implemented directly influences the amount of resources, and, thus, power it consumes, future research should integrate power consumption as one element in an overall performance evaluation. This would allow to quantify the energy efficiency of an EA.

This work has shown that by extending the PCM meta-model, the energy consumption of an EA for a given workload can be predicted with high accuracy. These capabilities could be extended to be applicable in several areas. First of all, the energy consumption should be included in the performance change detection approach to make developers aware of changes in the energy consumption of their software. Furthermore, the resource profile descriptions could be used to assign efficiency labels to EAs according to their resource and energy consumption, similar to what is being done for fridges or cars nowadays. In the short term, such an energy evaluation and efficiency label would be especially useful for applications that run on devices with limited battery capacity.

## References


**Balsamo, S.**; **Di Marco, A.**; **Inverardi, P.**; **Simeoni, M. (2004):** Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, vol. 30 no. 5, 295 – 310, ISSN 0098–5589.

**Bause, F. (1993):** 'QN + PN= QPN' - Combining Queueing Networks and Petri Nets. Department of Computer Science, University of Dortmund – technical report.

**Becker, S. (2008):** Coupled Model Transformations for QoS Enabled Component-Based Software Design. vol. 1, Karlsruhe Series on Software Quality. Universitätsverlag Karlsruhe, Karlsruhe, Germany, ISBN 9783866442719.

**Becker, S.**; **Hasselbring, W.**; **Hoorn, A. van**; **Reussner, R. eds. (2013):** Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days., CEUR Workshop Proceedings vol. 1083, CEUR-WS.org, Karlsruhe, Germany.

**Becker, S.**; **Koziolek, H.**; **Reussner, R. (2009):** The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, vol. 82 no. 1, 3 – 22, ISSN 0164–1212.

**Bolch, G.**; **Greiner, S.**; **Meer, H. de**; **Trivedi, K. S. (1998):** Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. Wiley-Interscience, New York, NY, USA, ISBN 0–471–19366–6.

**Brandl, R.**; **Bichler, M.**; **Ströbel, M. (2007):** Cost Accounting for Shared IT Infrastructures - Estimating Resource Utilization in Distributed IT Architectures. *Wirtschaftsinformatik*, vol. 49 no. 2, 83–94, ISSN 0937–6429.

**Brosig, F.**; **Gorsler, F.**; **Huber, N.**; **Kounev, S. (2013):** Evaluating Approaches for Performance Prediction in Virtualized Environments. In Proceedings of the 21st IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems., MASCOTS '13, ISSN 1526–7539, 404–408.

**Brosig, F.**; **Huber, N.**; **Kounev, S. (2011):** Automated Extraction of Architecture-Level Performance Models of Distributed Component-Based Systems. In Proceedings of the 26th IEEE/ACM International Conference On Automated Software Engineering. Lawrence, Kansas, USA, ASE '11, ISSN 1938–4300, 183–192.

**Brosig, F.**; **Huber, N.**; **Kounev, S. (2014):** Architecture-level software performance abstractions for online performance prediction. *Science of Computer Programming*, vol. 90, Part B no. 0, 71 – 92, Special Issue on Component-Based Software Engineering and Software Architecture, ISSN 0167–6423.

**Brunnert, A.**; **Hoorn, A. van (2015):** SPEC RG DevOps Performance Working Group. In Poster presented at the 6th International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '15.

**Brunnert, A.**; **Danciu, A.**; **Krcmar, H. (2015):** Towards a Performance Model Management Repository for Component-based Enterprise Applications. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '15, ISBN 978–1–4503–3248–4, 321–324.

**Brunnert, A.**; **Danciu, A.**; **Vögele, C.**; **Tertilt, D.**; **Krcmar, H. (2013):** Integrating the Palladio-Bench into the Software Development Process of a SOA Project. In **Becker et al.** (2013), 30–38.

**Brunnert, A.**; **Krcmar, H. (2014):** Detecting Performance Change in Enterprise Application Versions Using Resource Profiles. In Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, VALUETOOLS '14, ISBN 978–1–63190–057–0, 165–172.

**Brunnert, A.**; **Krcmar, H. (2015):** Continuous performance evaluation and capacity planning using resource profiles for enterprise applications. *Journal of Systems and Software*,, `http://dx.doi.org/10.1016/j.jss.2015.08.030`.

