TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl für Wirtschaftsinformatik (I 17)
Prof. Dr. Helmut Krcmar

Automatic Extraction and Selection of Workload Specifications for Load Testing and Model-Based Performance Prediction

Christian Vögele, M.Sc.

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl für Wirtschaftsinformatik (I 17)
Prof. Dr. Helmut Krcmar

Automatic Extraction and Selection of Workload Specifications for Load Testing and Model-Based Performance Prediction

Christian Vögele, 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:	Prof. Dr. Martin Bichler
Prüfer der Dissertation:	1. Prof. Dr. Helmut Krcmar
	2. Prof. Dr. Alexander Pretschner

Die Dissertation wurde am 04.10.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 23.04.2018 angenommen.

Acknowledgement

The present thesis arose during my six years as research associate at the performance management group at fortiss. Initially, I would like to take this opportunity to thank everybody who supported me finishing this dissertation.

First of all, I would like to thank Prof. Dr. Helmut Krcmar for the fact that he gave me the opportunity to finish my dissertation under his supervision. His questions, feedback, and suggestions helped me to continuously improve my dissertation. I also wish to thank Prof. Dr. Alexander Pretschner as second examiner and Prof. Dr. Martin Bichler for taking the chair of the examination board.

My thanks also goes to my former and current colleagues of the performance management group at the fortiss institute for their valuable advices, exchanges of experiences, suggestions and discussions around the topic of my dissertation.

My special thank goes to Dr. André van Hoorn from the University of Stuttgart (Institute of Software Technology - Reliable Software Systems (RSS)). The cooperation in the development of the WESSBAS approach was very productive. He was always available to discuss and realize new ideas and practical implementations.

Finally, my biggest thank goes to my wonderful wife Verena and my family. My family constantly supported me during my education and my personal development. Thus, I could complete my studies and start with the doctoral studies. Particularly, I admire the patience and support of my wife during the preparation of my thesis. She helped me throughout all my ups and downs. Without her support this work would have not been possible.

Munich, Germany, June 2018 Christian Vögele

Abstract

Problem To ensure that application systems are able to meet performance objectives, such as given throughputs, response times, and resource-utilization levels, load testing is an important and required testing procedure. Based on these objectives, one of the main challenges in this area is to derive workload specifications that are either representative compared to the real workload or are able to detect performance problems under load. In case a workload specifications is not specified in a way that it is able to test if given performance objectives are met the testing results are useless or the tests must be repeated. Moreover, the manual creation of workload specifications is difficult, time consuming, and error-prone. In response to these challenges, the goal of this dissertation is to propose an approach that supports the automatic extraction of workload specifications that match given performance objectives using measurement and model-based performance evaluation techniques.

Research Method In order to achieve the research goal this dissertation follows a design-science oriented strategy with the aim to create new and innovative artifacts for workload extraction. This strategy is based on existing research in the field of software performance and continuously develops, evaluates, and refines new software artifacts. The evaluation of these artifacts is conducted with the evaluation methods literature review, prototyping, controlled experiments, and scenarios.

Results In a first step, an approach is developed that enables the automatic extraction of representative and probabilistic workload specifications. A domain-specific language is introduced allowing tool- and system independent workload specification. Instances of this language are automatically extracted from system logs. During the extraction process user groups showing similar usage patterns are identified with clustering techniques. Based on these instances executable workload specifications of load testing tools are automatically generated. Furthermore, we transform instances of the domain-specific language into workload specification of architecture-level performance models.

To enable the selection of workload specifications following given performance objectives we define a multi-objective optimization process. This process derives workload specifications from performance models that are extracted in the previous step. A performance engineer can specify which performance objectives should be considered during the selection process. We defined a set of common performance objectives that comprise goals for resource-utilization levels, response times, throughput, number of test cases, system coverage, and representativeness. For each selected objective it can be defined if it should be maximized/minimized or if a predefined threshold value should be reached. Afterward, different workload alternatives are encoded into performance models and are simulated and assessed based on these objectives. A multi-objective optimization using an evolutionary algorithm is applied to evaluate these alternatives. The resulting candidates are presented to the performance engineer to make trade-off decisions. Finally, executable load test scripts are generated based on the selected solution(s).

Research Implications The selection of workload specifications that follow given performance objectives has been a research topic for several decades. Research in this area mainly focuses on extracting either representative workloads or on extracting workloads that are able to detect performance problems under load. There is no approach that addresses both goals. This dissertation addresses this gap by proposing the first automatic approach that derives workload specifications from performance models following multiple performance goals using a multi-objective optimization approach. The introduced domain-specific language is extracted from runtime data and can be used as intermediate language to generate executable load tests and workload specifications for model-based performance predictions. We extract the workloads in a way that probabilistic user behavior models are derived while ensuring that valid sequences of user requests are generated. As a result, we enable to extract system and tool-agnostic workload models as well as to integrate workload modeling for measurement and model-based prediction approaches.

Practical Implications The ability to automatically generate load test scripts and workload specifications of performance models significantly reduces the effort to specify workloads that fulfill given performance objectives. Moreover, the application of existing model-based performance evaluation techniques in practice is simplified. The approach is applicable for all session-based application systems and requires no detailed knowledge about workload extraction.

Limitations There are several limitations that must be considered. The main limitation of the multi-objective optimization process is that no global optimization criterion exists to which this approach can be compared. As a result, it cannot be guaranteed that the best workloads are found. Moreover, the time it takes to conduct the optimization might take a lot of time and is quite resource intensive.

All artifacts developed during this thesis are evaluated using an industry benchmark. However, to assess the practicality in practice the evaluation with real world applications is required. Moreover, we only evaluated the developed artifacts using Java Enterprise Edition (EE) applications. This comes from the fact that we used an evaluated performance model generator that is able to model the system-specific parts of Java EE applications. Since not for all application systems a performance model generator is available it might be required to model the system-specific parts of the performance models manually, which can be rather time consuming.

Zusammenfassung

Problem Die Durchführung von Lasttests ist ein wichtiger und notwendiger Bestandteil in der Softwarequalitätssicherung um sicherzustellen, dass Anwendungssysteme vorgegebene Performance-Ziele, beispielsweise Antwortzeiten und Ressourcennutzung, einhalten können. Eine der größten Herausforderungen in diesem Bereich ist es, basierend auf diesen Performance-Zielen, Lasttests zu entwerfen (Lasttestdesigns), die entweder repräsentativ zum produktiven Workload sind oder Performance-Probleme unter Last identifizieren können. Falls Lasttests nicht in der Lage sind diese Ziele zu berücksichtigen, sind sie Ergebnisse oftmals unbrauchbar und die Tests müssen wiederholt werden. Zudem kann die manuelle Erstellung von Lasttests schwierig, fehleranfällig, und zeitaufwändig sein. Um diesen Herausforderungen zu begegnen, ist das Ziel dieser Dissertation einen Ansatz zu entwickeln, der die automatische Extraktion von Lasttestdesigns für messbasierte- und modellbasierte Performance-Evaluations-Verfahren innerhalb vorgegebener Performance-Ziele, ermöglicht.

Forschungsmethode Diese Dissertation folgt einer designorientierten Forschungsstrategie, welche zum Ziel hat, neue und innovative Artefakte für die Extraktion von Workloads zu erstellen. Diese Strategie basiert auf Forschung im Bereich Software-Performance und entwickelt, evaluiert und verfeinert kontinuierlich neue Softwareartefakte. Die Evaluation dieser Artefakte wird mit den Evaluationsmethoden Literaturrecherche, Prototyping, kontrollierte Experimente und Szenarios durchgeführt.

Ergebnisse Im ersten Schritt dieser Dissertation wurde einen Ansatz entwickelt, der die automatische Extraktion von repräsentativen und probabilistischen Workload-Spezifikationen ermöglicht. Hierfür wurde eine domänenspezifische Sprache vorgestellt, die eine werkzeug- und systemunabhängige Spezifikation von Workloads ermöglicht. Instanzen dieser Sprache können automatisch aus Laufzeitdaten (z.B. Systemlogs) extrahiert werden. Während des Extraktionsprozesses werden zudem Nutzergruppen mit ähnlichen Verhaltensmustern mithilfe von Clustering-Techniken identifiziert. Basierend auf diesen Instanzen werden automatisch ausführbare Workload-Spezifikationen für Lasttest-Tools generiert. Darüber hinaus können Instanzen der domänenspezifischen Sprache in Workload-Spezifikationen von Performance-Modellen transformiert werden.

Im Rahmen dieser Arbeit wird zudem ein mehrkriterieller Optimierungsprozess vorgestellt, der die Auswahl von Lasttest-Designs innerhalb vorgegebener Ziele ermöglicht. Dieser Prozess leitet Lasttestdesigns aus Performance-Modellen ab, die im vorherigen Schritt extrahiert wurden. Ein Performance-Experte kann in diesem Prozess als erstes festlegen, welche Performance-Ziele bei der Auswahl berücksichtigt werden sollen. Hierfür wurde eine Reihe von üblichen Performance-Zielen definiert, die Ziele für Ressourcennut-

zung, Antwortzeiten, Durchsatz, Anzahl der Testfälle, Systemabdeckung und Repräsentativität umfassen. Für jedes ausgewählte Performance-Ziel kann vorgegeben werden, ob es maximiert/minimiert werden soll oder ob ein vorgegebener Schwellwert erreicht werden soll. Anschließend werden verschiedene Workload-Alternativen in Performance-Modelle codiert, simuliert und anhand dieser Ziele bewertet. Zur Bewertung dieser Alternativen wird eine mehrkriterielle Optimierung mithilfe eines evolutionären Algorithmus verwendet. Basierend auf den daraus resultierenden Kandidaten kann der Performance-Experte sich für ein Lasttestdesign entscheiden. Abschließend werden ausführbare Lasttest-Skripte aus der ausgewählten Lösung erzeugt.

Beitrag zur Forschung Die Erstellung von Lasttest-Designs, die vorgegebene Performance-Ziele einhalten können, ist seit Jahrzehnten ein Forschungsthema. Die Forschung in diesem Bereich konzentriert sich entweder auf die Extraktion von repräsentativen Workloads oder auf Workloads, die in der Lage sind, Performance-Probleme unter Last zu erkennen. Es gibt aktuell keinen Ansatz der beide Ziele verfolgt. Diese Dissertation versucht diese Lücke zu schließen. Hierfür wird der erste (semi-)automatische Ansatz vorgeschlagen, der unter Verwendung eines mehrkriteriellen Optimierungsansatzes Lasttest-Designs aus Performance-Modellen ableitet. Die entwickelte domänenspezifische Sprache wird aus Laufzeitdaten extrahiert und kann verwendet werden, um sowohl ausführbare Lasttests als auch Workload-Spezifikationen für modellbasierte Performance-Vorhersagen zu generieren. Die extrahierten Workloads können probabilistisches Nutzerverhaltensmodel abbilden, stellen aber gleichzeitig sicher, dass gültige Sequenzen von Nutzerrequests generiert werden. Somit können werkzeug- und systemunabhängige Workloadmodelle extrahieren werden. Zudem wird die Workload-Modellierung für mess- und modellbasierte Vorhersageansätze integriert.

Beitrag zur Praxis Die Möglichkeit Lasttestskripte und Workload-Spezifikationen für Performance-Modelle automatisch zu generieren kann den Aufwand des Performance-Testens erheblich reduzieren. Darüber hinaus wird der Einsatz modellbasierter Performance-Vorhersagen in der Praxis vereinfacht. Der Ansatz ist für alle Session-basierten Anwendungssysteme einsetzbar und erfordert keine detaillierte Kenntnis der Workload-Extraktion.

Limitationen Die größte Limitation des mehrkriterieller Optimierungsprozesses besteht darin, dass kein globales Optimierungskriterium existiert, mit dem dieser Ansatz verglichen werden kann. Somit kann nicht garantiert werden, dass die besten Lasttestdesigns gefunden werden. Darüber hinaus kann die Optimierung zum aktuellen Zeitpunkt viel Zeit in Anspruch nehmen und ist ressourcenintensiv.

Alle Artefakte, die während dieser Arbeit entwickelt wurden, wurden mit einem Industrie-Benchmark evaluiert. Um die praktische Anwendbarkeit in der Praxis zu beurteilen, ist zudem die Evaluation mit realen Anwendungen erforderlich. Darüber hinaus haben wir die entwickelten Artefakte ausschließlich mit Java Enterprise Edition (EE) Anwendungen evaluiert. Dies ergibt sich aus der Tatsache, dass wir einen Performance-Modell-Generator verwendet haben, der in der Lage ist, die systemspezifischen Teile von Java EE Anwendungen zu modellieren. Da nicht für alle Anwendungssysteme ein Performance-Modell-Generator zur Verfügung steht, kann es erforderlich sein, die systemspezifischen Teile der Performance-Modelle manuell zu modellieren, was zeitaufwendig sein kann.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

ACM	Association for Computing Machinery	22
APM	Application Performance Management	3
BISE	Business & Information Systems Engineering	23
CBMG	Customer Behavior Model Graph	63
CPU	Central Processing Unit	39
CRM	Customer Relationship Management	38
DML	Descartes Modeling Language	15
DS	Design Space	136
DSL	Domain-specific Language	59
EA	Evolutionary Algorithm	5
ED	Euclidean Distance	40
EFSM	Extended Finite State Machine	52
EJB	Enterprise Java Bean	17
EMF	Eclipse Modeling Framework	69
EVOP	Evolutionary Operator	20
FCFS	First Come, First Served	14
FSM	Finite State Machine	63
GaAs	Guards and Actions	59
HDD	Hard Disk Drive	15
HTTP	Hypertext Transfer Protocol	31
ICPE	International Conference on Performance Engineering	25
JSP	Java Server Pages	84
LIMBO	Load Intensity Modeling Tool	64
LQN	Layered Queuing Network	14
LT	International Workshop on Large-Scale Testing	26
LTS	Labeled Transition System	128
MARTE	UML Profile for Modeling and Analysis of Real-Time and Embedded Systems	15
MC	Percentage of Misclassified	40
MOP	Multi-objective Optimization Problem	18
MRC	Measured Request Count	56
NED	Normalized Euclidean Distance	40
OCL	Object Constraint Language	69

PCM	Palladio Component Model	15
PE	Prediction Error	56
QN	Queuing Network	14
QPN	Queuing Petri Net	14
RDSEFF	Resource Demanding Service Effect Specification	16
RQ	Research Question	4
SCM	Supply Chain Management	38
SEFF	Service Effect Specification	138
SLA	Service-level Agreement	9
SOA	Service-oriented Architecture	24
SPE	Software Performance Engineering	25
SRC	Simulated Request Count	56
SSE	Sum of Squared Error	138
SUT	System Under Test	3
T	Transaction Type	40
TM	Transaction Mix	40
UML	Unified Modeling Language	46
UML-SPT	UML Profile for Schedulability, Performance and Time	15
VALUETOOLS	International Conference on Performance Evaluation Methodologies and Tools	23

Part A

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Nowadays, many session-based application systems, i.e., systems that are used by users in time-bounded sessions of interrelated requests and think times between these requests, are often accessed by a very large amount of users. It is very important that these systems are able to scale to this demand. Studies in the field of software performance show that there is a strong correlation between revenue and the performance of these systems. Especially for web-based e-commerce systems, performance is a crucial success factor, as slow websites decrease the website usage and sales are reduced[1]. As a result, these systems have to meet non-functional performance objectives like given throughput and response times. To validate whether these objectives can be met, load testing is a commonly used testing procedure in addition to functional testing (e.g., unit testing and integration testing). With load testing the performance is evaluated before the software goes into production to detect potential performance bottlenecks and architectural weaknesses.

Several established load generation tools exist that enable the execution of load tests under various workload conditions such as the variation of the number of users. However, they do not support to design load tests (also referred to as workload specification within this work). The design of a load test defines the load that is executed on the system during testing (Jiang/Hassan, 2015). These tests are designed by specifying test cases and input parameters derived from usage scenarios. Furthermore, the proportion by which each test case is executed as well as the workload intensity defined as the number of expected users executing requests has to be specified. The load is selected based on the test objectives such as detecting functional and performance problems under load. In case the workload is not specified in a way that it is able to test if the given performance objective(s) are fulfilled the testing results are useless or the tests must be repeated. However, the execution of multiple load tests can be very time consuming and costly. Testing environments that represent the production system are also often not available or are used by other departments as well. In addition, there is an increasing demand that software systems are developed in faster release cycles (Brunnert et al., 2015). Thus, the time that is available to specify and conduct load tests decreases.

[1]http://www.carbon60.com/milliseconds-are-money-how-much-performance-matters-in-the-cloud/

Especially for new systems or new software features workload modeling is difficult, as there is little or no information about the expected user behavior. Business domain experts of the operational units can support in determining these workloads, as they know the business process very well. But this knowledge is often not documented in a way, that software experts can use this information to specify workloads. In case the system or parts of the system are already implemented the load can be characterized based on recorded system logs. However, as many different application systems, log formats, and load testing tools exist the extraction of workloads based on these recorded system logs is a cumbersome task. Furthermore, deriving workloads that cause load related problems can be very difficult. The number of possible load test design alternatives that must be evaluated to determine performance problems under load is too huge.

1.2 Research Goal and Research Questions

Many approaches can be found in literature that follow one of the following goals when load tests are designed (Jiang/Hassan, 2015):

- Designing the workload in a way that it is representative compared to the real workload in the field (Feitelson, 2002).
- Designing the workload in a way that it causes load related problems. During the design of the tests optimization and reduction techniques can be applied.

The goal of this dissertation is to fill the gap between these goals by proposing an approach that enables the selection of both representative workloads and workloads that can lead to load related problems. In order to achieve this goal, this approach enables to specify performance objectives that should be tested during the load testing process. Based on these objectives suitable load test designs are derived automatically. This approach automatically extracts, selects and generates load test designs for performance evaluations with measurement-based and model-based performance evaluation techniques.

First, with support of measurement-based approaches (Jain, 1991) like common Application Performance Management (APM) tools (van Hoorn/Waller/Hasselbring, 2012), workload specifications are extracted based on recorded system logs. In case the system is already in production these logs can be easily derived. For newly developed features and applications these logs can be derived during functional testing or by conducting friendly user tests. These systems records are then transformed into workload specifications for load testing tools.

Second, these specifications are used for the generation of workload specifications for model-based performance techniques. These techniques, named performance models, represent performance-relevant aspects of these systems and allow predicting performance characteristics of application systems like response times, throughput and resource utilization. With support of performance models the impact of different load test design alternatives on the System Under Test (SUT) can be evaluated before the load is even executed on the real system.

The following Research Questions (RQs) are answered in this dissertation to address the aforementioned challenges and to achieve the research goal:

Research Question 1: How can representative workload specifications for load testing session-based application systems be automatically extracted?

The question addresses how representativeness and probabilistic workload specifications can be derived from system records in an automatic way. Based on these specifications executable load tests should be derived independent of the used system and the used load testing tool. As the extraction and specification of workloads strongly depends on the used workload generation tool the challenge is to specify a tool and system-agnostic workload specification that covers all aspects needed to generate representative and executable load tests. The advantage of this extraction approach is that the effort to generate load tests that represent the real workload as accurately as possible is considerably reduced.

The outcome of this research question is an evaluated prototype for the automatic specification and extraction of probabilistic and representative workload specification.

Research Question 2: How can the extraction of workload specifications for load testing be integrated with the extraction of workload specifications for performance models?

Workload modeling and generation/simulation are essential to systematically and accurately evaluate performance properties of software systems for both load testing and model-based performance evaluation (e.g., prediction). Workload specifications are therefore also tightly integrated in formalisms and approaches for model-based performance evaluation, including analytic and simulative techniques. Only few approaches for sharing workload specifications between the model-based and the measurement-based worlds exist. This research question examines how the workload specification from *Research Question 1* can be used for model-based performance prediction approaches as well. Furthermore, we will examine if the used performance models are capable to model representative and complex workloads.

This integration comes with several advantages. First, with the use of performance models, workload design options can be evaluated without executing the load test on the real system. Second, to validate the prediction accuracy of performance models with real measurements it is important that both workload specifications show the same workload characteristics.

The outcome of this research question is an evaluated prototype for the generation of workload specifications for performance prediction approaches that is integrated with the prototype from *Research Question 1.*

Research Question 3: How can performance models be used to select load test designs fulfilling given performance objectives applying multi-objective optimization techniques?