**Brunnert, A.**; **Neubig, S.**; **Krcmar, H. (2014):** Evaluating the Prediction Accuracy of Generated Performance Models in Up- and Downscaling Scenarios. In Proceedings of the Symposium on Software Performance. Stuttgart, Germany, SOSP '14, 113–130.

**Brunnert, A.**; **Tertilt, D.**; **Vögele, C.**; **Krcmar, H. (2012):** Applying the Palladio Tool in a SOA Project. Paderborn, Germany, Palladio Days.

**Brunnert, A.**; **Vögele, C.**; **Danciu, A.**; **Pfaff, M.**; **Mayer, M.**; **Krcmar, H. (2014):** Performance Management Work. *Business & Information Systems Engineering*, vol. 6 no. 3, 177–179, ISSN 1867–0202.

**Brunnert, A.**; **Vögele, C.**; **Krcmar, H. (2013):** Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications. In **Balsamo, M. S.**; **Knottenbelt, W. J.**; **Marin, A. eds.:** Computer Performance Engineering. vol. 8168, Springer, Berlin/Heidelberg, Germany, ISBN 978–3–642–40724–6, 74–88.

**Brunnert, A.**; **Wischer, K.**; **Krcmar, H. (2014):** Using Architecture-level Performance Models As Resource Profiles for Enterprise Applications. In Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures. ACM, New York, NY, USA, QoSA '14, ISBN 978–1–4503–2576–9, 53–62.

**Brüseke, F.**; **Engels, G.**; **Becker, S. (2013):** Decision Support via Automated Metric Comparison for the Palladio-based Performance Blame Analysis. In Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '13, ISBN 978–1–4503–1636–1, 77–88.

**Brüseke, F.**; **Wachsmuth, H.**; **Engels, G.**; **Becker, S. (2014):** PBlaman: performance blame analysis based on Palladio contracts. *Concurrency and Computation: Practice and Experience*, vol. 26 no. 12, 1975–2004, ISSN 1532–0634.

**Bulej, L.**; **Kalibera, T.**; **Tůma, P. (2005):** Repeated Results Analysis for Middleware Regression Benchmarking. *Performance Evaluation*, vol. 60 no. 1-4, 345–358, ISSN 0166–5316.

**Capra, E.**; **Formenti, G.**; **Francalanci, C.**; **Gallazzi, S. (2010):** The Impact of MIS Software on IT Energy Consumption. In **Alexander, P. M.**; **Turpin, M.**; **Deventer, J. P. van eds.:** 18th European Conference on Information Systems, ECIS 2010, Pretoria, South Africa, June 7-9, 2010. , ISBN 978–0–620–47172–5.

**Chen, S.**; **Liu, Y.**; **Gorton, I.**; **Liu, A. (2005):** Performance Prediction of Component-based Applications. *Journal of Systems and Software*, vol. 74 no. 1, 35 – 43, ISSN 0164–1212.

**Cherkasova, L.**; **Ozonat, K.**; **Mi, N.**; **Symons, J.**; **Smirni, E. (2008):** Anomaly? application change? or workload change? Towards automated detection of application performance anomaly and change. In Proceedings of the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC., DSN '08, 452–461.

**Cherkasova, L.**; **Ozonat, K.**; **Mi, N.**; **Symons, J.**; **Smirni, E. (2009):** Automated Anomaly Detection and Performance Modeling of Enterprise Applications. *ACM Transactions on Computer Systems*, vol. 27 no. 3, 6:1–6:32, ISSN 0734–2071.

**Conover, W. (1971):** Practical Nonparametric Statistics. John Wiley & Sons, New York, NY, USA, ISBN 9780471168515.

**CORE (2013):** CORE Conference Rankings. ⟨URL: `http://www.core.edu.au/index.php/conference-rankings`⟩ last accessed 2014-10-15.

**Cortellessa, V.**; **Di Marco, A.**; **Inverardi, P. (2011):** Performance Modeling Notations. In Model-Based Software Performance Analysis. Springer, Berlin/Heidelberg, Germany, ISBN 978–3–642–13620–7, 35–63.

**Danciu, A.**; **Brunnert, A.**; **Krcmar, H. (2014):** Towards Performance Awareness in Java EE Development Environments. In Proceedings of the Symposium on Software Performance. Stuttgart, Germany, SOSP '14, 152–159.