Load tests for session-based application systems are designed by selecting and varying usage scenarios and workload intensities. In Research Question 1 and Research Question 2 the goal is to extract workload specifications in a way that they represent the measured workloads as accurately as possible. Like described before the selection of workloads that lead to load related problems is also a common goal of load testing. Therefore, in this research question we examine how workload specifications can be derived with support of performance models fulfilling given performance objectives. The resulting workloads are not necessarily representative to the measured workload. Objectives can be load related such as which workloads stress the CPU of the system to a predefined level. Moreover, also performance objectives should be evaluated that are not directly load related. As an example, the number of resulting test cases can be relevant to reduce load testing efforts and structural metrics like test coverage should be considered as well (Woodside/Franks/Petriu, 2007). With the use of multi-objective optimization techniques solved by a Evolutionary Algorithm (EA) these workload design candidates are derived. We use the extracted performance models from *Research Question 2* and encode different load test designs into these models. These candidates are simulated and assessed based on the given objectives.

As the design space is very huge the performance engineer cannot assess all possible combinations. Thus, the performance of the systems is difficult to assess without using model-based prediction results as otherwise a lot of different load test design alternatives must be executed on a test system.

The outcome of this research question is an evaluated prototype for the selection of load test designs using performance models based on given performance objectives.

1.3 Thesis Structure

The structure of this dissertation is shown in Figure 1.1. The thesis is structured in three parts (Part A, Part B, Part C).

Part A describes the introduction, the conceptual background and the used research methodology. Within the introduction (chapter 1) the problem statement and motivation of this dissertation and the derived research questions are explained. In the chapter conceptual background (chapter 2) preliminaries about load testing, model-based performance predictions and evolutionary optimization techniques are given. The research methodology (chapter 3) contains sections that explain the research design, the research methods and the embedded publications.

Part B contains the six embedded publications P1 to P6 (chapter 4-9). These publications resulted from research done by the author as part of this dissertation. A short summary of the content of the publications can be found in Section 3.3.

Part C concludes this dissertation with a discussion (chapter 10). First, the findings of the publications are summarized. Afterward, limitations and the contribution to research and practice are given. Finally, future research opportunities are explained.

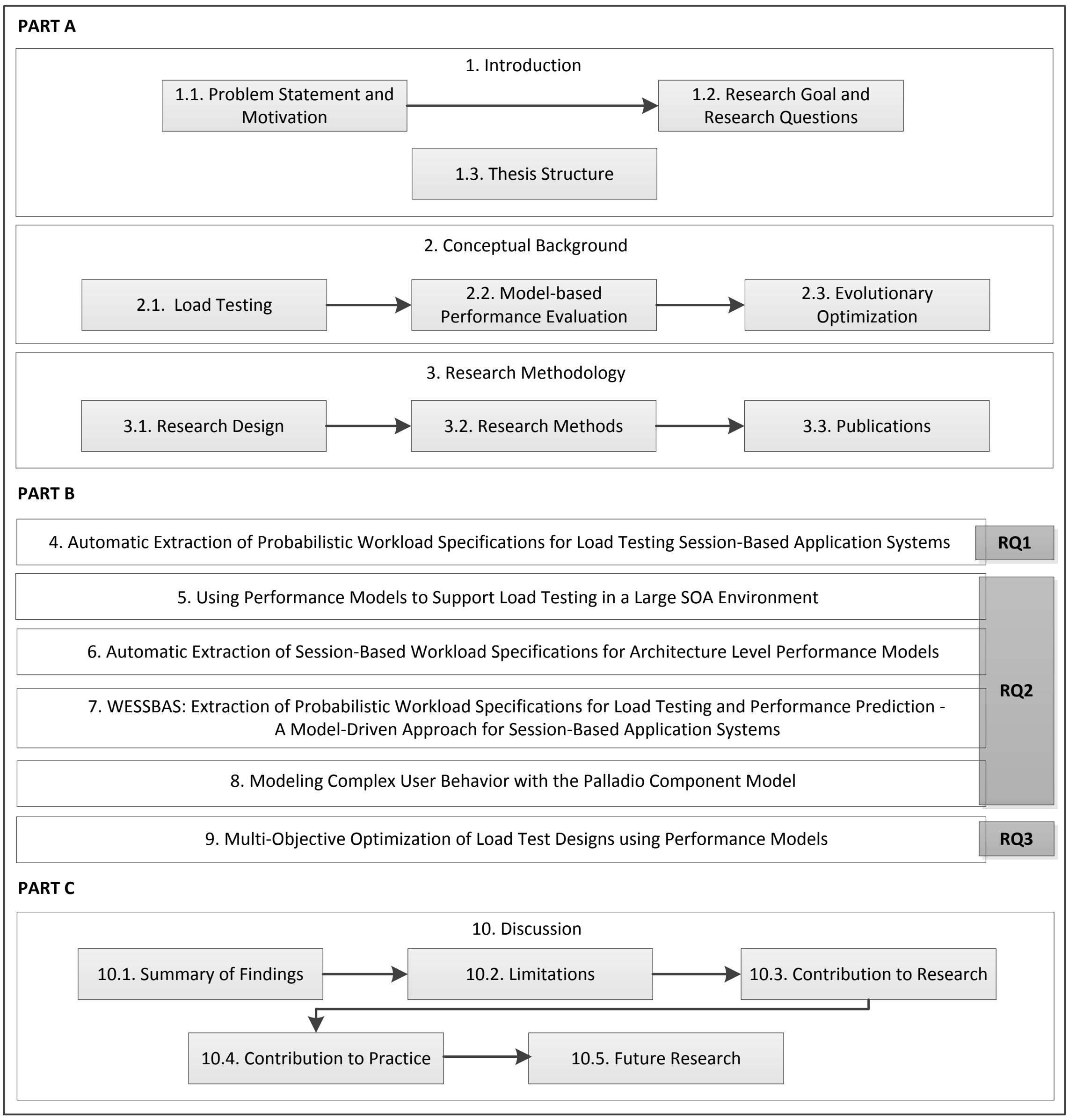

PART A
1. Introduction
1.1. Problem Statement and Motivation
1.2. Research Goal and Research Questions
1.3. Thesis Structure
2. Conceptual Background
2.1. Load Testing
2.2. Model-based Performance Evaluation
2.3. Evolutionary Optimization
3. Research Methodology
3.1. Research Design
3.2. Research Methods
3.3. Publications
PART B
4. Automatic Extraction of Probabilistic Workload Specifications for Load Testing Session-Based Application Systems
RQ1
5. Using Performance Models to Support Load Testing in a Large SOA Environment
6. Automatic Extraction of Session-Based Workload Specifications for Architecture Level Performance Models
7. WESSBAS: Extraction of Probabilistic Workload Specifications for Load Testing and Performance Prediction - A Model-Driven Approach for Session-Based Application Systems
8. Modeling Complex User Behavior with the Palladio Component Model
RQ2
9. Multi-Objective Optimization of Load Test Designs using Performance Models
RQ3
PART C
10. Discussion
10.1. Summary of Findings
10.2. Limitations
10.3. Contribution to Research
10.4. Contribution to Practice
10.5. Future Research

Figure 1.1: *Structure of this dissertation*

Chapter 2

Conceptual Background

This work is based on research in the field of load testing, model-based performance evaluation, and evolutionary optimization approaches. In the following the basic principles and the relations between these are presented that are required to understand the concepts of this dissertation.

2.1 Load Testing

Various definition and interpretations of test types exists in literature. These types have similarities but also their own focuses. We summarize the most important test types in the following based on Meier et al. (2007) and Jiang/Hassan (2015):

Performance tests measure performance related aspects (throughput, response times, resource utilization levels) of a software system. These performance characteristics can be tested with or without load. For example, the test of the performance of a single algorithm is not load testing.

Load tests verify application behavior under normal and/or peak load conditions. This type of test detects problems that only occur during load. The load can have the goal to test the performance, scalability, and/or stability of the SUT as it occurs in the field (cf. operational load (Avritzer/Weyuker, 1995)). Moreover, the load can have the goal to stress the system. This type of test should reveal bottlenecks in the system and test if users of the system will be satisfied with the performance characteristics. Examples are concurrency issues, load balancer configurations, and race conditions.

Stress tests determine or validate the behavior of an application when it is pushed beyond normal or peak load conditions (Zhang/Elbaum/Dwyer, 2011). With this type of test the boundaries of the system related to performance can be detected. It can be determined how robust the system is in case unexpected load conditions occur.

Capacity tests determine the limits of the system for different hardware and software configurations. The goal is to estimate the required hardware and software configuration that is able to fulfill the performance requirements under the expected load conditions.

Failover tests analyze the behavior of the system when different system components under load fail to detect the reaction of the system. For example when one of two data centers has a blackout.

Long-term load tests determine the performance in a long and continuous load test. Some types of performance problems only occur when the system is running for a long time. Examples of these problems are that caches or databases fill up or that memory leaks occur.

During this dissertation all types of tests are relevant when concurrent users (this includes other systems as well) generate requests to the SUT. Primarily, this includes load testing but also stress testing and performance testing under load. The type of chosen test is based on the testing objectives. For example, in case the goal is to derive a load test design that is representative to the real system load testing using operational load is applied. In case load related problems should be identified stress testing is a commonly used method.

We use the following definition of load testing within this dissertation (Jiang/Hassan, 2015):

„Load testing is the process of assessing the behavior of a system under load in order to detect load-related problems“.

The load is defined as the rate at which different service requests are submitted to the SUT. It is typically executed on a running system in a separate test environment. In the best case the setup of the test environment is identical to the setup of the production environment. According to Jiang/Hassan (2015) load testing consists of three main process steps: load test design, load test execution and load test analysis (Figure 2.1). Based on load testing objectives a suitable load test design is derived. In the load test design process step workload specifications are designed that define the key characteristics of user interactions with the system. This testing load is the input for the load test execution step. After the load test execution, recorded system logs representing the behavior of the system are available. In the load test analysis step this information is used in order to analyze the behavior of the system under load and to identify potential performance problems and the root causes of these problems.

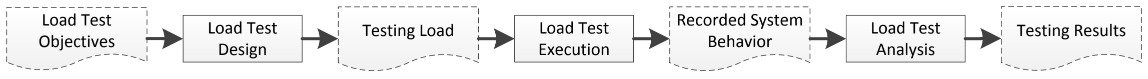


Figure 2.1: *Load testing process based on Jiang/Hassan* (2015)

The typical setup of a load test is depicted in Figure 2.2. A load test design is the input to the load test controller. The controller possibly distributes the load to several workload agents used during the load test to send requests to the system under test. During load testing several agents are often used as the execution of the load might require a lot of computing resources. To prevent that the agents are not able to generate the specified load, several agents are required. During load testing the SUT and the agents collect

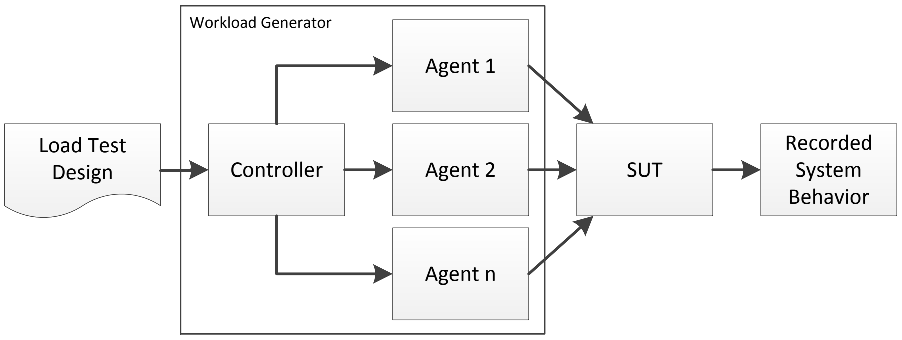


Figure 2.2: *Load testing setup*

relevant metrics about the behavior of the system. Load testing tools are often only able to collect metrics that can be derived from a black box view of the SUT such as response times and throughput. Therefore, also the additional usage of APM tools (e.g., Dynatrace (2016) and Kieker van Hoorn/Waller/Hasselbring (2012)) is recommended to derive insights of the system. The measured data is collected and used to generate load testing reports.

2.1.1 Load Test Objectives

In the load test design phase the load that will be executed on the SUT is designed based on given load testing objectives. The testing objectives are defined based on change, potential risks, and opportunities for improvement of the SUT (Meier et al., 2007). These objectives are the starting point of any performance validation and verification effort. These objectives should be refined periodically as enterprise systems evolves over time. Meier et al. (2007) defines performance objectives as:

„Performance objectives are usually specified in terms of response times, throughput (transactions per second), and resource-utilization levels and typically focus on metrics that can be directly related to user satisfaction.“

Examples of common objectives include but are not limited to:

- Test if the SUT is able to fulfill contracts, regulations, and Service-level Agreements (SLAs) with the expected load (Meier et al., 2007)
- Detect performance problems under load (bottlenecks) (Jiang/Hassan, 2015)
- Evaluate performance characteristics of different configuration settings (Meier et al., 2007)
- Deliver sizing estimates of hardware architecture that is capable to fulfill requirements for the predicted workload (Grinshpan, 2012)

In the following the three main load testing process steps are described in more detail.

2.1.2 Load Test Design

Basically, two different approaches exit to design realistic load (Jiang/Hassan, 2015): aggregated workloads and scenarios. Aggregated workloads are specified with a workload intensity and a behavior mix. The workload intensity defines the rate of incoming requests or the number of concurrent users that execute requests on the system. The behavior mix defines the percentage each request type is executed. For example, an application has two request types and the expected number of users accessing the application is 100. The first request type is executed in 30% of case and the second type in 70% of cases.

The scenario-based approach extends the concept of the aggregate workload. It generates requests for each user in a sequence of interrelated requests. The sequence of requests can be defined in a static way. In this case, always the same sequence is executed for each scenario. Scenarios can also be defined with a workload model using graph-based notations (e.g. Markov chains) with probabilities and think times between the subsequent requests. Then for each execution the sequence of requests is different for each user. This type of workload specification is able to simulate the behavior of real users. In the scenario-based approach the number of concurrent users is named the workload intensity as well. The behavior mix is the ratio for which each scenario is executed. In this case a scenario is also named behavior model in the remainder of this thesis (van Hoorn/Rohr/Hasselbring, 2008).

Scenarios can be derived in different ways. They can be derived from the production (operational profile load) in order to define a realistic workload. This process of extracting and representing the operational profile load as realistic as possible compared to the field workload is also called Workload Characterization. Scenarios can also be estimated based on similar applications when the application is still under development or by interviewing domain experts. Another approach is to derive workloads that stress the system or specific parts of the system to identify performance problems under load. In this case it is not the goal to execute a load that is representative to the real workload.

In-depth information about the workload specification formalism used within this thesis can be found in Section 7.3.1.

2.1.3 Load Test Execution

In the load test execution phase several different approaches exist. The most common approaches are *(i.)* live-users to manually generate load and *(ii.)* using load drivers to automatically generate load (Jiang/Hassan, 2015).

Load tests conducted with live-users have the goal to test realistic user behavior (Kim/Kim/Shin, 2013). Conducting tests with real users has the advantage that real user feedback on the acceptable performance can be directly collected. Furthermore, information about the functional correctness which is otherwise difficult to access can be provided. The reason why this type of load test execution is not very popular is that is very hard to scale as many users are required to generate enough load. Additionally, the tests cannot be reproduced or repeated exactly as they occurred.

The second very popular load test execution approach is the usage of load drivers to automatically generate the required load. Using load drivers the load can easily be scaled to many million requests and it is also easy to reproduce the load. Due to these advantages, we solely use load drivers for load test execution in this dissertation. In contrast to the live-user based testing load drivers setup and configuration effort is required. Existing load testing tools allow easy record and replay functionality by recording sequences of actions and replay them (Rodrigues et al., 2014). The load testing tools also evolve over time and tools are introduced enabling model-based testing (van Hoorn/Rohr/Hasselbring, 2008).

2.1.4 Load Test Analysis

The system behavior data (e.g., execution logs and performance counters) recorded during the test execution phase needs to be analyzed to determine if there are any functional or non-functional load-related problems. Besides key mathematical principles (e.g., median, percentiles, and mean values) to calculate and interpret performance data (Meier et al., 2007) the following methods can be applied:

Verifying Against Threshold Values

Often load test results are compared against predefined threshold values. A very popular way is to compare the results against agreed SLAs between service provider and consumer. During the load tests it must be evaluated if these contracts can be satisfied. An example of such an agreement is that x% of the requests are below a certain response time threshold. We primarily use this type of analysis, as we can easily compare given performance objectives, such as given response times with prediction values derived from performance model simulations.

Detecting Known Problems Using Patterns

Another load test analysis method is to compare the load test results against known problems. Especially problems that cannot be identified using simple threshold values are of interest. Examples of know problems are pattern in the memory utilizations, (e.g., memory leak detection), patterns in the CPU utilizations, (e.g., deadlocks), or patterns in the logs like special error keywords (e.g., failure).

Automated Detection of Anomalous Behavior

As not all patterns can be specified and detected automatically the automated detection of anomalous behavior is an important research topic in the field of load testing. The goal is to systematically analyze the system behavior to uncover anomalous behavior (Jiang/ Hassan, 2015; Foo et al., 2010). These methods have the goal to differentiate between normal and anomalous behavior.

2.2 Model-based Performance Evaluation

Model-based performance evaluation approaches in combination with analytical solvers or simulation engines enable to predict performance characteristics like response times, throughput and resource utilization. Using these models can reduce the load testing effort as it is not required to build test environments similar to the production environment. These approaches model performance relevant characteristics of the SUT. These characteristics include:

1. Software architecture
2. Hardware infrastructure
3. Workload specification

The software architecture and the hardware infrastructure represent the system-specific details of the SUT. The software architecture describes the components, the methods of the components and the relationships between these. Furthermore, the resource demands of the components need to be represented. Resource demands are defined as the demand from a hardware resource (e.g. CPU) for each unit of work (e.g. a request). For example the number of milliseconds the CPU has to calculate for a specific user request. The hardware environment specifies the most important parts of the used hardware of the SUT like the number of servers and the number of cores per server. The workload specification defines the load that is tested on the modeled system.

An overview of model-based performance evaluation can be found in Figure 2.3. In this example the structure of an architecture-level performance model is illustrated. An application server (hardware infrastructure) is accessed by business administrators and by customers (workload specification). The components (e.g., Parser or Administration) that are deployed on the server and the behavior of interface operations of these components (software architecture) are modeled as well. This model can be simulated directly to derive prediction metrics like response times or resource utilizations. There is also the possibility to automatically translate it into analytical models and then be processed by respective solvers to derive prediction metrics.

Performance models are able to support various use cases throughout the complete software life cycle (Brosig/Huber/Kounev, 2011; Krcmar, 2015). In the system development

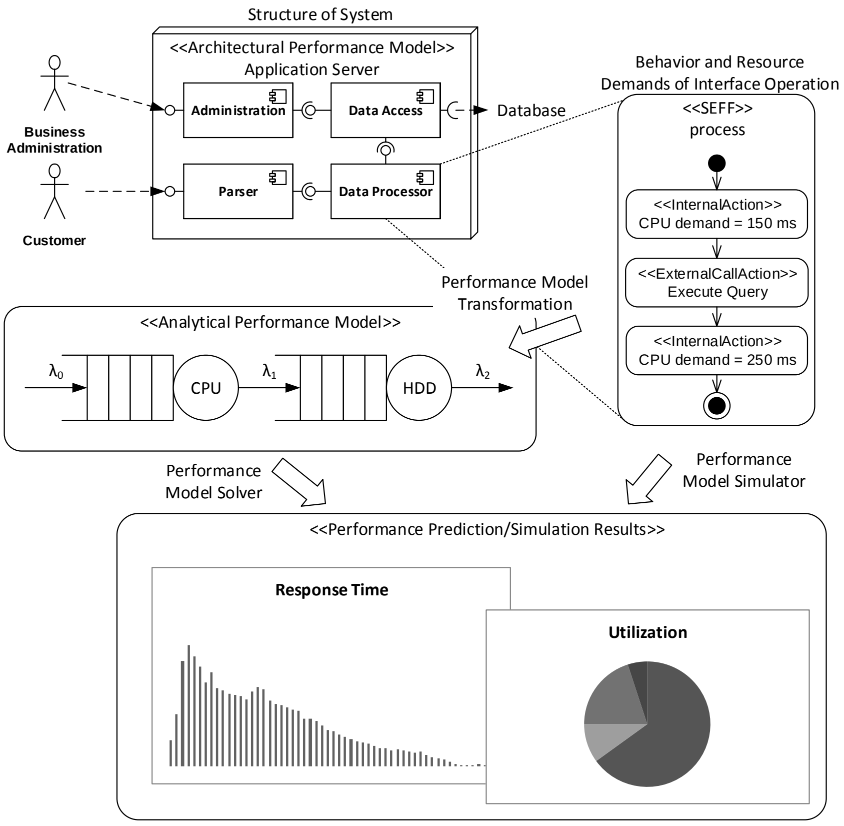


Figure 2.3: *Model-based performance evaluation (Brunnert* et al. (2015)*)*

phase these models are able to predict the performance of early-stage prototypes. Furthermore, continuous change detection between the prediction of two builds in continuous integration process (Brunnert/Krcmar, 2014) is enabled. During system deployment the required software and hardware capacity of the system can be determined. Finally, during operations these performance models can be used for continuous performance-aware resource management (Calinescu et al., 2011).