**Danciu, A.**; **Chrusciel, A.**; **Brunnert, A.**; **Krcmar, H. (2015a):** Performance Awareness in Java EE Development Environments. In **Beltrán, M.**; **Knottenbelt, W.**; **Bradley, J. eds.:** Computer Performance Engineering. vol. 9272, Springer International Publishing, ISBN 978–3–319–23266–9, 146–160.

**Danciu, A.**; **Kroß, J.**; **Brunnert, A.**; **Willnecker, F.**; **Vögele, C.**; **Kapadia, A.**; **Krcmar, H. (2015b):** Landscaping Performance Research at the ICPE and Its Predecessors: A Systematic Literature Review. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '15, ISBN 978–1–4503–3248–4, 91–96.

**DeMichiel, L.**; **Keith, M. (2006):** JSR 220: Enterprise JavaBeans, Version 3.0 - EJB 3.0 Simplified API..

**Dlugi, M.**; **Brunnert, A.**; **Krcmar, H. (2015):** Model-based Performance Evaluations in Continuous Delivery Pipelines. In Proceedings of the 1st International Workshop on Quality-Aware DevOps. ACM, New York, NY, USA, QUDOS '15, ISBN 978–1–4503–3817–2, 25–26.

**Economou, D.**; **Rivoire, S.**; **Kozyrakis, C.**; **Ranganathan, P. (2006):** Full-system power analysis and modeling for server environments. In Workshop on Modeling, Benchmarking, and Simulation. Boston, Massachusetts, USA.

**ERA (2010):** ERA 2010 Journal Rankings. ⟨URL: `http://www.research.swinburne.edu.au/researchers/publication-collections/era/journals/2010/search.html`⟩ last accessed 2014-10-15.

**Erich, F.**; **Amrit, C.**; **Daneva, M. (2014):** A Mapping Study on Cooperation between Information System Development and Operations. In **Jedlitschka, A.**; **Kuvaja, P.**; **Kuhrmann, M.**; **Männistö, T.**; **Münch, J.**; **Raatikainen, M. eds.:** Product-Focused Software Process Improvement. vol. 8892, Springer International Publishing, ISBN 978–3–319–13834–3, 277–280.

**EU, E. C. (2013):** European e-Competence Framework Version 2.0. ⟨URL: `http://www.ecompetences.eu/`⟩ last accessed 2014-10-15.

**Faban (2012):** Faban Harness and Benchmark Framework. ⟨URL: `http://java.net/projects/faban/`⟩ last accessed 2014-10-15.

**Fan, X.**; **Weber, W.-D.**; **Barroso, L. A. (2007):** Power Provisioning for a Warehouse-sized Computer. *Computer Architecture News*, vol. 35 no. 2, 13–23, ISSN 0163–5964.

**Ferstl, O.**; **Sinz, E. (2008):** Grundlagen der Wirtschaftsinformatik. Oldenbourg, Munich, Germany, 5. Auflage, ISBN 9783486587555.

**Franks, G.**; **Al-Omari, T.**; **Woodside, M.**; **Das, O.**; **Derisavi, S. (2009):** Enhanced Modeling and Solution of Layered Queueing Networks. *IEEE Transactions on Software Engineering*, vol. 35 no. 2, 148–161, ISSN 0098–5589.

**Grabski, B.**; **Günther, S.**; **Herden, S.**; **Krüger, L.**; **Rautenstrauch, C.**; **Zwanziger, A. (2007):** Very Large Business Applications. *Informatik-Spektrum*, vol. 30 no. 4, 259–263, ISSN 0170–6012.

**Grassi, V.**; **Mirandola, R.**; **Sabetta, A. (2007):** Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software*, vol. 80 no. 4, 528 – 558, ISSN 0164–1212.

**Grinshpan, L. (2012):** Solving Enterprise Applications Performance Puzzles: Queuing Models to the Rescue. 1. edition. Wiley-IEEE Press, Piscataway, NJ, USA, ISBN 9781118061572.

**Heger, C.**; **Happe, J.**; **Farahbod, R. (2013):** Automated Root Cause Isolation of Performance Regressions During Software Development. In Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '13, ISBN 978–1–4503–1636–1, 27–38.