Basically, there are two different types of performance models: architecture-level performance models and analytical models. Architecture-level performance models represent key performance characteristics of a system and enable to model the system architecture, execution environment, and workload specification separately from each other (Brosig/Huber/Kounev, 2011). In contrast analytical models do not provide this separation and models the system in only one single model.

In order to determine the resource demands of a system two different approaches can be applied: direct measurements or resource demand estimation. Direct measurements can be conducted using APM tools and profiling tools. These tools use fine-grained code instrumentation or statistical sampling. Furthermore, facilities of operating systems provide the possibility to track the consumed resource consumptions. On the other side approaches for resource demand estimation are based on a combination of statistics about aggregate resource usage (e.g., CPU utilization) and coarse-grained application statistics (e.g., end-to-end application response times or throughput) (Brunnert et al., 2015). Using these approaches fine grained instrumentation of the application is not required. Common examples for these approaches are response time approximation (Brosig/ Kounev/Krogmann, 2009), Kalman Filter (Wang et al., 2012) or optimization techniques (Menascé, 2008). A Library for Resource Demand Estimation (LibReDE) (Spinner et al., 2014) offers the implementation of several estimation approaches.

2.2.1 Analytical Performance Models

Classical performance models are analytical models. The main focus of these models is the accurate modeling of system resources. Examples for these types of models are Queuing Networks (QNs), Queuing Petri Nets (QPNs), or Layered Queuing Networks (LQNs) (Balsamo et al., 2004; Ardagna et al., 2014).

The basis of analytical performance models is queuing theory (Jain, 1991). As in computer systems many jobs share the same system resources (e.g., CPU, Disks) only one job can use the resource at the same time. Thus, the other jobs must wait until they are allowed to use the resource. These jobs are waiting in so called queues. The time each job acquires the resource and the time the jobs are waiting in the queue is the response time. With support of the queuing theory the time that jobs spend in (possibly multiple) queues is determined. In order to analyze a queuing system, the following characteristics of the system should be specified (Jain, 1991):

1. **Arrival Rate** Is defined as the time distribution between the arrival of new requests that arrive at the system.
2. **Service Time Distribution** Is defined as the distribution each request acquires a resource.
3. **Number of Servers** Defines the number of identical service stations such as CPU, disks, etc.
4. **System Capacity** Defines the capacity of the queues. This contains both waiting requests and requests that are served.
5. **Population Size** Defines the maximum number of the population potentially enters the system.
6. **Service Discipline** Defines the order in which the requests are served. The most common discipline in computer systems research are First Come, First Served (FCFS), processor sharing or round robin.

These six characteristics of a queuing system can be defined using the Kendall-Notation. An example of this notation is a G/G/1/1000/1000/FCFS system. In this case the arrival rate is generally distributed meaning the queuing results are valid for all distributions of inter-arrival times. The service time distribution is generally distributed as well. The number of servers is 1, the system capacity and the population size is 1000, and the service discipline is FCFS. In case the system capacity, the population size and the service discipline is not specified the queues are defined as having infinite buffer capacity, infinite population size, and an FCFS service discipline. In this case only the first three parameters must be specified as G/G/1.

In case more than one queue is in the system and requests can be processed by multiple queues a model is called QN. An example of a QN as analytical performance model can be found in Figure 2.3. In this example two service stations (CPU and Hard Disk Drive (HDD)) each having a queue is depicted. λ_0 and λ_1 define the arrival rates whereby these values are the same in this example as the requests cannot leave the system after being processed by the CPU. These types of models can be solved by corresponding solvers in order to derive prediction results.

An advancement of QN are LQN. The QN can only use one resource at a time. This makes the modeling and simulation of common use case very difficult or even not feasible. For example, when a server is calling another server and waits for the response (Franks et al., 2009). Thus, in contrast to QN the LQN allow for simultaneous use of resources. These models have a hierarchical structure and are solved with analytical solvers based on Mean-Value Analysis (MVA) for LQNs (Rolia/Sevcik, 1995).

2.2.2 Architecture-Level Performance Models

The goal of architecture-level performance models is to create designer friendly modeling notations. Examples of architecture-level performance models are Palladio Component Model (PCM) (Becker/Koziolek/Reussner, 2009), the Descartes Modeling Language (DML) (Kounev/Brosig/Huber, 2014), the UML Profile for Schedulability, Performance and Time (UML-SPT) (Object Management Group, Inc., 2005), and the UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) (Object Management Group, Inc., 2013). PCM and DML are performance models focusing on the prediction for component-based software systems like modern software systems. According to Koziolek (2010) the most important factors that influence the performance of component-based software systems are:

1. **Component Implementation** The implementation of the functionality specified by an interface.
2. **Required Services** The execution time of a component is dependent on the execution time of the required services.
3. **Deployment Platform** Different software (e.g., component container, virtual machine, operating system, etc.) and hardware layer (e.g., processor, storage device, network, etc.) provide different response times.
4. **Resource Contention** As a software component is not executed as a single process, waiting times for accessing limited resources occur. These waiting times are additional to the execution times of a software component.
5. **Usage Profile** The invocation of the component can occur with different input parameters and varying invocation frequency.

In order to derive prediction results these models must be simulated or solved. To solve these models they must be first transfered to analytical models and then be processed by respective solvers. For each performance model suitable solvers or simulation engines are available. For example, for PCM transformations to analytical performance models such as LQNs (Koziolek/Reussner, 2008) or stochastic regular expressions and different simulation engines (Becker/Koziolek/Reussner, 2009; Merkle/Henss, 2011) exist.

The advantages of solvers are that the prediction results are very precise and are derived very fast (Becker/Koziolek/Reussner, 2009). However, they do not support G/G/n queues and support only single user scenarios. On the other side simulation engines allow to simulate G/G/n queues and also multiple user scenarios. But the results are only approximated and are derived slower and require more computing capacity.

In this dissertation we focus on architecture-level performance models in combination with simulation engines. These types of models enable us to model and adjust the workload specification separately from the system specification. Furthermore, as the simulation of multiple user scenarios is required only simulation techniques are used to derive prediction results.

2.2.3 Palladio Component Model

In this dissertation the architecture-level performance model PCM is used and is therefore explained in more detail. This performance model is a domain-specific modeling language and is composed of five different model types (see Figure 2.4): repository model, system model, resource environment model, allocation model and usage model. The first four model types define the system-specific parts of the system and the usage model defines the workload specification. The *repository model* models the software components, component operations, and the relations between them using provided and required interfaces. The component operations are modeled in so called Resource Demanding Service Effect Specifications (RDSEFFs). These specifications represent probabilistic abstractions of the control flow. They also enable to specify the resource demand per activity and the dependencies of transition probabilities and resource demands on the formal parameters of the service. In the *system model* the modeled components are assembled to the system which also define the system boundaries. The resources of the system (e.g., servers, switches) are modeled in the *resource environment model*. For each resource container the available resources are modeled and performance specifications (e.g., CPU speed) are provided. In the *allocation model* the system components are allocated to the resource container. Finally, the *usage model* defines the workload of the system.

We use the simulation approach SimuCom (Becker/Koziolek/Reussner, 2009) within this work to derive prediction results. First, a PCM instance is transformed into Java source code by a model-2-code transformation. Afterward, the resulting Java code is executed in an OSGi runtime to start a simulation. SimuCom is a process-oriented simulation and starts a new thread for each simulated user. As a result it needs more resources than the event-driven simulation EventSim (Merkle/Henss, 2011). However, as SimuCom supports all features of PCM we use it to derive to required prediction results.

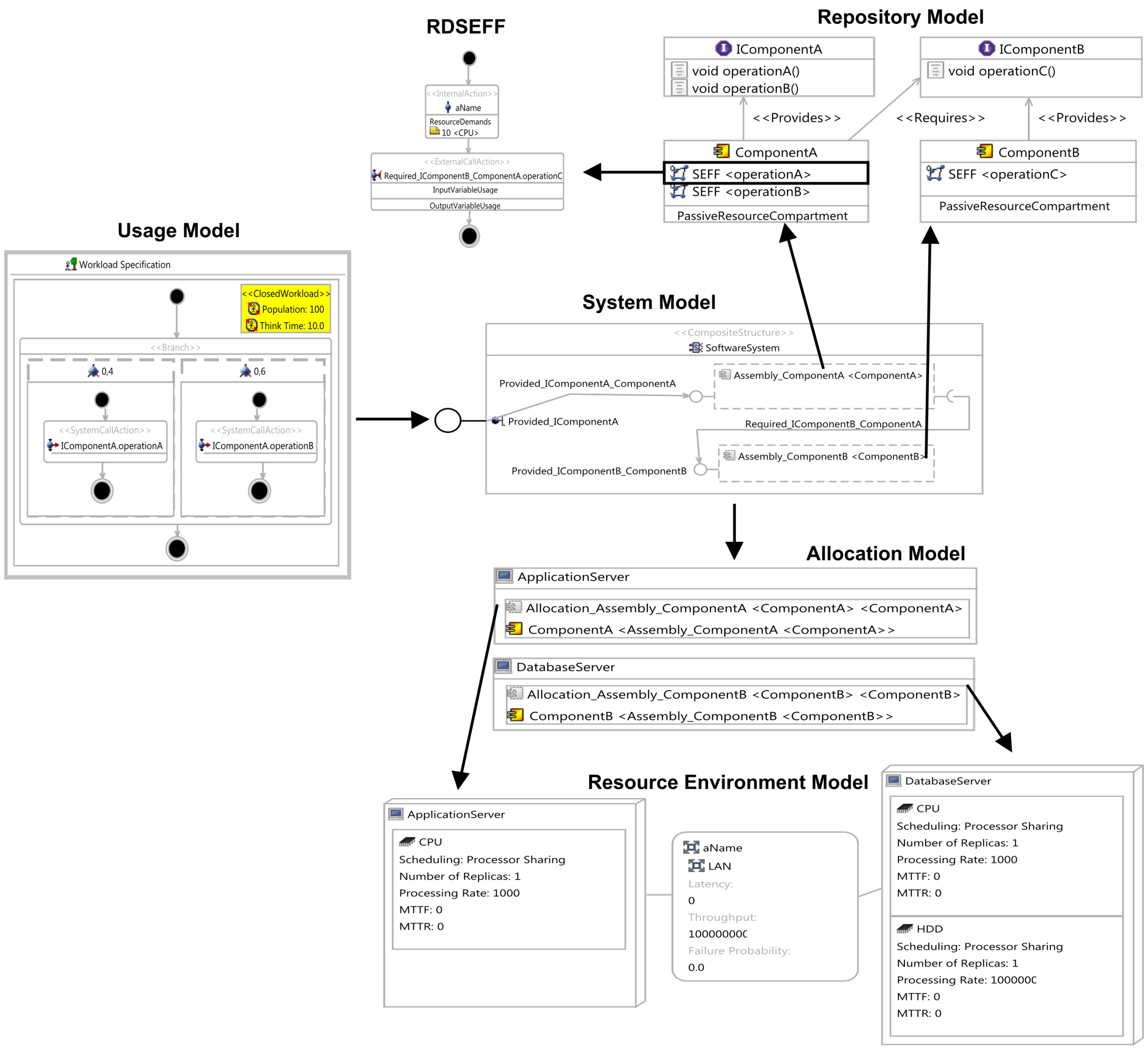


Figure 2.4: *PCM models and their relationships*

2.2.4 Automatic Extraction of Performance Models

As manual modeling of performance models often outweighs the advantage of using these types of models automatic extraction is required. Due to this, several approaches were proposed that extract performance models from running systems in an automatic way.

Approaches that extract analytical performance models (LQN) are proposed by Mizan/ Franks (2012) and Israr et al. (2005). This work demonstrates the generation of LQN performance models through the analysis of trace information from a live system. These models can then be solved to locate bottlenecks in both the hardware and software.

There are several approaches that extract PCM models. In Brunnert/Vögele/Krcmar (2013) servlet filter and Enterprise Java Bean (EJB) interceptors are used to derive measures that are needed to identify the components, the relationships between these components, the control flow of requests and the resource demands within a Java EE system. We use this performance model generator during this thesis to generate the system-specific parts of the performance model. In Willnecker et al. (2015) this approach is extended in a way that it also enabled to generate performance models based on Dynatrace recordings (Dynatrace, 2016). Brosig/Huber/Kounev (2011) proposes a semi-automatic approach based on monitoring data of a WebLogic Application Server collected at run-time. However, using this approach the identification of an appropriate granularity level for modeling the components must be specified manually by the user. The calculation of the resource demand must also be conducted in a manual way.

Another approach that extracts PCM models from measurements is proposed by Walter[2]. Using this approach log files measured by APM tool (e.g., Kieker (van Hoorn/Waller/Hasselbring, 2012)) are the basis to derive all required aspects of PCM.

Furthermore, Brebner (2016) developed an approach that automatically builds and parameterize performance models from APM data. In their paper the authors present an industry experience report about the usage of performance models in different industry projects.

2.3 Evolutionary Optimization

Having extracted a performance model with a representative workload specification there is the opportunity to question these models specific what-if questions. A what-if question is a specific workload, design or architectural decision like: „What happens if the number of users accessing the system increase to twice the expected amount?“. A lot of approaches exist having the goal to automatically improve a software architecture for performance (Aleti et al., 2013). For this purpose, single or multiple optimization functions are specified and based on predicted metrics (e.g., mean response times) from different performance model instances these functions are optimized. Examples for these functions are response times, costs, reliability, and resource utilization. In our work we do not focus on optimizing the software architecture. We have the goal the derive load test designs that are able to fulfill a given set of predefined performance objectives without changing the system-specific parts.

2.3.1 Multi-objective Optimization Problems

A single objective optimization has the goal to maximize or minimize a objective function $f(p)$ (Brunnert et al., 2015). The predicted metric of interest p is derived from the solving or simulation of a performance model. As for many what-if questions multiple performance objectives are of interest, Multi-objective Optimization Problems (MOPs) are relevant. A MOP is defined as (Coello/Lamont/Van Veldhuisen, 2007):

„A vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other. Hence,

[2]http://se.informatik.uni-wuerzburg.de/tools/pmx/

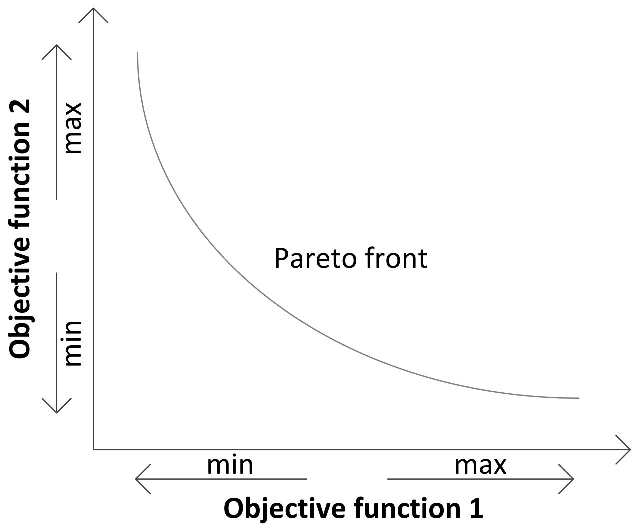


Figure 2.5: *Exemplary multi-objective optimization problem with two objective functions that should be minimized*

the term "optimize" means finding such a solution which would give the values of all the objective functions acceptable to the decision maker."

Thus, in a MOP multiple optimization functions are involved. The goal is to optimize k objective functions simultaneously. These functions can be minimized, maximized or a combination of maximization and minimization is applied. A general MOP is defined as minimizing (or maximizing) (Coello/Lamont/Van Veldhuisen, 2007)

$$F(x) = [f_1(c), f_2(c), ..., f_k(c)] \tag{2.1}$$

where k is the number of objectives and x is a n-dimensional decision variable vector $x = (x_1, ..., x_n)$ from some universe Ω. Each decision variable of the vector can be mapped to a design decision of the performance model such as the number of used servers or the population size (workload intensity).

The nature of these MOPs is to have not only one solution but a set of solutions. Pareto optimality theory (Ehrgott, 2006) is used to derive this set. A solution is defined as Pareto-optimal, if it is not possible to find another solution that is better or at least equal with respect to all objective functions. In Figure 2.5 a Pareto-front with two objective functions that should both be minimized is illustrated. For example, objective function one is the cost of a hardware infrastructure and objective function two the average response time of this system. When the costs of the infrastructure should be reduced than the response times will increase (and vice versa). Due to that reason, all optimal solution are on the Pareto-front or also trade-off surface.

2.3.2 Basic Concepts of Evolutionary Optimization

The aforementioned MOP can be solved with EA as these are suitable to handle multiple solutions simultaneously. An EA manipulates with a set of operators a population of encoded solutions (individuals) in order to evaluate predefined fitness functions.

The data structure used in EAs are based on biological concepts. An individual candidate that is evaluated is encoded to some problem and corresponds to a biological *genotype*. In our case a vector of performance design options is called the genotype. These genotypes are then decoded into a *phenotype*. In our case into a concrete instance of a performance model representing the genotype. A genotype consists of one to many *chromosomes*. These chromosomes are again composed of *genes* defining a specific value from a predefined alphabet. A genotype contains one or more chromosome instances which are called the *population*. The number of possible chromosomes is defined as the *design space* of the MOP.

Using this data structure the EA basically iterates through three phases which is called a *generation* or EA iteration: reproduction, competition and selection. The members of the current generation are called *parent* individuals and the members of the next generation the *children* individuals. Using this approach constantly new candidates are created, evaluated and selected until a predefined stop criterion occurs.

Reproduction Phase In order to find solutions that fulfill the objectives in an optimal way Evolutionary Operators (EVOPs) are used. Two EVOPs are used with EAs and are recombination and mutation:

1. **Recombination** The most popular form of recombination is crossover. Crossover mixes the genes of the selected candidates and creates new candidates. With crossover new solutions are derived within the design space to prevent local optima. Using crossover, parts of the genes of one parent are mixed with parts of another parent to create new children for the next generation.
2. **Mutation** The values of variables within parent are randomly changed and new children are created. Mutation is often executed after the recombination. Thus, the design space of possible solution is scanned continuously.

In the **Competition Phase** each candidate is evaluated. In our case that means that each performance model instance is simulated and the prediction metrics are derived.

During the **Selection Phase** the genes with higher fitness values are selected to contribute to one or more children in the succeeding generation. For example, candidates which are considered to be Pareto-optimal are selected and the others are removed.

Chapter 3

Research Methodology

3.1 Research Design

In order to answer the research questions this dissertation follows the design-science paradigm (Simon, 1996). Design-science in the field of information systems research is a problem solving paradigm with the goal to create new and innovative artifacts (Denning, 1997). The goal of this type of research is to create new *„things that serve human purposes“* (March/Smith, 1995).

Based on the work from Hevner et al. (2004), Peffers et al. (2007) proposes the *design science process* which consists of six steps:

1. **Identify problem and motivate** In this phase the problem is identified and the benefits of a solution presented. Furthermore, existing research and solutions for this problem must be analyzed and presented.

2. **Define objectives of the solution** Based on the understanding of the problem the goal(s) of the research is/are derived.

3. **Design and development** In this phase artifact(s) are designed and developed in order to fulfill the goals. This is accomplished with support of existing theories and knowledge.

4. **Demonstration** The efficacy of the artifact to solve the problem must be demonstrated. The demonstration could include experimentation, simulation, case studies, or other appropriate activities.

5. **Evaluation** The goal of this phase is to evaluate (observe and measure) the results of the artifacts. For this purpose Hevner et al. (2004) proposes several evaluation methods. The results of the evaluations decide if the artifacts are able to solve the problems or if the design, development and the evaluation has to be repeated.

6. **Communication** In case the usefulness is proven, the artifact and the evaluation results should be presented to researchers and practitioners.

This thesis is based on this process. In Section 1.1 the problem and motivation of the research is explained. Section 1.2 describes the objectives of this thesis and the research questions to achieve them. The phases design and development, demonstration and evaluation are conducted and described in the publications (Part B of this dissertation) which also fulfill the requirements to communicate the findings to the relevant audience.