**Henning, J. L. (2006):** SPEC CPU2006 Benchmark Descriptions. *Computer Architecture News*, vol. 34 no. 4, 1–17, ISSN 0163–5964.

**Hevner, A. R.**; **Ram, S.**; **March, S. T.**; **Park, J. (2004):** Design Science in Information Systems Research. *MIS Quarterly*, vol. 28 no. 1, 75–105, ISSN 1571–0270.

**Hoorn, A. van (2014):** Model-Driven Online Capacity Management for Component-Based Software Systems. Department of Computer Science, Kiel University, Kiel, Germany, Kiel Computer Science Series 2014/6, ISBN 978–3–7357–5118–8.

**Huber, N.**; **Quast, M. von**; **Hauck, M.**; **Kounev, S. (2011):** Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments. In Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011). SciTePress, ISBN 978–989–8425–52–2, 563 – 573.

**Humble, J.**; **Farley, D. (2010):** Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. 1. edition. Addison-Wesley Professional, Upper Saddle River, NJ, USA, ISBN 9780321601919.

**Humble, J.**; **Molesky, J. (2011):** Why Enterprises Must Adopt Devops to Enable Continuous Delivery. *Cutter IT Journal*, vol. 24 no. 8, 6–12, ISSN 1522–7383.

**Jain, R. (1991):** The Art of Computer Systems Performance Analysis. Wiley Computer Publishing, John Wiley & Sons, Inc., New York, NY, USA, ISBN 0471503363.

**Jamshidi, M. (2011):** System of Systems Engineering: Innovations for the Twenty-First Century. In Wiley Series in Systems Engineering and Management. vol. 58, John Wiley & Sons, New York, NY, USA, ISBN 1118210735.

**Jiang, Z. M. J.**; **Brunnert, A. (2015):** LT 2015: The Fourth International Workshop on Large-Scale Testing. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '15, ISBN 978–1–4503–3248–4, 345–346.

**Johann, T.**; **Dick, M.**; **Naumann, S.**; **Kern, E. (2012):** How to Measure Energy-efficiency of Software: Metrics and Measurement Results. In Proceedings of the First International Workshop on Green and Sustainable Software. IEEE Press, Piscataway, NJ, USA, GREENS '12, ISBN 978–1–4673–1832–7, 51–54.

**Jwo, J.-S.**; **Wang, J.-Y.**; **Huang, C.-H.**; **Two, S.-J.**; **Hsu, H.-C. (2011):** An Energy Consumption Model for Enterprise Applications. In Proceedings of the IEEE/ACM International Conference on Green Computing and Communications. IEEE, Washington, DC, USA, ISBN 978–0–7695–4466–3, 216–219.

**Kalibera, T.**; **Bulej, L.**; **Tůma, P. (2005a):** Automated detection of performance regressions: the mono experience. In Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems., MASCOTS '05, ISSN 1526–7539, 183–190.

**Kalibera, T.**; **Bulej, L.**; **Tůma, P. (2005b):** Benchmark precision and random initial state. In Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunications Systems. SCS, SPECTS '05, 853–862.

**Kalibera, T.**; **Tůma, P. (2006):** Precise Regression Benchmarking with Random Effects: Improving Mono Benchmark Results. In Proceedings of the Third European Conference on Formal Methods and Stochastic Models for Performance Evaluation.

Springer-Verlag, Berlin, Heidelberg, EPEW '06, ISBN 3–540–35362–3, 978–3–540–35362–1, 63–77.

**King, B. (2004):** Performance Assurance for IT Systems. Auerbach Publications - A CRC Press Company, Boca Raton, FL, USA, ISBN 0849327784.

**Kitchenham, B.**; **Charters, S. (2007):** Guidelines for performing systematic literature reviews in software engineering. Technical report, EBSE Technical Report EBSE-2007-01 – technical report.

**Koegel, M.**; **Helming, J. (2010):** EMFStore: a model repository for EMF models. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. Cape Town, South Africa, ICSE '10, ISSN 0270–5257, 307–308.

**Kounev, S. (2006):** Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, vol. 32 no. 7, 486–502, ISSN 0098–5589.

**Kounev, S. (2005):** Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction. Shaker Verlag, Ph.D. Thesis, Technische Universität Darmstadt, Germany, Aachen, Germany.