3.2 Research Methods

The used methods controlled experiments and scenarios to evaluate the developed solutions are derived from Hevner et al. (2004) and are enhanced with the research methods literature review and prototyping. The used methods to evaluate the findings and artifacts are explained in the following:

Literature Review In the first step of conducting research the design science process (Peffers et al., 2007) demands a literature review (Webster/Watson, 2002). The literature review identifies publications that are relevant to understand the problem and the existing solutions. Based on this knowledge research questions and a research methodology can be designed. Furthermore, the danger that a solution to the problem already exists is reduced.

During the work on each publication a literature review is conducted to find related work and already available solutions. Based on suitable keywords scholarly databases are searched. The main approaches that are used during the search are forward and backward searches (Levy/Ellis, 2006). First, the references of the publications that are identified by the keyword search are further analyzed for relevant references (backward search). Second, publications are considered that cite the identified publications (forward search). The relevance of the identified publications was rated based on the titles, keywords, and abstracts. The used keywords are *load testing*, *load test design*, *workload specification*, *performance prediction*, *performance models*, *selection of test cases*, *evolutionary algorithms*, *multi-objective optimization* and various combinations and modifications.

The most important online scholarly databases that were considered during this thesis are:

1. Association for Computing Machinery (ACM)[3]
2. IEEE[4]
3. Springer[5]

The main focus of the literature review are publications in the field that are published in the following workshops, conference, and journals:

[3]http://dl.acm.org/
[4]http://ieeexplore.ieee.org/Xplore/home.jsp
[5]http://www.springer.com/de/

1. European Performance Engineering Workshop (EPEW)
2. International Conference on Performance Engineering (ICPE)
3. International Conference on the Quality of Software Architecture (QoSA)
4. International Conference on Autonomic Computing (ICAC)
5. International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)
6. International Journal on Software and Systems Modeling (SoSyM)
7. Journal of Systems and Software (JSS)
8. ACM Transactions on Internet Technology (ACM TOIT)
9. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)
10. IEEE Transactions on Software Engineering
11. Business & Information Systems Engineering (BISE)

Prototyping Prototyping is a popular method to create software artifacts in information systems development and evaluate them for usefulness (Alavi, 1984). The development of software prototypes is a core method during this dissertation. Prototyping is especially suitable when *„there is a need for experimentation and learning before commitment of resources to development of a full-scale system“* (Alavi, 1984). Prototypes can iteratively be evaluated, refined and adjusted in order to reach the research goals.

Controlled Experiments According to Sjoeberg et al. (2005) a controlled experiment in software engineering is defined as a: *„randomized experiment or a quasi-experiment in which individuals or teams (the experimental units) conduct one or more software engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages, or tools (the treatments).“* We use these experiments in order to evaluate in a comparable and reproducible setup. Therefore, we use an industry standard benchmark or publicly available datasets.

Scenarios In order to demonstrate the applicability of the developed artifacts detailed scenarios are constructed Hevner et al. (2004). As we explained we use a controlled experiments setup in a lab environment and not real world applications to guarantee a comparable and reproducible setup. Thus, we defined different scenarios derived from real world problems to discuss the applicability and the plausibility of the results.

3.3 Publications

As explained in the introduction, Part B of this dissertation is composed of six publications of the author (Table 3.1). Further publications that have been coauthored during the research are given in Table 3.2. Both tables include a publication number, the authors, the title and the outlet of each publication. Publications listed in Table 3.1 (**P1**-**P6**) are included in Part B with permission of the corresponding publishers.

In the following a brief summary for each embedded publication is given:

P1 gives answers to the first research question: *How can representative workload specifications for load testing session-based application systems be automatically extracted?*. In **P1** the WESSBAS approach is introduced. In this paper the challenge is addressed that workloads which represent the real workload as accurately as possible are difficult to specify. Thus, an approach for the modeling and automatic extraction of probabilistic workload specifications for load testing session-based application systems is introduced.

Publications **P2** to **P5** gives answers to the second research question: *How can the extraction of workload specifications for load testing be integrated with the extraction of workload specifications for performance models?*. **P2** introduces the concept of using performance models and simulation techniques to support load testing. This paper address the challenges applied to load testing in a large Service-oriented Architecture (SOA) environment. It proposes to use performance models to select usage scenarios and to predict the number of resulting requests for newly developed services. With the number of predicted requests the services provider are able to scale the services before for the load test starts.

The WESSBAS approach introduced in **P1** is extended in **P3** by the capability of transforming WESSBAS-DSL instances into workload specifications of architecture-level performance models. This paper demonstrates that the WESSBAS-DSL can also be used as an intermediate language between workload specifications for load testing and the generation of required inputs for performance evaluation approaches.

Publication **P4** integrates the work of **P1** and **P3**. It contains considerable extensions of the WESSBAS approach and tooling support (modeling language, extraction, generation) which allow to (close to) fully automatically extract and generate executable load tests and model-based performance predictions. Only minor manual refinements are required for the load test scripts. This was not yet possible with the status of the approach as of van Hoorn et al. (2014) and Vögele/van Hoorn/Krcmar (2015).

In **P5** the limitations of modeling complex workloads using the current PCM usage model implementation are addressed by extending this meta-model. Furthermore, we integrated the extended PCM usage model into the WESSBAS approach. The WESSBAS-DSL instances can also be transformed into the extended PCM usage model.

The third research question (*How can performance models be used to select load test designs fulfilling given performance objectives applying multi-objective optimization techniques?*) will be answered in **P6**. This publication demonstrates that performance models in combination with evolutionary algorithms can be used to derive workloads that achieve given performance objectives before load test execution.

No.	Authors	Title	Outlet
P1	van Hoorn, **Vögele**, Schulz, Hasselbring, Krcmar	Automatic Extraction of Probabilistic Workload Specifications for Load Testing Session-Based Application Systems	International Conference on Performance Evaluation Methodologies and Tools 2014
P2	**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
P3	**Vögele**, van Hoorn, Krcmar	Automatic Extraction of Session-Based Workload Specifications for Architecture Level Performance Models	International Workshop on Large-Scale Testing (LT) 2015
P4	**Vögele**, van Hoorn, Schulz, Hasselbring, Krcmar	WESSBAS: Extraction of Probabilistic Workload Specifications for Load Testing and Performance Prediction - A Model-Driven Approach for Session-Based Application Systems	International Journal on Software and Systems Modeling (SoSyM) [6] 2016
P5	**Vögele**, Heinrich, Heilein, van Hoorn, Krcmar	Modeling Complex User Behavior with the Palladio Component Model	Symposium on Software Performance (SOSP) 2015
P6	**Vögele**, Krcmar	Multi-Objective Optimization of Load Test Designs using Performance Models	ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)

Table 3.1: *Publications embedded in this thesis*

Additionally to the embedded publications, the author contributes as coauthor to several further publications related to the topic of this dissertation.

In **P7** an approach is presented that translated performance measurements from running Java EE applications into the architecture level performance models PCM. The approach is evaluated by applying a prototype of the approach to an industry standard benchmark. The approach is also used in **P3**, **P5** and **P6** to generate the system specific parts of the performance model.

In **P8** and **P9** the challenges of applying model-based performance evaluation in industry are examined. The challenges of integrating model-based performance evaluation methods into the software development process of an industry project are outlined in **P8**. Whereby **P9** outlines the challenges of integrating Software Performance Engineering (SPE) and APM activities to continuously evaluate the performance of enterprise applications.

P10 presents a systematic literature review of papers published in the proceedings of the International Conference on Performance Engineering (ICPE) and its predecessors (Danciu et al., 2015). The topics, evaluation methods, and types of systems that are presented at this conference are analyzed over time and are enriched with geographical and organizational information.

[6]Impact factor, 2 years: 0.990, 5 years: 1.497, http://www.sosym.org/, (accessed: 28th May 2017)

The focus of **P11** is to outline existing performance management activities that are common in development (Dev) and operations (Ops) phases. Furthermore, performance management activities that enable a tight integration between both phases as well as open research challenges in this area are outlined.

P12 represents the proceedings of the International Workshop on Large-Scale Testing (LT) 2015 co-organized by the author that addresses the challenge of large-scale testing[7].

Finally, in **P13** a scalable simulation service for the Palladio Component Model (PCM) workbench based on a headless Eclipse instance and a Java EE application server is introduced. With this simulation service multiple simulation runs can be executed in parallel. This simulation service is used in **P6** to simulate multiple workload candidates concurrently.

No.	Authors	Title	Outlet
P7	Brunnert, **Vögele**, Krcmar	Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications	European Workshop on Performance Engineering (EPEW) 2013
P8	Brunnert, Danciu, **Vögele**, Tertilt, Krcmar	Integrating the Palladio-Bench into the Software Development Process of a SOA Project	Symposium on Software Performance (SOSP) 2013
P9	Brunnert, **Vögele**, Danciu, Pfaff, Mayer, Krcmar	Performance Management Work	Business & Information Systems Engineering[8] 2014, Wirtschaftsinformatik (WI) 10 2014
P10	Danciu, Kroß, Brunnert, Willnecker, **Vögele**, Kapadia, Krcmar	Landscaping Performance Research at the ICPE and its Predecessors: A Systematic Literature Review	International Conference on Performance Engineering (ICPE) 2015
P11	Brunnert, van Hoorn, Willnecker, Danciu, Hasselbring, Heger, Herbst, Jamshidi, Jung, von Kistowski, Koziolek, Kroß, Spinner, **Vögele**, Walter, Wert	Performance-oriented DevOps: A Research Agenda	Technical Report, SPEC Research Group – DevOps Performance Working Group 2015
P12	Jiang, **Vögele**	LT 2016: The Fifths International Workshop on Large-Scale Testing	International Workshop on Large-Scale Testing (LT) 2016
P13	Willnecker, **Vögele**, Krcmar	SiaaS: Simulation as a Service	Symposium on Software Performance (SOSP) 2016

Table 3.2: *Further publications during the work on this dissertation*

[7]http://lt2016.eecs.yorku.ca/
[8]Ranked A according to WKWI list (WKWI, 2008)

Part B

Chapter 4

Automatic Extraction of Probabilistic Workload Specifications for Load Testing Session-Based Application Systems

Authors	van Hoorn, André[1] (andre.van.hoorn@acm.org) Vögele, Christian[2] (voegele@fortiss.org) Schulz, Eike[3] (esc@informatik.uni-kiel.de) Hasselbring, Wilhelm[3] (wha@informatik.uni-kiel.de) Krcmar, Helmut[4] (krcmar@in.tum.de) [1]Institute of Software Technology, University of Stuttgart, Stuttgart, Germany [2]fortiss GmbH, Munich, Germany [3]Department of Computer Science, Kiel University, Kiel, Germany [4]Chair for Information Systems, Technical University of Munich (TUM), Garching, Germany
Outlet	Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools 2014 (VALUETOOLS)
Status	Accepted
Individual Contribution	Problem and scope definition, construction of the conceptual approach, prototype development, experiment design, execution and result analysis, paper writing, paper editing

Table 4.1: *Bibliographic details for P1*

Abstract Workload generation is essential to systematically evaluate performance properties of application systems under controlled conditions, e.g., in load tests or benchmarks. The definition of workload specifications that represent the real workload as accurately as possible is one of the biggest challenges in this area. This paper presents our approach for the modeling and automatic extraction of probabilistic workload specifications for load testing session-based application systems. The approach, called Wessbas, comprises *(i.)* a domain-specific language (DSL) enabling layered modeling of workload specifications as well as support for *(ii.)* automatically extracting instances of the DSL from recorded sessions logs and *(iii.)* transforming instances of the DSL to workload specifications of existing load testing tools. During the extraction process, different groups of customers with similar navigational patterns are identified using clustering techniques. We developed corresponding tool support including a transformation to probabilistic test scripts for the Apache JMeter load testing tool. The evaluation of the proposed approach using the industry standard benchmark SPECjEnterprise2010 demonstrates its applicability and the representativeness of the extracted workloads.

4.1 Introduction

For essentially any measurement-based software performance evaluation activity—e.g., load, stress, and regression testing, or benchmarking—it is necessary to expose the system under test (SUT) to synthetic workload (Brunnert et al., 2014; Jain, 1991; Krishnamurthy/Rolia/Majumdar, 2006; Menascé, 2002), i.e., automatically generating requests to system-provided services. Workload generation tools—also called workload drivers—are used to emulate a multitude of concurrent system users based on workload specifications, ranging from manually defined scenarios over recorded traces to analytical models (Barford/Crovella, 1998). This paper focuses on analytical model-based workload generation for session-based systems, i.e., systems that are used by users in time-bounded sessions of interrelated requests and think times between subsequent requests (Menascé et al., 1999).

Approaches have been proposed for specifying and generating workloads for this type of systems (e.g., (Krishnamurthy/Rolia/Majumdar, 2006; Menascé, 2002; Menascé et al., 1999). However, one of the biggest challenges is how to obtain workload specifications that produce workload characteristics similar to a system's production usage profile, e.g., arrival rates of sessions and requests to system-provided services. Further, the extraction and specification of workloads strongly depends on the used workload generation tool. Because of that the workload must be extracted for each tool and specified into a specific structure.

In response to these challenges, this paper presents our WESSBAS[9] approach for specifying and extracting probabilistic workloads for session-based application systems. A domain-specific language (DSL), called WESSBAS-DSL, is introduced which enables the system- and tool-agnostic modeling of these workload specifications. Recorded session logs from the systems are used as basis for the automatic extraction of WESSBAS-DSL instances. Different groups of customers showing similar navigational patterns are identified during the creation of these instances. WESSBAS-DSL instances are transformed to workload specifications for load generation tools. Finally, a transformation to the common load testing tool Apache JMeter, including the Markov4JMeter extension developed in our previous work (van Hoorn/Rohr/Hasselbring, 2008), is presented in this paper. Figure 4.1 provides an overview of the WESSBAS approach.

To summarize, the contribution of this paper is our WESSBAS approach for automatic extraction of probabilistic workload specifications of session-based application systems, comprising *(i.)* a DSL for modeling session-based probabilistic workload specifications, *(ii.)* an automatic extraction of DSL instances from recorded sessions logs including the clustering of navigational patterns, *(iii.)* transformations from DSL instances to JMeter Test Plans, and *(iv.)* tool support for this approach. The tool support serves as an extensible implementation of the approach, including the DSL, the extraction, as well as a proof-of-concept transformation from the DSL to JMeter Test Plans. Supplementary material for this paper, including the developed tools, models, and experimental results, is publicly available online.[10]

[9]WESSBAS is an acronym for *Workload Extraction and Specification for Session-Based Application Systems*

[10]http://markov4jmeter.sf.net/valuetools14/

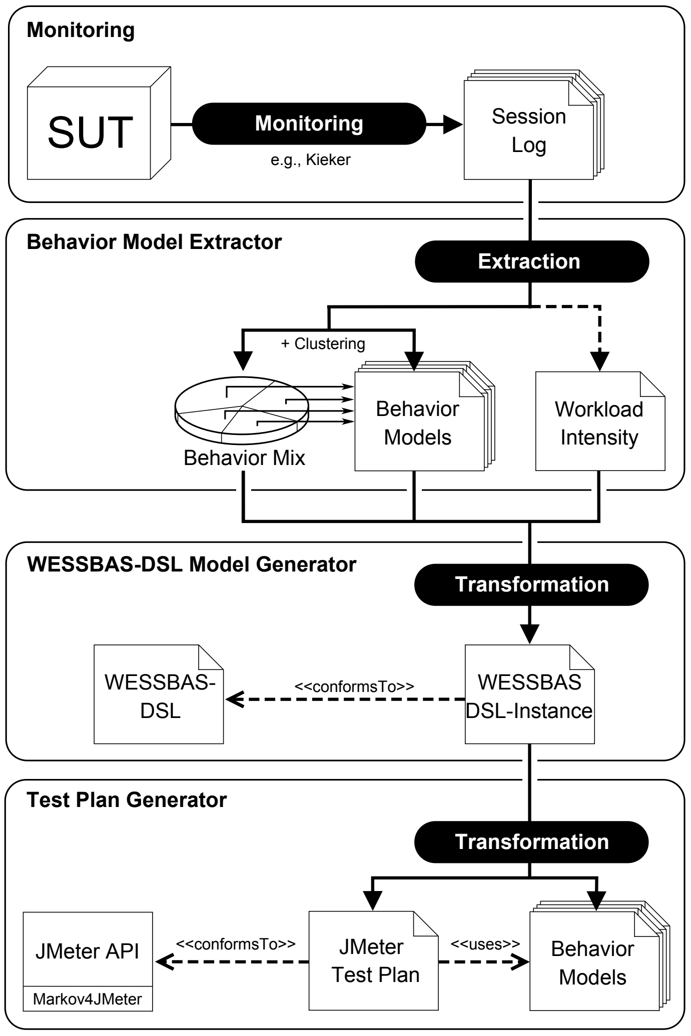


Figure 4.1: *Overview of the* WESSBAS *approach*

4.2 Background and Related Work

The approach described in this paper builds on our previous work on generating probabilistic and intensity-varying workloads (Schulz et al., 2014; van Hoorn/Rohr/Hasselbring, 2008) for session-based systems—particularly the workload modeling formalism that extends the work by Menascé et al. (1999) and Krishnamurthy/Rolia/Majumdar (2006). This section introduces the concepts needed for the remainder of this paper, including a brief discussion of related work.

The workload modeling formalism (*Workload Model*) comprises two different types of models, which will be detailed below (van Hoorn/Rohr/Hasselbring, 2008): *(i.)* an *Application Model*, specifying allowed sequences of service invocations and SUT-specific details for generating valid requests; and *(ii.)* a weighted set of *Behavior Models*, each providing a probabilistic representation of user sessions in terms of invoked services and think times among subsequent invocations. Additionally, the Workload Model includes a function specifying the number of active sessions during the workload generation execution. We developed a publicly available extension, called Markov4JMeter (van Hoorn/Rohr/

Hasselbring, 2008), for the well-known load generator Apache JMeter, allowing to define and execute these Workload Models.

The Application Model is a two-layered hierarchical finite state machine (FSM), consisting of a *Session Layer* and a *Protocol Layer*. Inspired by the work by Krishnamurthy/Rolia/Majumdar (2006), the Session Layer is a finite state machine, in which states refer to system-provided services and allowed transitions among these–possibly labeled with guards and actions. A guard is a boolean expression, defining the condition under which the related application transition fires. An action is a list of statements to be executed, in case the related application transition fires. For each Session Layer state, the Protocol Layer contains an associated FSM, which models the sequence of protocol-level requests to be executed when the Session Layer state is executed.

A Behavior Model roughly corresponds to the Customer Behavior Model Graphs (CBMGs) introduced by Menascé et al. (1999). A Behavior Model $\mathcal{B}_A$ is defined as a tuple $(S \cup \{\$\}, P, z_0, f_{tt})$. S specifies the set of states contained in the Behavior Model with initial state $z_0 \in S$ and exit state \$. $P = [p_{i,j}]$ is an $n \times n$-matrix of transition probabilities, with $n = |S \cup \{\$\}|$. A matrix entry $p_{i,j}$ defines the probability for a transition from state i to state j. The distribution function f_{tt} specifies the think time associated with a transition. Think times can, for instance, be modeled by using random values. The *Behavior Mix* is a set $\{(\mathcal{B}_0, r_0), \ldots, (\mathcal{B}_{m-1}, r_{m-1})\}$, which assigns a relative frequency r_i to the Behavior Model $\mathcal{B}_i$. A tuple $(\mathcal{B}_i, r_i)$ indicates that sessions that correspond to the Behavior Model $\mathcal{B}_i$ are generated with a relative frequency of $r_i \in [0, 1]$. During the workload generation process to a SUT, the Behavior Mix determines the user type to be emulated next by selecting the corresponding Behavior Model based on the assigned relative frequencies.

In the proposed approach, workload specifications representing the measured usage profiles of session-based systems are extracted. A similar approach, yet focusing on the aforementioned CBMGs, has been proposed by Menascé (2002) and Menascé et al. (1999). The authors extract CBMGs from Hypertext Transfer Protocol (HTTP) server logs, including K-means clustering to identify CMBGs for similar types of users. In contrast, in our work an advancement of the K-means algorithm, called X-means is applied.

Our approach focuses on the specification of the behavior of users and offers basic support for modeling workload intensities. An approach focusing on the definition of the workload intensities can be found in v. Kistowski/Herbst/Kounev (2014). It allows a DSL-based definition of variable and dynamic load profiles and workload scenarios over time.