**Kounev, S.**; **Brosig, F.**; **Huber, N. (2014):** The Descartes Modeling Language. Department of Computer Science, University of Wuerzburg – technical report.

**Kounev, S.**; **Buchmann, A. (2003):** Performance Modeling and Evaluation of Large-Scale J2EE Applications. In International Computer Measurement Group (CMG) Conference on Resource Management and Performance Evaluation of Enterprise Computing Systems. vol. 1, Computer Measurement Group, Dallas, Texas, USA, 273–284.

**Koziolek, A. (2013):** Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes. vol. 7, The Karlsruhe Series on Software Design and Quality. KIT Scientific Publishing, Karlsruhe, Karlsruhe, Germany, ISBN 978–3–86644–973–2.

**Koziolek, H. (2010):** Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, vol. 67 no. 8, 634–658, ISSN 0166–5316.

**Krcmar, H. (2010):** Informationsmanagement. Springer, Berlin/Heidelberg, Germany, 5. Auflage, ISBN 978–3–642–04285–0.

**Krogmann, K. (2012):** Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis. vol. 4, The Karlsruhe Series on Software Design and Quality. KIT Scientific Publishing, Karlsruhe, Germany, ISBN 9783866448049.

**Kroß, J.**; **Brunnert, A.**; **Prehofer, C.**; **Runkler, T. A.**; **Krcmar, H. (2015a):** Model-based Performance Evaluation of Large-Scale Smart Metering Architectures. In Proceedings of the 4th International Workshop on Large-Scale Testing. ACM, New York, NY, USA, LT '15, ISBN 978–1–4503–3337–5, 9–12.

**Kroß, J.**; **Brunnert, A.**; **Prehofer, C.**; **Runkler, T.**; **Krcmar, H. (2015b):** Stream Processing on Demand for Lambda Architectures. In **Beltrán, M.**; **Knottenbelt, W.**; **Bradley, J. eds.:** Computer Performance Engineering. vol. 9272, Springer International Publishing, ISBN 978–3–319–23266–9, 243–257.

**Kuperberg, M. (2010):** Quantifying and Predicting the Influence of Execution Platform on Software Component Performance. vol. 5, The Karlsruhe Series on Software Design and Quality. KIT Scientific Publishing, Karlsruhe, Germany, ISBN 9783866447417.

**Levy, Y.**; **Ellis, T. J. (2006):** A systems approach to conduct an effective literature review in support of information systems research. *Informing Science Journal*, vol. 9, 181–212, ISSN 1521–4672.

**Li, H.**; **Casale, G.**; **Ellahi, T. (2010):** SLA-driven Planning and Optimization of Enterprise Applications. In Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '10, ISBN 978–1–60558–563–5, 117–128.

**Libič, P.**; **Tůma, P.**; **Bulej, L. (2009):** Issues in Performance Modeling of Applications with Garbage Collection. In Proceedings of the 1st International Workshop on Quality of Service-oriented Software Systems. ACM, New York, NY, USA, QUASOSS '09, ISBN 978–1–60558–709–7, 3–10.

**Liu, T.-K.**; **Kumaran, S.**; **Luo, Z. (2001):** Layered queueing models for Enterprise JavaBean applications. In IEEE International Enterprise Distributed Object Computing Conference. Seattle, Washington, USA, 174–178.

**Liu, T.-K.**; **Shen, H.**; **Kumaran, S. (2004):** A Capacity Sizing Tool for a Business Process Integration Middleware. In Proceedings of the IEEE International Conference on E-Commerce Technology. IEEE, Washington, DC, USA, ISBN 0–7695–2098–7, 195–202.

**Liu, Y.**; **Gorton, I.**; **Zhu, L. (2007):** Performance Prediction of Service-Oriented Applications Based on an Enterprise Service Bus. In International Computer Software and Applications Conference. Beijing, China, COMPSAC '07, ISSN 0730–3157, 327–334.

**Liu, Y. D.**; **Smith, S. F. (2006):** A Formal Framework for Component Deployment. In Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications. ACM, New York, NY, USA, OOPSLA '06, ISBN 1–59593–348–4, 325–344.

**Mania, D.**; **Murphy, J. (2002):** Framework for Predicting the Performance of Component-Based Systems. In IEEE International Conference on Software, Telecommunications and Computer Networks. Italy, SoftCOM '02, 46–50.