4.3 WESSBAS-DSL

The Wessbas domain-specific language (DSL), referred to as Wessbas-DSL, follows the Markov4JMeter workload modeling formalism (van Hoorn/Rohr/Hasselbring, 2008) introduced in the previous section and therewith denotes a language for expressing such models. In our approach, the Wessbas-DSL is used as an intermediate language between the construction of SUT-specific but tool-agnostic workload models on the one side, and the generation of corresponding inputs to load testing tools on the other side. The WESSBAS-DSL is implemented as an Ecore-based meta-model, using the benefits and tool support of the widely spread Eclipse Modeling Framework (EMF).[11] The meta-model is enriched with a comprehensive amount of constraints (specified in the common Object Constraint Language, OCL), for checking the validity of model instances. The WESSBAS-DSL structure offers a high degree of flexibility and extensibility. The remainder of this section introduces the core concepts of the WESSBAS-DSL. Details are also provided by Schulz (2014).

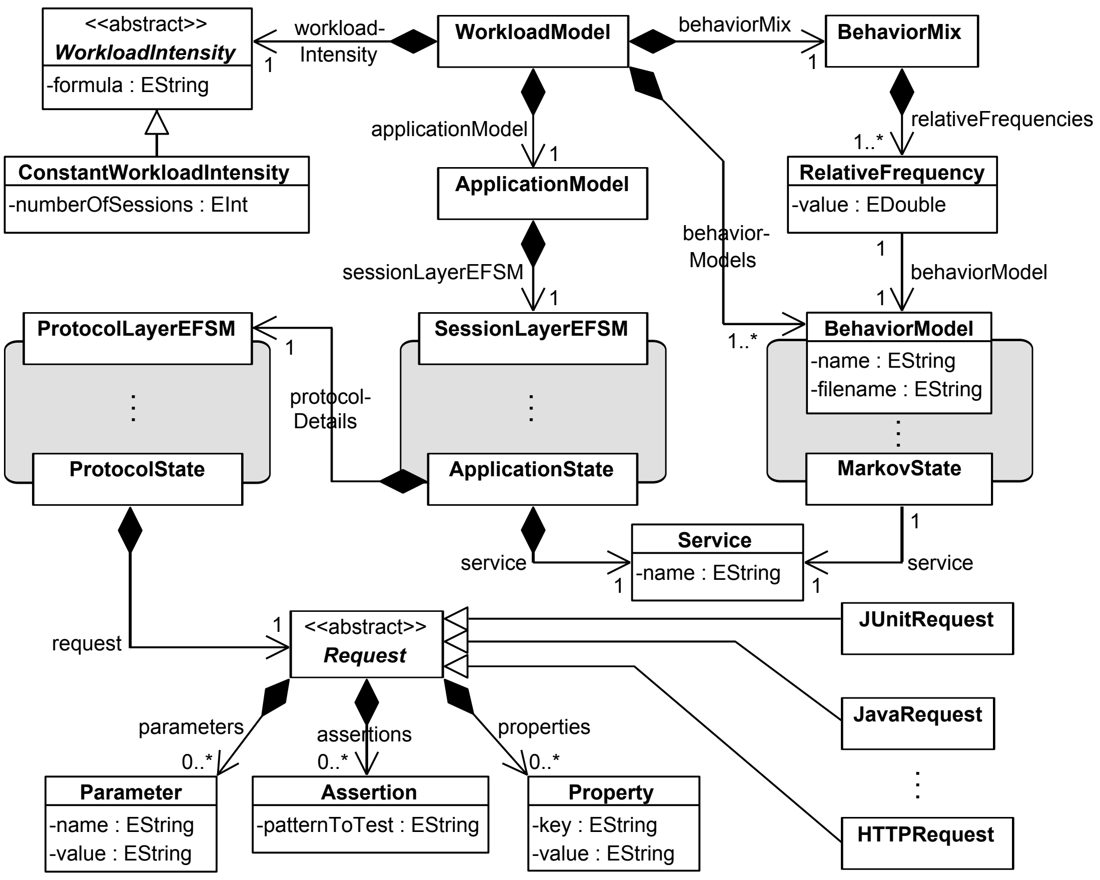
As a language for the Markov4JMeter workload modeling formalism, the WESSBAS-DSL includes the essential components of that model, in particular the Application Model, the (Behavior) Mix of Behavior Models, and the workload intensity, as introduced in Section 4.2. Figure 4.2(a) gives an overview of the WESSBAS-DSL class structure.

The representation of the Application Model corresponds to the two-layered structure of that component, including FSMs for the Session Layer and the Protocol Layer as well. States of the Session Layer FSM, shortly referred to as Application States, are associated with services and Protocol Layer FSMs. States of the Protocol Layer FSMs are associated with protocol-specific requests, which might be of type HTTP, Java, JUnit, BeanShell, SOAP, etc.; the set of currently supported request types can be extended easily by deriving additional subclasses from the common base class. Mention should be made of the difference between properties and parameters of a request: properties correspond to the information which is required for sending a request, e.g., domain, path, or port number of a targeted server; parameters denote values to be sent with the request, e.g., input data for a web form. Behavior Models are modeled as FSMs, with (Markov) States being associated with services. Figure 4.2(b) illustrates the class structure of Behavior Models.

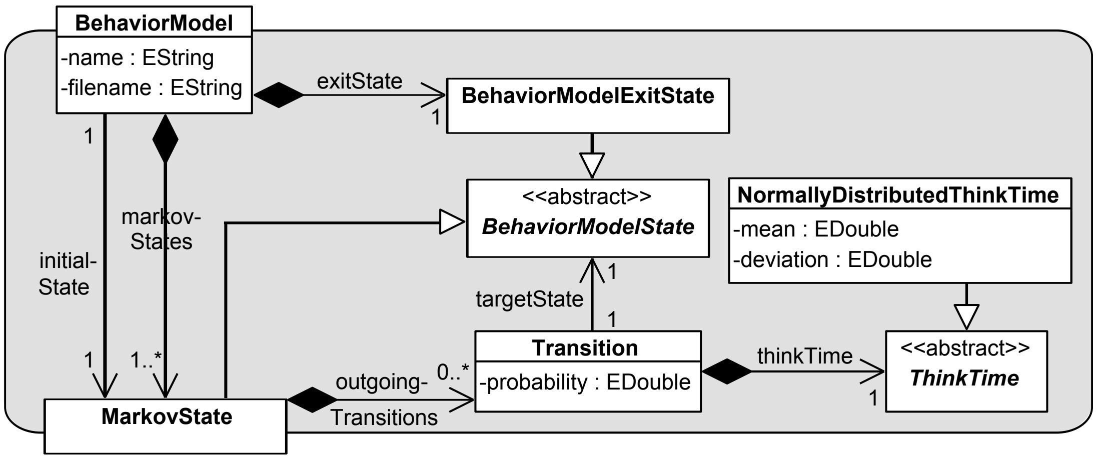
Transitions are labeled with probabilities and think times, whereas think times follow a certain type. Currently supported think times are of type Gaussian, that is, they underlie a normal distribution, indicating mean and (standard) deviation values as parameters. Exit states are modeled explicitly, as they are—in contrast to Markov States—not associated with services. Each Behavior Model is associated with a relative frequency, stored as a double value in a dedicated class. These frequencies are contained in the Behavior Mix, whose corresponding class denotes a further component of the workload model. Session Layer and Protocol Layer FSMs are modeled analogous to Behavior Models, with transitions being labeled with guards and actions. The formula for the workload intensity is stored as a string attribute in a dedicated class that also serves as a base class for all types of workload intensity. This facilitates a simple installation of according formulas, which might be provided by appropriate tools, e.g., (v. Kistowski/Herbst/Kounev, 2014).

Even though the WESSBAS-DSL is independent of specific testing tools, it includes all core information required for generating workload specifications that build on the Markov4JMeter workload modeling formalism. In this paper, we exemplify this by generating JMeter Test Plans through passing WESSBAS-DSL models as input to a transformation tool. This will be further discussed in Section 4.5. The implementation of

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



(a) Wessbas-DSL classes and relationships



(b) Details on the Behavior Model

Figure 4.2: *Overview of the* Wessbas-DSL

the Wessbas-DSL as an Ecore meta-model offers the benefits of EMF tools such as EMF Form Editors or serialization support. In particular, Wessbas-DSL instances can be viewed, validated, and modified in an editor, before being passed as input to any transformation process. The EMF Form Editor offers a constraint *Live-Validation* option, which facilitates the maintenance of Wessbas-DSL models. The extensibility of the Wessbas-DSL is given through its class structure: additional types of workload intensity, requests, or think times can be simply implemented by deriving appropriate subclasses from the related base classes.

4.4 Extracting WESSBAS-DSL Instances

The extraction of WESSBAS-DSL instances is based on so-called *session logs* obtained from raw session information, recorded from a running application system. Raw session information is usually provided by a *request log* generated by monitoring facilities, comprising the associated requests to system-provided services with a session identifier and timestamps for the request and completion time. A typical example is the HTTP request log provided by common web servers (Menascé et al., 1999), or tracing information obtained from application-level monitoring tools (van Hoorn/Waller/Hasselbring, 2012). The session log groups the requests by the session identifier, giving access to the sequence and timing information of subsequent service requests within a session. We will not detail the process of obtaining session logs from request logs any further but refer to existing works (Menascé et al., 1999) and assume that a session log in the WESSBAS format is available. The remainder of this section details the two-step procedure to obtain a WESSBAS-DSL instance, comprising the *(i.)* clustering-based extraction of the Behavior Mix (Section 4.4.1), and the *(ii.)* generation of a complete WESSBAS-DSL instance from the Behavior Mix (Section 4.4.2).

4.4.1 Clustering-Based Behavior Mix Extraction

During the transformation of a session log to a WESSBAS-DSL instance, the Behavior Mix is determined by identifying different groups of customers with similar navigational patterns. As proposed in Menascé et al. (1999), clustering methods can be used to support this task. The identification of different customer groups has several advantages. First, the system can be optimized upon these navigational patterns. Further, the impact of different Behavior Mixes on the performance can be evaluated, e.g., investigating the performance impact of an increased fraction of heavy buyers. To reduce the complexity and to increase the comprehensibility of the resulting Behavior Mix, the goal of the clustering is to obtain a relative small number of clusters.

In this paper, we focus on clustering with the centroid-based X-means algorithm, which is an advancement of the well-known K-means algorithm (Pelleg et al., 2000). The advantage of X-means over K-means is, that it is not mandatory to specify the number of clusters K in advance by the user. The user provides a minimum and a maximum number of resulting clusters and the algorithm determines how many clusters are best suited. The evaluation of K-means clustering is very costly as the results of the K-means must repeatedly be evaluated with different numbers of K (Berkhin, 2006). Further, the X-means algorithm scales better and the risk of finding local minima is lower. The X-means clustering algorithm is integrated into our proposed approach using the data mining framework Weka (Hall et al., 2009). Other algorithms can be integrated accordingly.

Input instances for the clustering are *absolute* Behavior Models, each representing a $n \times n$-matrix of absolute transition frequencies of one user session. Think times are not part of the clustering as they have no impact on the navigational patterns. Each matrix is transformed into a vector, as Weka cannot handle matrices as clustering input. Therefore, the values of a matrix is transformed into a vector by concatenating the rows of the matrix. In a first step, a central vector, called centroid, is determined randomly for each cluster. Each centroid represents a cluster and is the mean of the instances in that cluster. Then, the clustering algorithm iterates several times over the dataset and assigns instances to the nearest cluster centroid, until no instance changes the cluster anymore.

The distance between the instances is calculated using the Euclidean distance metric. During the calculation of a distance, the attributes of the instances can be normalized to a value between zero and one. Without data normalization, attributes with highest variance are driving the clustering. That means in our case, that high transition counts have a high influence on the clustering. In order to figure out the best settings, both the normalized and the non-normalized Euclidean distance will be evaluated in Section 4.6.3.

Having executed the clustering, each attribute of a centroid represents the mean of the respective attribute values of all instances within this cluster. As a result, the centroids represent the absolute Behavior Model of the corresponding cluster. Think times per cluster centroid are determined by calculating the sum of the think times per transition of the respective cluster instances. Finally, the resulting Behavior Mix is calculated like proposed in Menascé et al. (1999). It consists of the (relative) Behavior Models, the mean think times per transition and the relative frequencies of the Behavior Models.

4.4.2 Generating WESSBAS-DSL Instances

The next task is to transform the extracted Behavior Models and the determined Behavior Mix to a valid WESSBAS-DSL instance, which can be further transformed to any test script format. Therefore, our dedicated Java-based implementation, namely WESSBAS-DSL *Model Generator* (Figure 4.1), performs the following three steps: *(i.)* construction of an Application Model, based on SUT-specific states and transitions, *(ii.)* integration of the determined Behavior Mix including the extracted Behavior Models, and *(iii.)* integration of the workload intensity definition.

The construction of an Application Model builds on SUT-specific information, particularly validness of service execution sequences for the Session Layer FSM and protocol-specific information for the Protocol Layer FSMs. The range of such information differs as well as the format it might be provided in; consequently, extensions might be necessary. In our approach, the *Behavior Model Extractor* tool (Figure 4.1) outputs a list of all available services associated with any states of Behavior Models. (Note that SUT-specific Behavior Models are defined on a common set of services.) A small script converts this list into a format that can be processed by the WESSBAS-DSL Model Generator. This information can be enriched with transition specifications; currently, our script generates all possible transitions between services, assuming all sequences of service executions are valid. In particular, neither transition guards nor actions are considered.

After reading the appropriately-formatted input data, the WESSBAS-DSL Model Generator builds a corresponding Session Layer FSM and assigns Protocol Layer FSMs to the Markov States. Assuming that a (virtual) user provides valid input only, the structure of

Table 4.2: *Mapping of* WESSBAS-DSL *concepts to (Markov4)JMeter elements*

WESSBAS-DSL	*Markov4JMeter Elements*
Session Layer FSM	Markov States (+ outgoing transitions)
Protocol Layer FSMs	JMeter Elements (Markov State children)
Workload Intensity	MSC (Session Arrival Controller)
Behavior Mix	MSC (frequency table)
Behavior Models	MSC (frequency table) → CSV-files

MSC = Markov Session Controller

our Protocol Layer FSMs remains trivial with exactly one Protocol State per FSM, indicating exactly one request being sent in a Markov State. A DSL that allows the definition of more complex, protocol-specific FSMs, e.g., failed user logins, denotes a future work issue.

The integration of Behavior Mix and Behavior Models includes the construction of corresponding WESSBAS-DSL fragments. As the Application Layer includes all available services, corresponding Behavior Models can be derived, to be equipped with probabilities and think times provided by the extracted Behavior Models. Finally, the workload intensity is read as a formula string from a properties file, to be included into the resulting model. For further processing, the resulting WESSBAS-DSL model is serialized to an XMI file, using dedicated Ecore techniques. That file can be loaded into an EMF Form Editor to be validated and analyzed, before being passed to the next transformation module.

4.5 Generating JMeter Test Plans

The final task of the extraction process is to transform a given WESSBAS-DSL instance into a corresponding JMeter Test Plan. Our Java-based implementation, namely *Test Plan Generator* (Figure 4.1), reads a serialized WESSBAS-DSL instance from file and constructs a further XMI structure, which can be processed by the JMeter tool. The XMI output is generated via the JMeter API and denotes a JMeter-typical tree structure of Test Plan elements, including Markov4JMeter-specific elements, namely *Markov States* and *Markov Session Controller*, that are provided by the Markov4JMeter add-on for JMeter. The core transformation process builds on a mapping between WESSBAS-DSL concepts and (Markov4)JMeter Test Plan elements. An overview of the underlying mappings is given in Table 4.2.

A Session Layer FSM in the WESSBAS-DSL is mapped to a corresponding set of Markov States in JMeter. Each Markov State includes its individual set of outgoing transitions with guards and actions, for defining the validity of state execution sequences. The name of a Markov State in the resulting JMeter Test Plan corresponds to the name of the state's associated service in the WESSBAS-DSL instance. Protocol Layer FSMs are modeled as child elements of Markov States in the tree-structured result. They are constructed with the use of JMeter controllers and samplers as well, according to their related WESSBAS-DSL structure. The workload intensity is stored as a formula string in the *Session Arrival Controller* sub-component of a Test Plan's (unique) Markov Session Controller. That controller additionally includes a table for Behavior Mix frequencies, to be filled with according values of the input Wessbas-DSL instance. Behavior Models are stored separately—indicated by a separation line in Table 4.2—in CSV-files, which are referred by the frequency table of the Markov Session Controller.

Besides the Test Plan elements that result from the core transformation process for a given Wessbas-DSL instance, several JMeter elements are added to a generated Test Plan by default. This step is required for making a Test Plan's structure accessible for the JMeter tool and providing additional functionality, such as handling of HTTP session cookies. Currently, the Test Plan structure is predefined, targeting HTTP-based tests only; an appropriate mechanism for specifying alternative structures, particularly for different types of requests, denotes a future work issue.

4.6 Evaluation

In this evaluation, we apply our proposed extraction approach and tooling to the industry-standard benchmark SPECjEnterprise2010. This serves as an investigation of *(i.)* the practicality of the approach and tooling support and *(ii.)* the representativeness of the extracted workload specifications. With respect to *(ii.)* we particularly investigate the following two research questions: *(RQ 1) How accurately do the clustering results match the input Behavior Mix?* and *(RQ 2) What is the impact of the clustering results on the workload characteristics*? Section 4.6.1 describes the experimental setting. The SPECjEnterprise2010 deployment is explained in Section 4.6.2. The results for RQ 1 and RQ 2 are detailed in Sections 4.6.3 and 4.6.4.

4.6.1 Evaluation Methodology

An instrumented version of SPECjEnterprise2010[12] is executed with three different Behavior Mixes to obtain a session log, from which instances of the Wessbas-DSL are extracted and transformed into JMeter Test Plans. For the Behavior Model extraction we applied different configurations of the X-means clustering. A basic Application Model is automatically generated from the obtained Behavior Models. Its Session Layer comprises the superset of all states from the Behavior Models, assuming that all transitions between all states are allowed (no guards and actions). The Protocol Layer comprises a mockup HTTP request per state. The transformation from the instances to JMeter Test Plans is performed according to Section 4.4. In order to measure the characteristics of extracted workload models, we developed a web application that is instrumented according to the SPECjEnterprise2010. Hence, the same session log analysis infrastructure can be applied to both the session information obtained from the SPECjEnterprise2010 runs and the JMeter runs for the synthetic workloads of the extracted workload specifications. The reason why we do not execute the extracted workload against the SPECjEnterprise2010 is that currently input parameters for the workload are not extracted automatically.

[12]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.

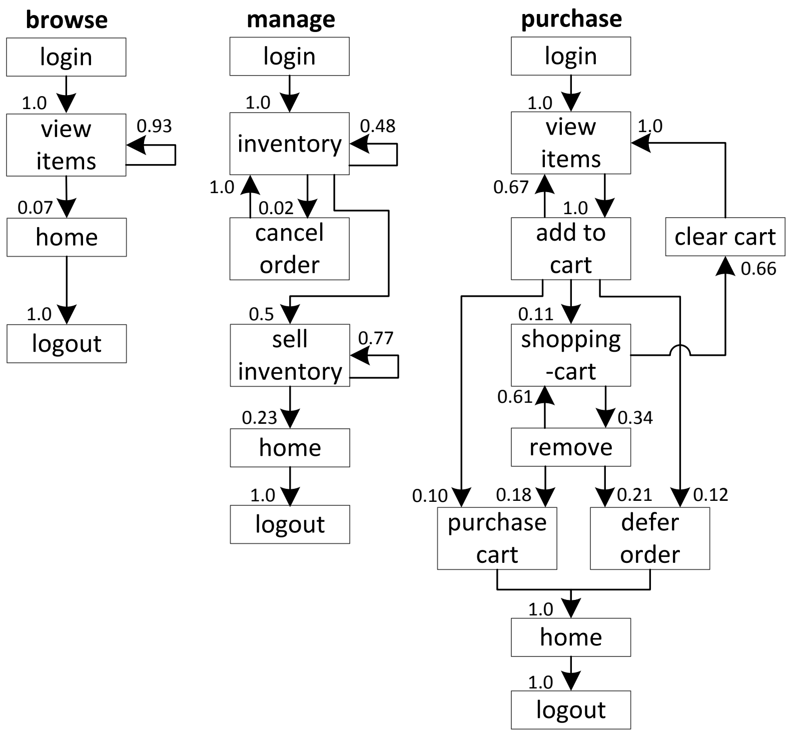


Figure 4.3: *SPECjEnterprise2010 transactions as Behavior Models*

The accuracy of the clustering (RQ 1) is evaluated based on the fraction of misclassified sessions over all classifications of the clustering for a benchmark run. The impact of the clustering on the workload characteristics (RQ 2) is evaluated based on *(i.)* two session-based metrics, session length as number of requests per sessions and number of distinct session types, as well as *(ii.)* a request-based metric, namely the relative invocation frequency of all request types. Note that due to the nature of the SPECjEnterprise2010 workload we do not consider timing-related metrics such as think times or arrival rates, even though they are correctly extracted and executed by our approach. Conclusions about the arrival rates of requests can be drawn by looking at the invocation frequencies of requests.