**Mayer, M.**; **Gradl, S.**; **Schreiber, V.**; **Wittges, H.**; **Krcmar, H. (2011):** A Survey on Performance Modelling and Simulation of SAP Enterprise Resource Planning Systems. In The 10th International Conference on Modeling and Applied Simulation. Diptem University of Genoa, Genoa, Italy, ISBN 978–88–903724–5–2, 347–352.

**McGuinness, D.**; **Murphy, L.**; **Lee, A. (2004):** Issues in Developing a Simulation Model of an EJB System. In International Conference of the Computer Measurement Group (CMG) on Resource Management and Performance Evaluation of Enterprise Computing Systems. Las Vegas, Nevada, USA, 173–182.

**Menascé, D. (2002a):** Load Testing of Web Sites. *Internet Computing, IEEE*, vol. 6 no. 4, 70–74, ISSN 1089–7801.

**Menascé, D. A. (2002b):** Load Testing, Benchmarking, and Application Performance Management for the Web. In Proceedings of the Computer Measurement Group (CMG) Conference. CMG, Reno, Nevada, USA, 271–282.

**Menascé, D. A.**; **Almeida, V. A. F. (2002):** Capacity Planning for Web Services: Metrics, Models, and Methods. Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN 9780130659033.

**Menascé, D. A.**; **Almeida, V. A.**; **Lawrence, F.**; **Dowdy, W.**; **Dowdy, L. (2004):** Performance by Design: Computer Capacity Planning by Example. vol. 1, Pearson Education, Inc., Upper Saddle River, New Jersey, USA, ISBN 9780130659033.

**Merkle, P.**; **Henß, J. (2011):** EventSim – An Event-driven Palladio Software Architecture Simulator. In Karlsruhe Reports in Informatics (Palladio Days 2011 Proceedings). vol. 32, KIT, Fakultät für Informatik, Karlsruhe, Germany, ISSN 2190–4782, 15–22.

**Mi, N.**; **Cherkasova, L.**; **Ozonat, K.**; **Symons, J.**; **Smirni, E. (2008):** Analysis of Application Performance and Its Change via Representative Application Signatures. In Proceedings of the IEEE Network Operations and Management Symposium., NOMS '08, ISSN 1542–1201, 216–223.

**Microsystems, S. (2006):** Java Management Extensions (JMX) Specification, version 1.4..

**Mordani, R. (2007):** Java Servlet Specification v2.5..

**Nguyen, T. H.**; **Adams, B.**; **Jiang, Z. M.**; **Hassan, A. E.**; **Nasser, M.**; **Flora, P. (2012):** Automated Detection of Performance Regressions Using Statistical Process Control Techniques. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '12, ISBN 978–1–4503–1202–8, 299–310.

**O'Brien, L.**; **Brebner, P.**; **Gray, J. (2008):** Business Transformation to SOA: Aspects of the Migration and Performance and QoS Issues. In International Workshop on Systems Development in SOA Environments. Leipzig, Germany, ISBN 978–1–60558–029–6, 35–40.

**OMG (2005):** UML Profile for Schedulability, Performance, and Time. Object Management Group (OMG)..

**OMG (2011):** UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems. Object Management Group (OMG)..

**Patel, C. D.**; **Shah, A. J. (2005):** Cost model for planning, development and operation of a data center. Palo Alto, CA, USA.

**Perez, J. F.**; **Wang, W.**; **Casale, G. (2015):** Towards a DevOps Approach for Software Quality Engineering. In Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development. ACM, New York, NY, USA, WOSP '15, ISBN 978–1–4503–3340–5, 5–10.

**Petriu, D. B.**; **Woodside, M. (2007):** An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *Software & Systems Modeling*, vol. 6 no. 2, 163–184, ISSN 1619–1366.

**Poess, M.**; **Nambiar, R. O. (2008):** Energy Cost, the Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results. *Proceedings of the VLDB Endowment*, vol. 1 no. 2, 1229–1240, ISSN 2150–8097.

**Reussner, R.**; **Becker, S.**; **Burger, E.**; **Happe, J.**; **Hauck, M.**; **Koziolek, A.**; **Koziolek, H.**; **Krogmann, K.**; **Kuperberg, M. (2011):** The Palladio Component Model. Karlsruhe, Germany, KIT, Fakultät für Informatik – technical report, ISSN 2190–4782.

**Reussner, R.**; **Becker, S.**; **Happe, J.**; **Koziolek, H.**; **Krogmann, K.**; **Kuperberg, M. (2007):** The Palladio Component Model. Universität Karlsruhe, Karlsruhe, Germany – technical report, ISSN 1432–7864.

**Rivoire, S.**; **Shah, M.**; **Ranganathan, P.**; **Kozyrakis, C.**; **Meza, J. (2007):** Models and Metrics to Enable Energy-Efficiency Optimizations. *Computer*, vol. 40 no. 12, 39–48, ISSN 0018–9162.

**Rivoire, S.**; **Ranganathan, P.**; **Kozyrakis, C. (2008):** A Comparison of High-level Full-system Power Models. In Proceedings of the Conference on Power Aware Computing and Systems. USENIX Association, Berkeley, CA, USA.

**Sambasivan, R. R.**; **Zheng, A. X.**; **De Rosa, M.**; **Krevat, E.**; **Whitman, S.**; **Stroucken, M.**; **Wang, W.**; **Xu, L.**; **Ganger, G. R. (2011):** Diagnosing Performance Changes by Comparing Request Flows. In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, Berkeley, CA, USA, NSDI'11, 43–56.

**Schmietendorf, A. (2001):** Prozess-Konzepte zur Gewährleistung des Software-Performance-Engineerings in großen IT-Organisationen. In **Dumke, R. ed.:** Magdeburger Schriften zum Empirischen Software Engineering. Shaker Verlag, Aachen, Germany, ISBN 3–8265–9590–4.

**Shannon, B. (2006):** Java Platform, Enterprise Edition (Java EE) Specification, v5..

**Sim, S. E.**; **Easterbrook, S.**; **Holt, R. C. (2003):** Using Benchmarking to Advance Research: A Challenge to Software Engineering. In Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society, Washington, DC, USA, ICSE '03, ISBN 0–7695–1877–X, 74–83.

**Smirnov, N. (1948):** Table for Estimating the Goodness of Fit of Empirical Distributions. *Annals of Mathematical Statistics*, vol. 19 no. 2, 279–281.

**Smith, C. U. (1981):** Increasing Information Systems Productivity by Software Performance Engineering. In **Deese, D. R.**; **Bishop, R. J.**; **Mohr, J. M.**; **Artis, H. P. eds.:** Seventh International Computer Measurement Group Conference. Computer Measurement Group, New Orleans, LA, USA, 5–14.

**Smith, C. U. (2007):** Introduction to Software Performance Engineering: Origins and Outstanding Problems. In Proceedings of the 7th International Conference on Formal Methods for Performance Evaluation. Springer-Verlag, Berlin, Heidelberg, SFM'07, ISBN 978–3–540–72482–7, 395–428.

**Smith, C. U.**; **Lladó, C. M.**; **Cortellessa, V.**; **Marco, A. D.**; **Williams, L. G. (2005):** From UML Models to Software Performance Results: An SPE Process Based on XML Interchange Formats. In Proceedings of the 5th International Workshop on Software and Performance. ACM, New York, NY, USA, WOSP '05, ISBN 1–59593–087–6, 87–98.

**SPEC (2002):** SPEC jAppServer Development Page. ⟨URL: `http://www.spec.org/osg/jAppServer/`⟩ last accessed 2014-10-15.

**SPEC (2012):** SPECjEnterprise2010. ⟨URL: `http://www.spec.org/jEnterprise2010/`⟩ last accessed 2014-10-15.

**Spinner, S.**; **Casale, G.**; **Brosig, F.**; **Kounev, S. (2015):** Evaluating approaches to resource demand estimation. *Performance Evaluation*, vol. 92, 51 – 71, ISSN 0166–5316.

**Spinner, S.**; **Casale, G.**; **Zhu, X.**; **Kounev, S. (2014):** LibReDE: A Library for Resource Demand Estimation. In Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '14, ISBN 978–1–4503–2733–6, 227–228.

**Tiwari, N.**; **Nair, K. C. (2010):** Performance extrapolation that uses industry benchmarks with performance models. In International Symposium on Peformance Evaluation of Computer and Telecommunication Systems. Ottawa, ON, USA, SPECTS '10, 301–305.