4.6.2 SPECjEnterprise2010 Deployment

SPECjEnterprise2010 is a Java EE application representing a business case that combines Customer Relationship Management (CRM), Supply Chain Management (SCM), and manufacturing. It includes a workload specification and a dataset needed for the execution of load tests. The workload is generated by the Faban Harness and Benchmark Driver.[13] The benchmark consists of three different application domains, namely *Orders domain* (CRM), *Manufacturing domain*, and *Supplier domain* (SCM). In this work we consider only the Orders domain, which represents a typical web-based application that provides e-commerce functionality to the customers—in this case automobile dealers. The customers are able to purchase and sell cars, to manage their accounts and dealership inventory, and to browse the catalogue of cars. The Orders domain represents the SUT.

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

4.6.2.1 Workload Description

SPECjEnterprise2010 defines three different transaction types which are executed by automobile dealers: *Browse (B)*, *Manage (M)*, and *Purchase (P)*. Within Browse, the benchmark driver navigates to the catalogue of available cars and browses the catalogue for a constant number of times. Manage describes a scenario where open orders are canceled and vehicles are sold. In the more complex transaction type Purchase, orders are placed and immediately purchased or deferred. The shopping cart is either cleared or items are removed one by one until only one item remains. Each of these transaction types is a sequence of HTTP requests. In total, 13 different HTTP request types are defined. Within the transactions, no think times are defined, i.e., each HTTP action is executed directly after its previous request has been completed. Therefore, the evaluation of think times extraction is out of scope for this paper. Figure 4.3 depicts the structure of the three transaction types as Behavior Models obtained by applying our Wessbas extraction approach.

In the original benchmark workload, automobile dealers log in to the system, execute *multiple instances* of the three transactions types, and log out. Each of the three transaction types is executed with a specified probability. The standard transaction mix is 50% Browse, 25% Manage, and 25% Purchase. We modified the dealer driver such that *each* transaction starts with a login and ends with a logout. This way each transaction corresponds to a unique session, and the transaction mix corresponds to the Behavior Mix.

4.6.2.2 Benchmark Execution and Monitoring

Three different transaction mixes are used to evaluate the proposed approach. For each mix, one of the transaction types is executed with a probability of 50% and the other two with 25% each. A load of 800 concurrent users is executed, resulting in a moderate Central Processing Unit (CPU) utilization of the SUT of approximately 40%. Each benchmark run is executed for ten minutes after a four minute ramp-up phase and before a four minute ramp-down phase.

Table 4.3: *Clustering Results*

		X-means (min 3 cluster, max 3 cluster)								X-means (min 2 cluster, max 20 cluster)							
		ED				NED				ED			NED				
TM	T	C1	C2	C3	MC	C1	C2	C3	MC	C1	C2	MC	C1	C2	C3	C4	MC	N
50	B	0	0	31,060		0	31,060	0		0	31,060		0	0	0	31,060		
25	M	15,298	0	0	2.91%	15,298	0	0	0%	15,298	0	24.62%	632	14,666	0	0	1.03%	61,500
25	P	1,789	13,353	0		0	0	15,142		15,142	0		0	0	15,142	0		
25	B	15,091	0	0		15,091	0	0		0	15,091		0	15,091	0	0		
25	M	0	0	15,000	15.98%	0	15,000	0	0%	15,000	0	24.96%	0	0	707	14,293	15.30%	60,089
50	P	0	20,397	9,601		0	0	29,998		29,998	0		21,513	8,485	0	0		
25	B	0	15,231	0		15,231	0	0		0	15,231		0	0	0	15,231		
50	M	30,510	0	0	2.99%	0	30,510	0	0%	30,510	0	25.16%	29,375	1,135	0	0	1.86%	61,118
25	P	1,824	0	13,553		0	0	15,377		15,377	0		0	0	15,377	0		

In order to obtain the raw session information, the SUT was instrumented using Kieker (van Hoorn/Waller/Hasselbring, 2012). For each request the requested URL, the query string, the session ID, and the server-side entry and exit timestamp are recorded. Afterwards, the raw log data is transformed to a session log. During the transformation, the sessions within the ramp-up and ramp-down phase are not taken into account. In order to be able to evaluate the clustering results of the transaction types, the name of the transaction type is added as additional parameter to the login HTTP action.

4.6.3 Accuracy of Clustering

The evaluation of clustering accuracy (RQ 1) is split into two steps. In the first step, the accuracy of the clustering is determined based on the assumption that the number of resulting clusters is known in advance. For this reason, the number of resulting clusters is fixed to three. As the number of clusters is usually not known in advance, we let the X-means algorithm determine the number of clusters in a second step. As the seed value for the random selection of the initial centroids can have a high impact on the clustering results, multiple clustering runs with different seed values between one and twelve are executed. Afterwards, the run with the lowest sum of squared error value (Pelleg et al., 2000) is selected.

The results of the clustering are presented in Table 4.3. For each Transaction Mix (TM), the clustering shows for each Transaction Type (T) the cluster (C_x) where a transaction is assigned to, and the Percentage of Misclassified (MC) transactions. The left side shows the results of exactly three predefined clusters (step one); the right side shows the results letting X-means determine the number of clusters between two and twenty (step 2). The number of transactions (N) clustered for each transaction mix is around $61,000$.

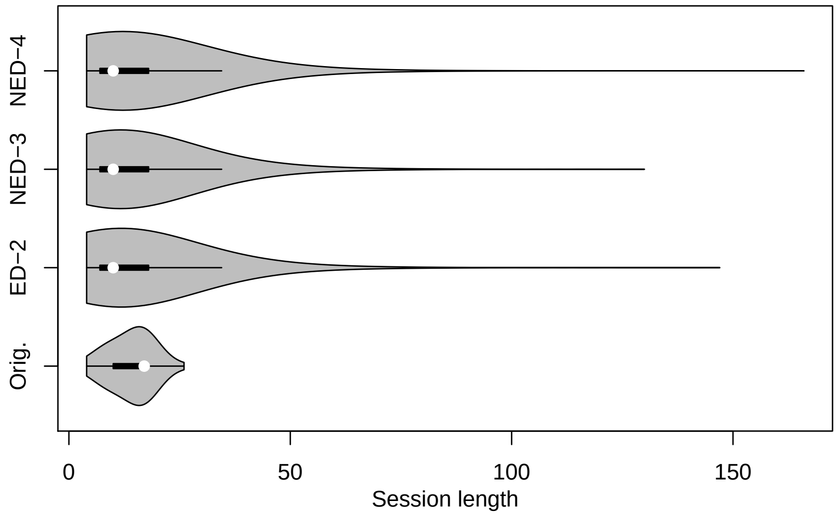
The results using exactly three clusters indicate that the clustering using Normalized Euclidean Distance (NED) is able to cluster all transactions correctly (100%) resulting in the Behavior Models shown in Figure 4.3. The clustering using Euclidean Distance (ED) without normalization classifies the transactions Browse and Manage correctly, whereas a fraction of transactions of type Purchase is assigned mistakenly to the same cluster as the Manage transactions. In the second transaction mix, the fraction of Purchase transactions is higher than in the other mixes. Hence, the percentage of misclassified transactions is with 15.98% relatively high.

The clustering without predefining the correct number of clusters, results in two clusters using ED and four clusters using NED. As clustering with ED always merges transactions of type Purchase and Manage, the percentage of misclassified transactions is around 25% for all mixes. It is assumed that the transaction type with the lower number of instances merged within one cluster count as missclassified. The clustering using NED always correctly classifies Browse transactions. Manage transactions are always split into two clusters whereas Purchase is only split into two clusters in the second transaction mix. Hence, the percentage of misclassified transactions is again relatively high (15.3%) in the second transaction mix.

Transactions of type Browse seem to be homogeneous in a way that they were clustered correctly among all clustering runs. This can be explained as Browse transactions are executed with a constant number of actions without probabilistic behavior. NED is better suited to cluster the different transaction types than the non-normalized version. The normalization has the effect that high transaction counts and therefore also the length of the sessions has a lower impact on the clustering. Thus, the structure of the transactions in terms of the number of different HTTP requests grows in significance. As each of the three transaction types consist of different HTTP request types (except for login, home and logout), the clustering results are significantly better.

4.6.4 Accuracy of Workload Characteristics

To evaluate the accuracy of the extracted workload specifications (RQ 2), we compare the server-side session-based and request-based metrics mentioned in Section 4.6.1 for the original measurements with the corresponding metrics obtained by executing extracted workload specifications using JMeter. Due to space limitations, we present only the results of the original benchmark Behavior Mix (25% P, 50% B, and 25% M), using the X-means clustering algorithms results with 2 (ED), 3 (NED), and 4 (NED) clusters (entries for the bottom TM in Table 4.3). The original workload includes 61, 500 sessions and 847, 927 HTTP requests. These numbers served as an approximate stopping criteria during the execution of the synthetic workload with JMeter (cf. Figure 4.4(b) and Figure 4.5(b)).



(a) Violin plot (combination of box and density plot)

	Min.	Q_1	*Med.*	*Mean*	$CI_{0.95}$	Q_3	*Max.*	*N*
Orig.	4	10	17	14.23	[14.19,14.26]	17	26	61,500
ED-2	4	7	10	14.24	[14.15,14.33]	18	147	60,957
NED-3	4	7	10	14.24	[14.15,14.33]	18	130	62,054
NED-4	4	7	10	14.26	[14.17,14.35]	18	166	59,971

(b) Summary statistics

Figure 4.4: *Session length statistics for the original workload (Orig.) and the synthetic workloads (ED-2, NED-3, NED-4)*

4.6.4.1 Session Length and Distinct Sessions

Statistics about the session length distributions of the original and the three synthetic workloads are listed in Figure 4.4. Looking only at the mean values and the 0.95 confidence interval (Figure 4.4(b)), one may conclude that the session length distributions of the three synthetic workloads exactly match the distribution of the original workload. However, particularly the violin plot (Figure 4.4(a)) indicates that the synthetic distributions are similar but differ considerably from the original workload. The quartile-based statistics in Table 4.4(b) confirm this observation.

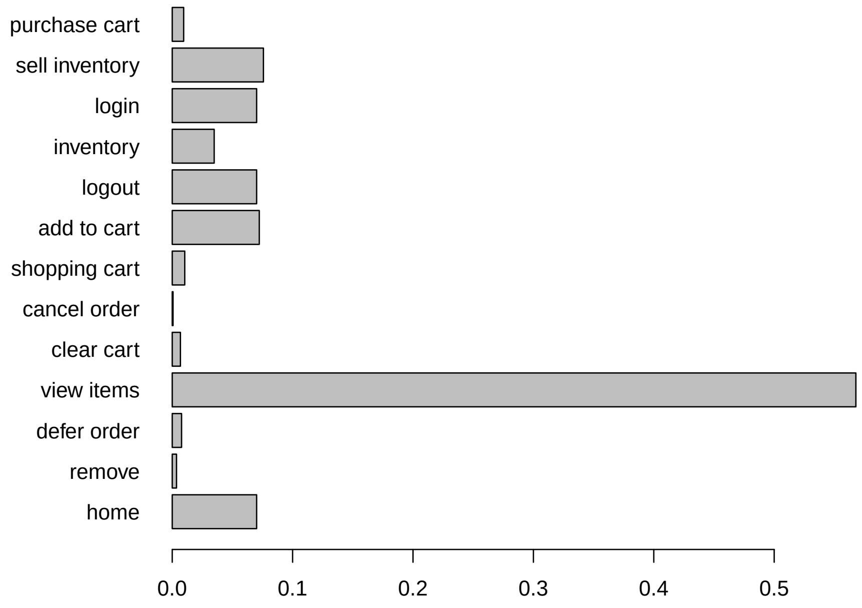
It can be observed that for the synthetic workloads, very long sessions are generated. While for the original workload the longest sessions comprise 26 requests, the synthetic sessions reach maximums of 147, 130, and 166. Looking at the individual session lengths, 11% of the synthetic sessions are longer than the longest sessions of the original workload.

In the original workload, we identified 78 distinct sessions. The number of distinct sessions in the synthetic workloads is considerably higher, namely 2, 126 (2 clusters), 2, 144 (3 clusters), 1, 996 (4 clusters). The relatively low number of distinct session types is caused by the fact that the original SPECjEnterprise2010 workload contains only few probabilistic elements, which are all bounded in the number of maximum iterations. Hence, the maximum number of possible distinct sessions is countable. After having described the session length distributions of the synthetic workloads, the high number of distinct sessions is not surprising. Inspecting the structure of the synthetic sessions, we observed the following recurring patterns: *(i.)* sell inventory+, *(ii.)* inventory+, *(iii.)* view items+, *(iv.)* (view items, add to cart)+, *(v.)* (view items, add to cart, shopping cart, clear cart)+. These patterns can be explained by the corresponding transitions with high probabilities already indicated by the probabilities of the original workload depicted in Figure 4.3.

Considering the setting for SPECjEnterprise2010, the following conclusions can be drawn about the impact of the clustering results on the session-based metrics session length and number of distinct session types. No statistically significant differences between the synthetic workloads for 2, 3, and 4 clusters can be observed. Both the session length distributions and the number of distinct sessions deviate from the characteristics of the original workload. The deviation of the session length distributions is mainly caused by a number of synthetic long sessions. The mean value shows no statistically significant difference.

4.6.4.2 Request Counts

Figure 4.5 depicts statistics about the frequency of invoked requests. Based on the absolute numbers of requests to the 13 SPECjEnterprise2010 request types. We computed the relative frequencies for the original workload and the three synthetic workloads. An exact match of the relative frequencies could be observed. That is, the deviation, e.g., in form of the sum of squared errors, is zero. Hence, from the server-perspective, the synthetic workloads provide representative workloads in terms of the distributions of requests. A barplot, which looks the same for each of the four workloads, is shown in Figure 4.5(a).



(a) Relative counts (common to all workloads)

	Request	*Orig.*	*ED-2*	*NED-3*	*NED-4*	*Rel.*
1	add to cart	63,761	63,316	64,250	61,838	0.07
2	cancel order	632	607	634	591	0.00
3	clear cart	6,047	5,941	6,140	5,843	0.01
4	defer order	6,782	6,799	6,863	6,651	0.01
5	home	59,934	60,957	62,054	59,971	0.07
6	inventory	30,596	30,212	31,378	29,808	0.03
7	login	61,500	60,957	62,054	59,971	0.07
8	logout	59,934	60,957	62,054	59,971	0.07
9	purchase cart	8,360	8,328	8,351	8,139	0.01
10	remove	3,027	2,993	3,044	3,064	0.00
11	sell inventory	66,679	65,413	67,691	64,794	0.08
12	shopping cart	9,074	8,934	9,184	8,907	0.01
13	view items	498,601	492,675	499,983	485,611	0.57
	$\sum$	874,927	868,089	883,680	855,159	1.00

(b) Absolute and relative counts

Figure 4.5: *Request count statistics*

4.7 Conclusion and Future Work

In this paper, we presented our WESSBAS approach for the systematic extraction and specification of probabilistic workloads for session-based systems including a transformation to the load testing tool Apache JMeter. To address the challenge of specifying workloads for different workload tools, we first introduced a domain-specific language that describes the structure of a workload in a generalized way. Additionally, we demonstrated how groups of customers with similar behavioral patterns can be identified using clustering algorithms. Finally, the evaluation with the standard industry benchmark SPECjEnterprise2010 demonstrated the practicality and high accuracy of the proposed approach.

As future work, we plan to further automate the generation of Application Models, including Protocol Layer and test data, as well as automatic learning of guards and actions (Shams/Krishnamurthy/Far, 2006). We want to extend the set of supported logging formats and load testing tools. The measurement-based approach will be combined with model-based performance evaluation approaches (Vögele et al., 2014) by generating workload specifications of performance models from WESSBAS-DSL instances. Moreover, the evaluation of other clustering algorithms and the integration of approaches for the generation of varying workload intensities (v. Kistowski/Herbst/Kounev (2014)) will be investigated.

Chapter 5

Using Performance Models to Support Load Testing in a Large SOA Environment

Authors	Vögele, Christian[1] (voegele@fortiss.org) Brunnert, Andreas[1] (brunnert@fortiss.org) Danciu, Alexandru[1] (danciu@fortiss.org) Tertilt, Daniel[1] (tertilt@fortiss.org) Krcmar, Helmut[2] (krcmar@in.tum.de) [1]fortiss GmbH, Munich, Germany [2]Chair for Information Systems, Technical University of Munich (TUM), Garching, Germany
Outlet	International Workshop on Large Scale Testing 2014 (LT 2014)
Status	Accepted
Individual Contribution	Problem and scope definition, construction of the conceptual approach, prototype development, paper writing, paper editing

Table 5.1: *Bibliographic details for P2*

Abstract Load testing in large SOA environments is especially challenging when services are under the control of different teams. It gets even more difficult if the capacity of services, which are part of a load test, need to be scaled before the test starts. It is thus important to know which service operations will be called and how many requests arrive at each service while executing a load test. We propose the use of performance models to derive this information for SOA-based applications before executing load tests. Additionally, we use these models to select usage scenarios on which load test scripts can be based. To evaluate the service workload of the selected usage scenarios, these models are transformed in a way that each scenario can be simulated separately from each other. Predictions based on these transformed models allow to gather the required information about the service call frequency.

5.1 Introduction

This paper describes an approach to support load testing in a large service-oriented architecture (SOA) environment using performance models. The approach is developed and applied in an ongoing project that transforms an IT landscape to a SOA[14]. The SOA paradigm describes how loosely coupled software components offer services in a distributed environment (Liu/Gorton/Zhu, 2007). SOA enables the integration of legacy applications and aims at increasing the flexibility of enterprise IT environments. However, the transformation into a SOA is associated with technical challenges. One of the key challenges is to ensure that given performance requirements (i.e. response times for a given workload) are met by enterprise applications in such an environment. Applications that need to be integrated into a SOA are often not designed for this type of interaction. New access patterns and additional software layers lead to different performance characteristics.

Especially for enterprise applications that extensively reuse existing services, it is important to evaluate the performance before they are rolled out to a production environment (Liu/Gorton/Zhu, 2007). Planning and executing load tests to evaluate their performance is a cumbersome task. Often several teams are involved in this process and the services provided by these teams are not yet in production. The IT environment in our project context comprises more than twenty complex information systems maintained by several teams in the organization. These information systems provide more than seventy different services in the SOA environment. It is thus required to not only estimate the workload for the enterprise applications but also for the services used during a test. Estimating this information is necessary so that a test environment can be set up correctly for the expected load. Otherwise, the performance evaluation results might be interesting for the service providers but not for the enterprise application consuming these services. We propose an approach to support these tasks using performance models.

The applications we are evaluating are used by 10,000 users in a pilot phase whereas 100,000 users are expected once they are in production. These applications are developed using a model-driven development approach and are modeled as Unified Modeling Language (UML) activity diagrams. These diagrams contain the application control flow including information about which service operations are called. Additionally, data about the expected user behavior represented as user think times and call probabilities is integrated into the UML activity diagrams. This information is collected by interviewing domain experts.

In the next section, we describe how performance models derived from these UML activity diagrams can be used to support load testing. First, we explain how usage scenarios are extracted from performance models. These usage scenarios build the foundation for the creation of load test scripts. Afterwards, we describe how performance model predictions help to estimate the service workloads for selected usage scenarios.

[14]Project details can be found in our previous work (Brunnert et al., 2013)

5.2 Using Performance Models to Support Load Testing

In order to reduce the modeling effort, UML activity diagrams are automatically transformed into performance models. The PCM (Becker/Koziolek/Reussner, 2009) is used as meta-model for the performance models. The PCM modeling notation is closely aligned with the UML notation and is thus easily comprehensible for technical staff in an organization. A detailed description of the transformation process from UML activity diagrams to PCM models can be found in Brunnert et al. (2013). The next section describes how usage scenarios which serve as basis for load test scripts can be extracted from the performance models.

5.2.1 Extracting Usage Scenarios

A usage scenario is defined as a path from a specified start element to one of the specified end elements of a PCM model. Parameterizable control flow elements like probabilistic branches and loops describe the sequence of the modeled user actions within these paths and represent the information collected from the domain experts (Becker/Koziolek/Reussner, 2009).

To detect all possible scenarios, the generated PCM models are traversed recursively using a depth-first search. While traversing the PCM models, the call probability for each scenario is calculated by multiplying the probabilities of all branch transitions within a specific path. User defined thresholds for the minimum likelihood of execution and the minimum (and/or maximum) number of user actions within a scenario can be defined. These thresholds help to avoid that too many scenarios are extracted and that endless loops occur.

The result of the depth-first search is a set of usage scenarios including their probability of being called. Additionally, the number of user actions and the services (and their operations) called during a usage scenario is provided as a result. Based on this information test experts can select a set of scenarios for the load test which match their test goals. Afterwards, load test scripts for each of these scenarios are created manually.

5.2.2 Transforming Performance Models

The PCM models generated based on the UML activity diagrams are modeled in a way that all usage scenarios are represented by one graph. Thus, it is not possible to evaluate only selected usage scenarios as they are not modeled separately from each other. Therefore, we transform the PCM models again. The PCM models resulting from this transformation represent the usage scenarios independently from each other. Thus, single usage scenarios can be excluded or included for the following analysis phase by adjusting their probabilities manually. To exclude single usage scenarios their call probability can be set to zero. The probability of the remaining usage scenarios must then be extrapolated to one.

Additionally, the PCM models are enhanced with information about worst case response times for specific services and their operations as specified in service level agreements. Using these transformed models as input for a simulation engine allows to derive predictions for selected usage scenarios and given workloads.

5.2.3 Analyzing Prediction Results

The prediction results show how many requests arrive on each service operation for a given workload. Additionally, the expected throughput for the given usage scenarios can be derived from these results. The simulation time should be chosen according to the planned execution time of the load tests to simplify the analysis.

The number of simulated users can be varied to assess the impact of different user counts. This is especially important if services involved in a load test need to be scaled in an integration or pre-production environment before a test starts. Therefore, the prediction results should be communicated to each service development team. These teams can then ensure that given service level agreements can be met for the expected workload on the service.

5.3 Conclusion and Future Work

As shown in this paper, predictions based on performance models can greatly benefit load test planning and execution in a SOA project. Future work for the proposed approach includes automatic load test script generation for selected usage scenarios. Additionally, we investigate the use of machine learning to prioritize usage scenarios based on the test goals automatically, e.g. using scenarios that are most likely, that include the most service calls or which lead to the highest resource utilization. Capacity planning using performance models enhanced with resource demand information as shown in Brunnert/Vögele/Krcmar (2013) is another area to pursue in the future.

Chapter 6

Automatic Extraction of Session-Based Workload Specifications for Architecture Level Performance Models

Authors	Vögele, Christian[1] (voegele@fortiss.org) van Hoorn, André[2] (andre.van.hoorn@acm.org) Krcmar, Helmut[3] (krcmar@in.tum.de) [1]fortiss GmbH, Munich, Germany [2]Institute of Software Technology, University of Stuttgart, Stuttgart, Germany [3]Chair for Information Systems, Technical University of Munich (TUM), Garching, Germany
Outlet	International Workshop on Large Scale Testing 2015 (LT 2015)
Status	Accepted
Individual Contribution	Problem and scope definition, construction of the conceptual approach, prototype development, experiment design, execution and result analysis, paper writing, paper editing

Table 6.1: *Bibliographic details for P3*

Abstract Workload specifications are required in order to accurately evaluate performance properties of session-based application systems. These properties can be evaluated using measurement-based approaches such as load tests and model-based approaches, e.g., based on architecture-level performance models. Workload specifications for both approaches are created separately from each other which may result in different workload characteristics. To overcome this challenge, this paper extends our existing Wessbas approach which defines a domain-specific language (Wessbas-DSL) enabling the layered modeling and automatic extraction of workload specifications, as well as the transformation into load test scripts. In this paper, we extend Wessbas by the capability of transforming Wessbas-DSL instances into workload specifications of architecture-level performance models. The transformation demonstrates that the Wessbas-DSL can be used as an intermediate language between system-specific workload specifications on the one side and the generation of required inputs for performance evaluation approaches on the other side. The evaluation using the standard industry benchmark SPECjEnterprise2010 shows that workload characteristics of the simulated workload match the measured workload with high accuracy.

6.1 Introduction

In order to validate whether non-functional performance requirements like given response times of application systems can be met, measurement- and model-based performance evaluation approaches are applied (Woodside/Franks/Petriu, 2007). Workload specifications are required for both approaches. Workload specifications serve as input for measurement-based approaches in order to generate synthetic workload to the system under test (SUT), i.e., executing a set of consecutive and related customer requests within a session (Krishnamurthy/Rolia/Majumdar, 2006; Menascé et al., 1999). Additionally, these specifications are taken into account in formalisms for model-based approaches to predict performance properties early in the software development cycle (Becker/Koziolek/Reussner, 2009; Koziolek, 2010; Woodside/Franks/Petriu, 2007).

To ensure that the measured and predicted performance characteristics of the SUT are similar, corresponding workload specifications must be used. However, there is a lack of approaches enabling the common automatic extraction and specification of workloads for both approaches. The extraction and specification of workloads is done separately for each approach and each tool. This results in additional specification and maintenance effort. The reasons for this are, that these approaches are not integrated and that workload specifications are defined on different levels of detail. Measurement-based approaches need detailed system-specific information like protocol data, whereas model-based approaches are specified on a more abstract level.

In response to these challenges, this paper extends our WESSBAS approach[15] (van Hoorn/Rohr/Hasselbring, 2008; van Hoorn et al., 2014), originally developed to automatically extract probabilistic workload specifications for load testing session-based application systems. So far, WESSBAS comprises a *(i)* domain-specific language (WESSBAS-DSL), intended to be a system- and tool agnostic intermediate modeling language for workload specifications, and *(ii)* a transformation to load test scripts. The contribution of this paper is the extension of the WESSBAS approach for model-based performance evaluation. Therefore, we propose a transformation of the WESSBAS-DSL into workload specifications of the PCM (Becker/Koziolek/Reussner, 2009), representing an architecture-level performance modeling language. Along with the existing WESSBAS-DSL extraction, this transformation can be exploited by model-based approaches. We focus on architecture-level performance models as they allow to model system architecture, execution environment, and workload specification separately from each other (Brosig/Huber/Kounev, 2011). We evaluate the approach using the SPECjEnterprise2010 benchmark. The developed tools, models, and results of this paper are publicly available online.[16]

[15]WESSBAS is an acronym for *Workload Extraction and Specification for Session-Based Application Systems*

[16]http://markov4jmeter.sf.net/lt15

6.2 Related Work

In order to automate the extraction of workload specifications for session-based applications systems, several measurement-based approaches (Arlitt/Krishnamurthy/Rolia, 2001; Barford/Crovella, 1998; Menascé et al., 1999) were introduced. These approaches generate workload specifications in tool-specific formats which are not envisaged for model-based approaches.

To evaluate performance properties with architecture-level performance models several approaches were introduced enabling the automatic generation of these models (Brosig/Huber/Kounev, 2011; Brunnert/Vögele/Krcmar, 2013). These approaches focus on the automatic extraction of the system-specific details of the SUT, like the system components, the relationship between the components, the component allocations, and the resource demands. However, the workload specifications must still be modeled manually, which requires a lot of effort for the performance expert.

To reduce the complexity of generating different kinds of analytical performance models from architecture-level performance models several intermediate languages such as PUMA (Woodside et al., 2005) or Klaper (Ciancone et al., 2011) were introduced. These approaches only focus on model-based performance evaluations and do not support the definition of workload specifications for session-based software systems.

6.3 Transforming WESSBAS-DSL Instances into PCM

Before describing the transformation from WESSBAS-DSL instances into PCM workload specifications (Section 6.3.3), we introduce the required concepts of the WESSBAS approach (Section 6.3.1) and PCM (Section 6.3.2).

6.3.1 WESSBAS *Approach*

The WESSBAS approach, depicted in Figure 6.1, introduces a *(i)* domain-specific language (DSL) for layered modeling of workload specifications, *(ii)* an automatic extraction of WESSBAS-DSL instances from session logs, and *(iii)* a transformation from these instances into load test scripts (van Hoorn et al., 2014). The performance model generation process, which is the contribution of this paper, is detailed in Section 6.3.3. The WESSBAS-DSL represents a modeling language for workload specifications of session-based systems. The specification is based on our previous work on the definition and extraction of probabilistic workload specifications (van Hoorn/Rohr/Hasselbring, 2008; van Hoorn et al., 2014) for session-based systems. It describes a formalism of workload specifications based on Menascé et al. (1999) and Krishnamurthy/Rolia/Majumdar (2006). The advantage of using this DSL is that it defines the structure of valid workload specifications for session-based systems.

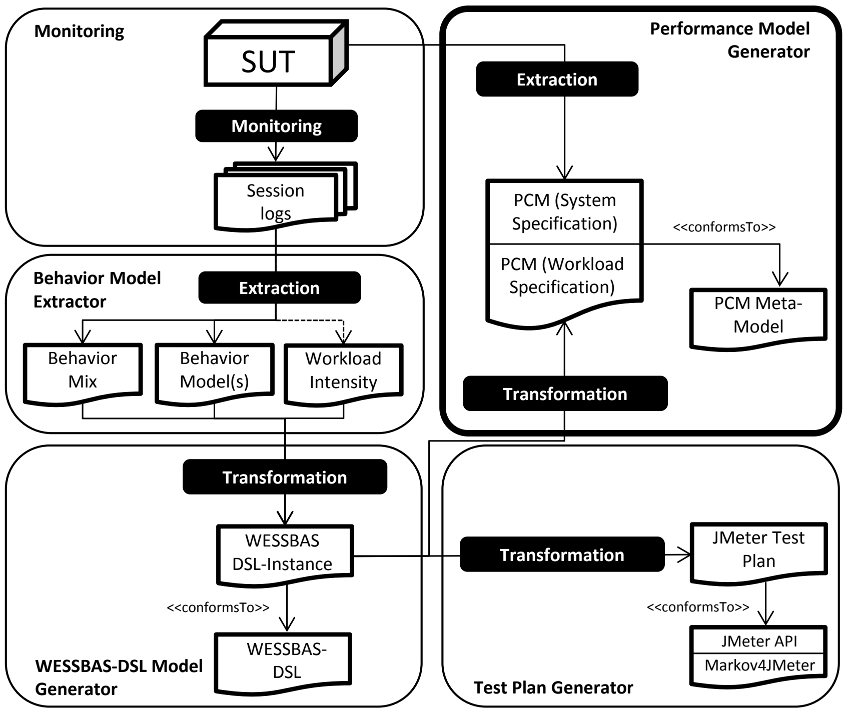


Figure 6.1: *Overview of* WESSBAS *approach and its extension (bold rectangle), adapted from van Hoorn* et al. (2014)

The WESSBAS-DSL enables the modeling of all aspects of a workload model for session-based systems: *Workload Intensity*, *Application Model*, *Behavior Models*, and *Behavior Mix* (van Hoorn/Rohr/Hasselbring, 2008; van Hoorn et al., 2014). The Workload Intensity defines the arrival rates of new sessions as a function over time. The Application Model includes a Session Layer and a Protocol Layer. The Session Layer specifies the allowed sequences of service invocations and SUT-specific details to generate valid requests as Extended Finite State Machine (EFSM). The Protocol Layer models the sequence of protocol-level requests to be executed on the real system. Behavior Models are a probabilistic representation of user sessions in terms of invoked services, associated with Markov States. Transitions between Markov States are labeled with think times and call probabilities. The Behavior Mix specifies the relative frequencies of different Behavior Models representing different customer groups, e.g., Behavior Models for heavy users and/or occasional buyers. These customer groups are identified in the WESSBAS approach using clustering algorithms.

WESSBAS-DSL instances are extracted from recorded session logs of the SUT. From these instances, the test plan generator generates load test scripts for the common load testing tool Apache JMeter[17] including the extension Markov4JMeter (van Hoorn/Rohr/Hasselbring, 2008). Further details on the WESSBAS approach can be found in (van Hoorn et al., 2014).

[17]http://jmeter.apache.org/

Table 6.2: *Mapping of* WESSBAS-DSL *concepts to PCM model elements*

WESSBAS-DSL	*PCM Model Elements*
Behavior Models	Repository Model (Basic Component, RDSEFF)
Session Layer FSMs	not required
Protocol Layer FSMs	not required
Workload Intensity	Usage Model (Closed Workload)
Behavior Mix	Usage Model (Branch)

6.3.2 Palladio Component Model

PCM is a modeling language enabling the prediction of quality-of-service attributes (QoS) like response times, utilization, and throughput (Becker/Koziolek/Reussner, 2009). PCM is composed of five complementary model types. The central model type is the *Repository Model*. It models the software components, component operations, and the relations between them. The modeled components are then assembled in a *System Model* to represent the application system. Resource containers (e.g., servers) and their associated hardware resources are modeled in the *Resource Environment Model*, whereas the *Allocation Model* defines the allocation of assembled components to the resource container. The *Usage Model* defines the workload of the system.

As our proposed approach focuses on the generation of PCM workload specifications, the system-specific parts of the model must be created in a separate step. As manual modeling requires too much effort, approaches which automatically extract PCM instance from design specification or running applications, e.g., (Brosig/Huber/Kounev, 2011; Brunnert/Vögele/Krcmar, 2013) can be used to generate the system-specific part of the SUT.

6.3.3 Transformation

The PCM Usage Model offers only basic support for modeling complex workloads. For example, they grow in complexity for larger workloads due to the lack of reuse concepts. Consequently, we cannot transform the WESSBAS-DSL solely to the usage model. As the Repository Model offers this kind of structuring, we generate parts of the workload specification into the Repository Model (cf. Vögele et al. (2014)). This violates the clear separation of the PCM models but reduces the complexity of the transformation considerably. Further, this way we do not need to extend the PCM meta-model.

The transformation maps elements of the WESSBAS-DSL to elements of PCM as described in Table 6.2. First, the existing PCM Repository Model is loaded and for each Behavior Model of the WESSBAS-DSL a component with a corresponding interface used to represent the relationships between the components is generated. For each Markov State of a Behavior Model a component operation, represented as RDSEFF (Becker/Koziolek/Reussner, 2009), is created. RDSEFFs describe the behavior of component operations in a way similar to the Unified Modeling Language. Within each RDSEFF, the transitions of the current Markov State to the next states are represented. This way, the allowed sequence of service invocations is controlled by the Markov States themselves. An example can be found in Figure 6.2. The RDSEFF for the *View_Items* Markov State of the generated Behavior Model component *gen_behavior_model3* has a probability branch with two branch transitions, each representing a transition to the next Markov State. The left and the right transitions have a call probability of 92.9 % and 7.1 % respectively. This branch transition specifies the call probability and contains three different actions:

- First, the think time of this transition is modeled as specified in the WESSBAS-DSL using an *InternalAction* either as mean value or as normal distribution with mean and deviation. In our example, the think time is specified with a mean value of one time unit for both transitions.
- Second, the matching system operation of the modeled SUT is called as an *ExternalCallAction*. This call models a request to the system, e.g., clicking a link of a web page. To identify the corresponding system operation we use a name mapping between the name of the system operation and the name of the Markov State. Only the operations of components providing external system calls will be matched with the Markov State names. In the left transition of our example the operation *View_Items* of the system component with the name *_app* is called as it has the same name as the next Markov State.
- Third, the RDSEFF of this Behavior Model component representing the next Markov State is called as *ExternalCallAction*; in the left transition of our example, the *View_Items* state is called again and in the right transition the state *home* is called. In this way, each Behavior Model component calls itself until a RDSEFF without successor is reached. In this case no further call is modeled and the sequence terminates.

Having created the Behavior Model components in the Repository Model, each newly created component is allocated to the System Model and correspondingly to the Allocation Model. A new Usage Model is generated with one probabilistic branch representing the Behavior Mix. For each Behavior Model, a branch transition with the relative frequency as call probability is created. Within this transition the initial Markov State of the Behavior Model is called. Finally, the workload intensity is modeled as closed workload with *(i)* the population representing the number of active sessions and *(ii)* the think time between the end and the start of a new session. The generation of open workloads will be examined in the future.

The Session and the Protocol Layer are not mapped to PCM elements. The Session Layer could be modeled as an additional abstraction layer to the SUT. However, this would increase the complexity of the model and has no impact on the simulation results as the allowed sequences of service invocations are already specified in the representation of the Behavior Models. The Protocol Layer is not used in performance models.

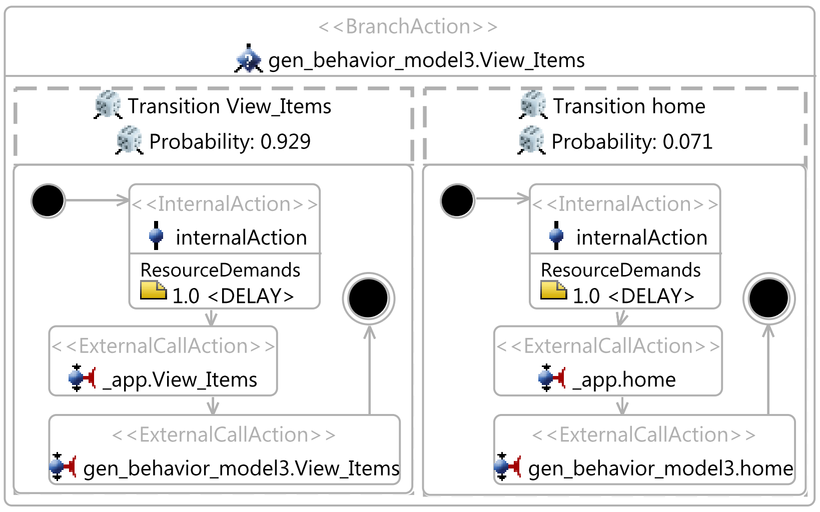


Figure 6.2: *Generated RDSEFF example*

6.4 Evaluation

In this section, the practicality and the prediction accuracy of transformed workload specifications will be examined. First, the SPECjEnterprise2010[18] benchmark is briefly explained and then the accuracy of transformed workload specifications is summarized.

SPECjEnterprise2010 represents a Java EE industry application of an automobile manufacturer whose main users are automobile dealers. This benchmark contains a workload specification and a dataset required for load test executions. In this work, we use the Orders domain of the benchmark as the SUT. The Orders domain represents a web-based e-commerce application. SPECjEnterprise2010 defines three different transaction types and in total 13 different HTTP request types. It enables customers purchasing and selling cars (Purchase), managing their accounts and inventory (Manage), and browsing the catalogue of available cars (Browse).

To evaluate the accuracy of the extracted workload specification we employ the evaluation methodology used in our previous paper (van Hoorn et al., 2014). The number of simulated requests for the different HTTP request types are compared with the originally measured request counts to the SUT. In our previous work we extracted (van Hoorn et al., 2014) WESSBAS-DSL instances from session logs of a SPECjEnterprise2010 benchmark run with 800 users, a duration of ten minutes (600 seconds), and the original benchmark Behavior Mix (25 % Purchase, 50 % Browse, and 25 % Manage). Afterwards, we selected three different WESSBAS-DSL instances for that evaluation; one with two, one with three, and one with four Behavior Models. These instance were extracted using different clustering settings; however the resulting workload characteristics are the same (van Hoorn et al., 2014). To ensure the comparability with workload specifications of performance models, we evaluate whether the same results can be achieved. We generate a PCM instance

[18]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/.

Table 6.3: *Evaluation Results*

		Orig.	2 Behavior Models		3 Behavior Models		4 Behavior Models	
	Request	MRC	SRC	PE%	SRC	PE%	SRC	PE%
1	add to cart	63,761	64,943	1.82%	61,812	3.15%	60,986	4.55%
2	cancel order	632	609	3.78%	661	4.39%	625	1.12%
3	clear cart	6,047	6,178	2.12%	5,927	2.02%	5,846	3.44%
4	defer order	6,782	6,873	1.32%	6,524	3.95%	6,606	2.66%
5	home	59,934	61,146	1.98%	58,747	2.02%	58,744	2.03%
6	inventory	30,596	30,539	0.19%	29,574	3.46%	29,405	4.05%
7	login	61,500	61,156	0.56%	58,747	4.69%	58,745	4.69%
8	logout	59,934	61,146	1.98%	58,747	2.02%	58,744	2.03%
9	purchase cart	8,360	8,388	0.33%	7,976	4.81%	7,836	6.69%
10	remove	3,027	2,986	1.37%	2,876	5.25%	2,949	2.64%
11	sell inventory	66,679	66,131	0.83%	63,185	5.53%	63,914	4.33%
12	shopping cart	9,074	9,164	0.98%	8,803	3.08%	8,795	3.17%
13	view items	498,601	491,812	1.38%	470,392	6.00%	475,000	4.97%
	$\sum$	874,927	871,071	0.44%	833,971	4.91%	838,195	4.38%

representing the SPECjEnterprise2010 system using the approach proposed by Brunnert/ Vögele/Krcmar (2013). Then, each of the three instances is transformed into this PCM instance. This PCM instance is then simulated for 600 time units corresponding to the ten minutes of the original benchmark run.