**Vaquero, L. M.**; **Rodero-Merino, L.**; **Buyya, R. (2011):** Dynamically scaling applications in the cloud. *Computer Communication Review*, vol. 41 no. 1, 45–52, ISSN 0146–4833.

**Vögele, C.**; **Brunnert, A.**; **Danciu, A.**; **Tertilt, D.**; **Krcmar, H. (2014):** Using Performance Models to Support Load Testing in a Large SOA Environment. In Proceedings of the Third International Workshop on Large Scale Testing. ACM, New York, NY, USA, LT '14, ISBN 978–1–4503–2762–6, 5–6.

**Webster, J.**; **Watson, R. T. (2002):** Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Quarterly*, vol. 26 no. 2, xiii–xxiii, ISSN 0276–7783.

**Wert, A.**; **Happe, J.**; **Happe, L. (2013):** Supporting Swift Reaction: Automatically Uncovering Performance Problems by Systematic Experiments. In Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, Piscataway, NJ, USA, ICSE '13, ISBN 978–1–4673–3076–3, 552–561.

**Wert, A.**; **Happe, J.**; **Westermann, D. (2012):** Integrating software performance curves with the palladio component model. In Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering. ACM, New York, NY, USA, ICPE '12, ISBN 978–1–4503–1202–8, 283–286.

**Willnecker, F.**; **Brunnert, A.**; **Gottesheim, W.**; **Krcmar, H. (2015a):** Using Dynatrace Monitoring Data for Generating Performance Models of Java EE Applications. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '15, ISBN 978–1–4503–3248–4, 103–104.

**Willnecker, F.**; **Brunnert, A.**; **Krcmar, H. (2014a):** Model-based Energy Consumption Prediction for Mobile Applications. In **Gómez, J. M.**; **Sonnenschein, M.**; **Vogel, U.**; **Winter, A.**; **Rapp, B.**; **Giesen, N. eds.:** Proceedings of the Workshop on Energy Aware Software Development (EASED) @ EnviroInfo 2014. BIS-Verlag, Oldenburg, Germany, ISBN 978-3-8142-2317-9, 747–752.

**Willnecker, F.**; **Brunnert, A.**; **Krcmar, H. (2014b):** Predicting Energy Consumption by Extending the Palladio Component Model. In Proceedings of the Symposium on Software Performance. Stuttgart, Germany, SOSP '14, 177–188.

**Willnecker, F.**; **Dlugi, M.**; **Brunnert, A.**; **Spinner, S.**; **Kounev, S.**; **Gottesheim, W.**; **Krcmar, H. (2015b):** Comparing the Accuracy of Resource Demand Measurement and Estimation Techniques. In **Beltrán, M.**; **Knottenbelt, W.**; **Bradley, J. eds.:** Computer Performance Engineering. vol. 9272, Springer International Publishing, ISBN 978–3–319–23266–9, 115–129.

**WKWI (2008):** WI: WI-Journalliste 2008. ⟨URL: `http://gcc.upb.de/K-Pool/WKWI-Ranking`⟩ last accessed 2014-10-15.

**Woodside, C.**; **Neilson, J.**; **Petriu, D.**; **Majumdar, S. (1995):** The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers*, vol. 44 no. 1, 20–34, ISSN 0018–9340.

**Woodside, M. (2013):** Tutorial Introduction to Layered Modeling of Software Performance. Carleton University. ⟨URL: `http://www.sce.carleton.ca/rads/lqns/lqn-documentation/tutorialh.pdf`⟩ last accessed 2014-10-15.

**Woodside, M.**; **Franks, G.**; **Petriu, D. C. (2007):** The Future of Software Performance Engineering. In Future of Software Engineering. IEEE Computer Society, Washington, DC, USA, FOSE '07, ISBN 0–7695–2829–5, 171–187.

**Wu, X.**; **Woodside, M. (2004):** Performance modeling from software components. *SIGSOFT Software Engineering Notes*, vol. 29 no. 1, 290–301, ISSN 0163–5948.

**Zhu, L.**; **Liu, Y.**; **Bui, N. B.**; **Gorton, I. (2007):** Revel8or: Model Driven Capacity Planning Tool Suite. In Proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society, Washington, DC, USA, ICSE '07, ISBN 0–7695–2828–7, 797–800.