The results of the Measured Request Count (MRC) and Simulated Request Count (SRC) per HTTP action can be found in Table 6.3. In addition, for each simulation run the relative Prediction Error (PE) compared to the measured data is given. The original workload includes 61,500 sessions and in total 874,927 HTTP requests. The relative counts of the request types are very similar for all predicted workloads. Further, the PE of the request types are at maximum 6.69% for purchase cart with four Behavior Models. Thus, from the server-side perspective the SRCs are representative compared to the MRC. As this was the case for load tests extracted from Wessbas-DSL instances as well (van Hoorn et al., 2014), the suitability of the Wessbas-DSL as intermediate language for workload specifications could be demonstrated.

6.5 Conclusion and Future Work

Several authors describe the need to integrate measurement and model-based approaches to evaluate the performance of software systems (Smith, 2007; Woodside/Franks/Petriu, 2007). To close this gap for workload specification, our Wessbas approach (van Hoorn et al., 2014) is extended. In this paper, the existing Wessbas-DSL is used as an intermediate language for the transformation to workload specifications of architecture-level performance models. The evaluation using Wessbas-DSL instances extracted from the Java EE benchmark SPECjEnterprise2010 demonstrates that representative PCM workload specifications compared to the original workload can be generated. To the best of our knowledge no other approach generates model-based workload specifications for session-based systems from extracted data.

In our future work, we plan to investigate the impact of the extracted workload specification on the measured and on the predicted performance. Furthermore, we plan to evaluate the proposed approach by transforming the Wessbas-DSL into other architecture-level performance models such as the Descartes Modeling Language (Kounev/Brosig/Huber, 2014). The prioritization of load test cases using the generated performance models will be investigated (Vögele et al., 2014). Moreover, we plan to implement the transformation between the Wessbas-DSL and PCM in a bidirectional way. The advantage is when the test cases are analyzed and prioritized within PCM corresponding load test scripts can be generated using the Wessbas-DSL.

Chapter 7

WESSBAS: Extraction of Probabilistic Workload Specifications for Load Testing and Performance Prediction - A Model-Driven Approach for Session-Based Application Systems

Authors	Vögele, Christian[1] (voegele@fortiss.org) van Hoorn, André[2] (andre.van.hoorn@acm.org) Schulz, Eike[3] (esc@informatik.uni-kiel.de) Hasselbring, Wilhelm[3] (wha@informatik.uni-kiel.de) Krcmar, Helmut[4] (krcmar@in.tum.de) [1]fortiss GmbH, Munich, Germany [2]Institute of Software Technology, University of Stuttgart, Stuttgart, Germany [3]Department of Computer Science, Kiel University, Kiel, Germany [4]Chair for Information Systems, Technical University of Munich (TUM), Garching, Germany
Outlet	International Journal on Software and Systems Modeling
Status	Accepted
Individual Contribution	Problem and scope definition, construction of the conceptual approach, prototype development, experiment design, execution and result analysis, paper writing, paper editing

Table 7.1: *Bibliographic details for P4*

Abstract The specification of workloads is required in order to evaluate performance characteristics of application systems using load testing and model-based performance prediction. Defining workload specifications that represent the real workload as accurately as possible is one of the biggest challenges in both areas. To overcome this challenge, this paper presents an approach that aims to automate the extraction and transformation of workload specifications for load testing and model-based performance prediction of session-based application systems. The approach (Wessbas) comprises three main components. First, a system and tool agnostic domain-specific language (DSL) allows the layered modeling of workload specifications of session-based systems. Second, instances of this DSL are automatically extracted from recorded session logs of production systems. Third, these instances are transformed into executable workload specifications of load generation tools and model-based performance evaluation tools. We present transformations to the common load testing tool Apache JMeter and to the Palladio Component Model. Our approach is evaluated using the industry-standard benchmark SPECjEnterprise2010 and the World Cup 1998 access logs. Workload-specific characteristics (e.g., session lengths and arrival rates) and performance characteristics (e.g., response times and CPU utilizations) show that the extracted workloads match the measured workloads with high accuracy.

7.1 Introduction

The specification and execution of workloads is essential for evaluating performance properties of application systems. In order to assess whether non-functional performance requirements of these systems can be met, load testing and model-based performance evaluation approaches are applied (Brunnert et al., 2014; Woodside/Franks/Petriu, 2007). Workload specifications serve as input for load testing to generate synthetic workload to the SUT, i.e., executing a set of customer requests (Goševa-Popstojanova et al., 2006; Krishnamurthy/Rolia/Majumdar, 2006; Menascé et al., 1999). Additionally, several specifications are taken into account in formalisms for model-based performance evaluation approaches, to predict performance properties early in the software development cycle (Becker/Koziolek/Reussner, 2009; Koziolek, 2010; Woodside/Franks/Petriu, 2007).

In session-based application systems, especially Web-based systems, different types of users interact with the system in a sequence of interdependent requests. The complexity of these interactions makes the workload specification a difficult task (Menascé et al., 2000). Thus, the manual creation of these workload specifications is time consuming (Avritzer et al., 2002) and error prone (Shams/Krishnamurthy/Far, 2006). One of the main challenges is to ensure that these specifications are representative compared to the real workload (Draheim et al., 2006). This is a key requirement for both load testing and model-based performance prediction approaches. To ensure that the measured and the predicted performance characteristics of the SUT are comparable, similar workload specifications must be used. However, there is a lack of approaches enabling the common automatic extraction and specification of workloads for both approaches. The extraction and specification of workloads is done separately for each approach and each tool which results in additional specification and maintenance effort. The reasons for this development are that these approaches are not integrated and that workload specifications are defined on different levels of detail. Measurement-based approaches need detailed system-specific information like protocol data, whereas model-based approaches are often specified on a more abstract level.

In response to these challenges, this paper presents our WESSBAS[19] approach for specifying and extracting representative workloads for session-based application systems. We introduce a Domain-specific Language (DSL), called WESSBAS-DSL, which enables the system- and tool-agnostic modeling of these workload specifications. Recorded session logs are used as a basis for the automatic extraction of WESSBAS-DSL instances. Different groups of customers showing similar navigational patterns are identified during the creation of these instances. Additionally, inter-request dependencies (Guards and Actions (GaAs)) among the execution of requests are automatically learned. These dependencies come from the fact that the execution of requests often depends on the result of previous requests. The combination of probabilities and GaAs requires the calculation of conditional probabilities, which are also determined in an automatic way. Finally, protocol information required to generate executable load tests are integrated.

[19]WESSBAS is an acronym for *Workload Extraction and Specification for Session-Based Application Systems*

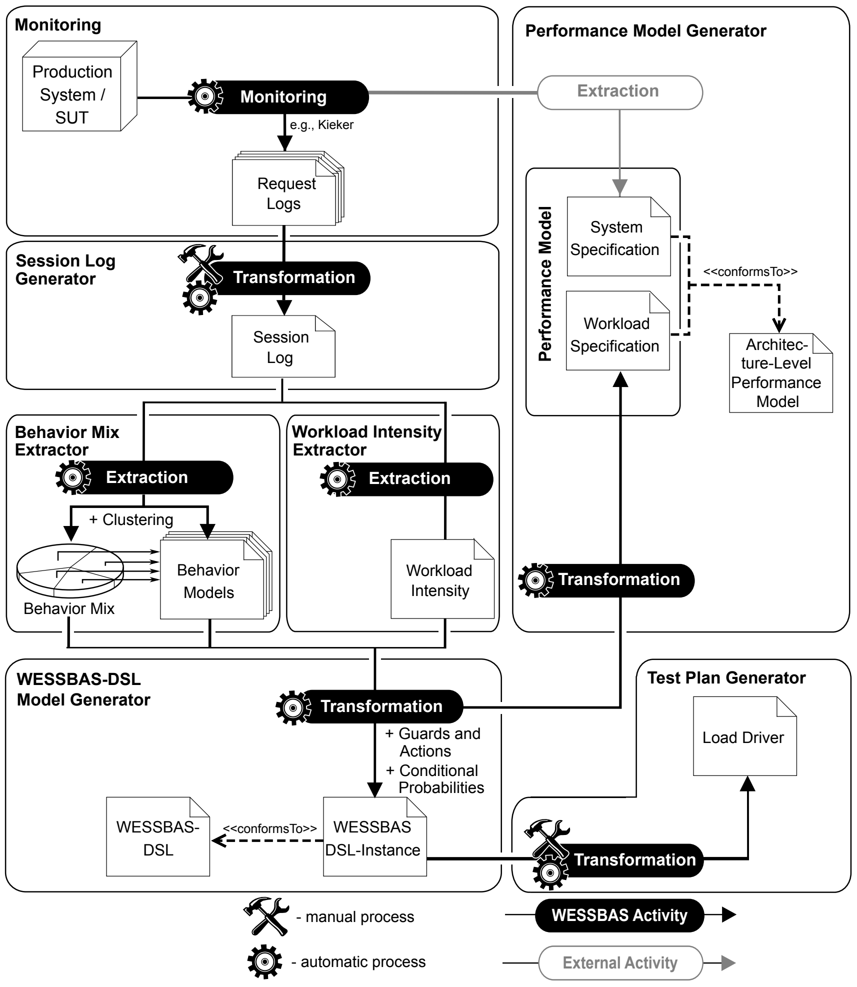


Figure 7.1: *Overview of the* WESSBAS *approach*

As an example, the resulting WESSBAS-DSL instances are then transformed to executable workload specifications for the common load testing tool Apache JMeter, including the Markov4JMeter extension developed in our previous work (van Hoorn/Rohr/Hasselbring, 2008). Furthermore, the DSL instances are transformed into workload specifications of the Palladio Component Model (Becker/Koziolek/Reussner, 2009) representing an architecture-level performance modeling language. We focus on architecture-level performance models as they permit to model system architecture, execution environment, and workload specification separately from each other (Brosig/Huber/Kounev, 2011). Figure 7.1 provides an overview of the WESSBAS approach.

The sharing of a common workload model has several key benefits for both approaches during the system life cycle. During development time for example, system specifications (e.g., UML diagrams) or information from expert interviews are available in order to derive estimates about the workload profile. This knowledge can be encoded in the WESSBAS-DSL and then transformed to a performance model to conduct early performance predictions. Once the application system is running and load tests should be executed, parts of the load test (e.g., user scenarios and workload intensity) can be generated based on the WESSBAS-DSL. Thus, the workload only needs to be created once for both approaches.

In order to evaluate the performance characteristics of new releases, load tests are often conducted on test systems. The workload used during these tests should be comparable to the workload of the production systems. This has the advantage that bottlenecks which also occur in production systems can be found with a higher probability. Thus, extracting the workload from the production system and transforming it to load tests and workload specifications of performance models comes with several benefits. The first benefit is that the effort to create and specify load tests and performance models is reduced. Because production workloads can change over time, the latest workload can be extracted again in an easy way to reduce maintenance effort. The second benefit is that the integration of software development and operations (DevOps) is supported (Brunnert et al., 2015). The extracted workload during operations can be used for measurement-based (Bulej/Kalibera/Tůma, 2005) and model-based (Brunnert/Krcmar, 2014) continuous integration approaches to detect performance regressions during development.

The Wessbas approach is also useful to validate performance models. To validate the correctness of a performance model, the simulation results are compared with measurement results derived from the system. For the comparability of these results, it must be ensured that the simulation results and the measurements are derived by applying the same workload specification.

To summarize, the contribution of this paper is our Wessbas approach for automatic extraction of probabilistic workload specifications of session-based application systems comprising the following elements:

1. A DSL for modeling session-based probabilistic workload specifications.
2. An automatic extraction of DSL instances from recorded system logs including the clustering of navigational patterns.
3. Transformations from DSL instances to executable JMeter Test Plans and to workload specifications of the PCM.
4. Tool support for this approach.

To the best of our knowledge Wessbas is the first approach to enable the process from runtime data to executable load tests and performance prediction approaches. The tool support serves as an extensible implementation of the approach, including the DSL, the extraction, as well as a proof-of-concept transformation from the DSL to JMeter Test Plans and workload specifications of PCM. The developed tools[20], and the models and results of this paper (Vögele et al., 2016) are publicly available online.

This paper builds on our previous work (van Hoorn/Rohr/Hasselbring, 2008; van Hoorn et al., 2014; Vögele/van Hoorn/Krcmar, 2015) on the extraction and specification of workload specifications and contains the following major improvements and extensions:

1. Tool support for the transformation of arbitrary system logs to the required session log format.

[20]Wessbas tool support: https://github.com/Wessbas

2. Automatic learning of GaAs and the calculation of conditional probabilities.
3. Generation of executable load tests (this includes the extraction and integration of protocol information).
4. Comprehensive evaluation of workload characteristics (e.g., session lengths and action counts) and performance characteristics (e.g., response times and CPU utilizations) against the industry-standard benchmark SPECjEnterprise2010 and the World Cup 1998 access logs.

The remainder of this paper is structured as follows: Section 7.2 provides an overview of related work. In Section 7.3, the used workload formalism required to understand the paper is introduced. The extraction of Wessbas-DSL instances is presented in Section 7.4. The description of the JMeter Test Plan generation process in Section 7.5 is followed by the illustration of the transformation of workload specifications to performance models in Section 7.6. The evaluation of the proposed approach can be found in Section 7.7. Finally, Section 7.8 details conclusions of our work and presents suggestions for future work.

7.2 Related Work

Workload specification (also referred to as workload characterization) is defined by the process of first analyzing key characteristics of user interactions (this includes other systems as well) with an application system and modeling these characteristics into a workload model (Jain, 1991; Calzarossa/Massari/Tessera, 2016). The key workload characteristics of session-based systems can be divided into intra-session and inter-session metrics (Goševa-Popstojanova et al., 2006). Intra-session metrics characterize single sessions and include the session length, number of requests per session, and think times between the executions of the requests. They also describe the behavior of the user as a sequence of executed requests. In contrast, the inter-session metrics characterize the number of sessions per user and the number of active sessions over time (also referred to as workload intensity).

We group the related work on workload characterization into user behavior modeling and workload intensity. Related work on the extraction of workloads and workload modeling for performance models are also introduced.

7.2.1 User Behavior Modeling

User behavior is either specified script-based or it is specified using analytical models. Scripts representing single user scenarios with a fixed sequence of user requests are executed by a number of concurrent load generator threads. These scripts are quite easy to record and execute. However, they provide little opportunity to vary workload characteristics, such as different navigational patterns, and therefore are often not as representative as the real workload (Draheim et al., 2006; Lutteroth/Weber, 2008). Furthermore, as examined by Rodrigues et al. (2014), the effort to generate capture and replay scripts is higher with these scripts than the effort using analytical models with increasing complexity of the software system.

In order to model the user behavior in a more representative way, analytical models were introduced. A popular way to model user behavior, especially user behavior related to Web sites, are Markov Chains (Zhao/Tian, 2003). An approach similar to our approach was proposed by Menascé (2002) and Menascé et al. (1999). These authors extract so-called Customer Behavior Model Graphs (CBMGs) from HTTP server logs, which are based on Markov Chains. They apply K-means clustering to identify CBMGs for similar types of users. In contrast, in our work, an advancement of the K-means algorithm, called X-means is applied. Another approach using the CBMGs to generate representative workloads is presented by Ruffo et al. (2004). First, CBMGs are automatically extracted from Web application log files, and then representative user behavior traces are generated from these CBMGs. Based on these traces, a modified version of the performance testing tool httperf (Mosberger/Jin, 1998) is used to generate the Web traffic. In both approaches, Markov States represent the user interaction with the system. Transitions between these states are annotated with user think times and probabilities.

Zhao/Tian (2003) have proven that these models can be used for workload generation. However, one of their limitations is that they are not able to handle the aforementioned inter-request dependencies. In an inter-request dependency an item can only be removed from a shopping cart if items have already been added to that shopping cart. To overcome these limitations, Shams/Krishnamurthy/Far (2006) proposed a workload modeling formalism based on EFSMs. EFSMs allow a description of valid sequences of user requests within a session. In contrast to the approaches based on Markov Chains, the transitions are labeled with GaAs based on predefined state variables and not with probabilities. Valid sessions are obtained by simulating the EFSMs. Additionally, inter- and intra-session characteristics, such as think times and session length distributions and a workload mix defining the relative frequency of request types, can be specified. Our work combines the modeling approaches based on CBMGs and EFSMs (van Hoorn/Rohr/Hasselbring, 2008). Thus, probabilistic user behavior modeling is enabled while ensuring that valid sequences of user requests are generated.

Other approaches exist that use analytical formalisms to define workload models. Examples include stochastic form-oriented models (Draheim et al., 2006; Lutteroth/Weber, 2008), probabilistic timed automata (Abbors et al., 2012) or context-based sequential action models (Junzan/Bo/Shanping, 2014).

One limitation of the proposed approaches is the need for manual specification of GaAs as they are not extracted automatically from log files. To overcome this challenge, several approaches exist for extracting behavior models from system logs in the form of Finite State Machines (FSMs) (Beschastnikh et al., 2011; Ohmann et al., 2014) or EFSMs (Walkinshaw/Taylor/Derrick, 2013). We extend the work of Beschastnikh et al. (2011) to automatically derive the GaAs for the application model, based on temporal invariants mined from the session log files (explained in detail in Section 7.4.4).

7.2.2 Workload Intensity

An approach focusing on the definition of workload intensities can be found in the work by v. Kistowski/Herbst/Kounev (2014). Their Load Intensity Modeling Tool (LIMBO) approach allows a DSL-based definition and extraction of variable and dynamic load profiles and workload scenarios over time including seasonal patterns, long term trends, bursts, and a certain degree of noise. The work proposed by Herbst et al. (2013) offers an approach to forecast workload intensities in which suitable forecasting methods are chosen, based on a decision tree and feedback cycles to improve the forecast accuracy. Wessbas focuses on the specification of the behavior of users and offers basic support for modeling workload intensities (see Section 7.4.3).

7.2.3 Workload Extraction

The extraction of workloads is usually based on request logs (Menascé et al., 1999; van Hoorn et al., 2014), design specifications such as UML diagrams (da Silveira et al., 2011), or expert knowledge (Barber, 2004).

An approach similar to ours, introducing an abstract intermediate model that defines workload specification independent from the used technology, is defined by Costa et al. (2012). These authors focus on the separation of technology details from the test scenarios. UML diagrams are used as input for creating abstract intermediate models. These model instances are transformed to the load test tools Visual Studio Load Test and HP Loadrunner. As UML diagrams are often not available or not detailed enough, we propose to extract the intermediate language Wessbas-DSL from log files taking also protocol information and inter-request dependencies into account.

7.2.4 Workload Modeling for Performance Models

Architecture-level performance models (Becker/Koziolek/Reussner, 2009; Kounev/Brosig/Huber, 2014; Object Management Group, Inc., 2013) allow the modeling of usage behavior, e.g., based on UML formalisms. These models also allow the specification of the Workload Intensity. The effort to create performance models in a manual way significantly reduces any benefit to be gained. Thus, approaches for the automatic performance model generation have been proposed (Brosig/Huber/Kounev, 2011; Brunnert/Vögele/Krcmar, 2013). These approaches focus on the automatic extraction of the system-specific details of the SUT, like the system components, the relationship between the components, the component allocations, and the resource demands. However, the workload specifications must still be modeled manually, which requires a lot of effort on the part of the performance expert.

To reduce the complexity of generating different kinds of analytical performance models from architecture-level performance models, several intermediate languages such as PUMA (Woodside et al., 2005) or Klaper (Ciancone et al., 2011) were introduced. These approaches only focus on model-based performance evaluation and do not support the definition of workload specifications for session-based application systems.

An approach that combines model-based performance testing with load testing for Web-based systems is introduced by Barna/Litoiu/Ghanbari (2011b) and Barna/Litoiu/Ghanbari (2011a). In their approach, the SUT is modeled as a two-layer queuing model. Then, workload mixes and workload intensities are derived from the model under which software and hardware bottlenecks are saturated. Finally, the test cases are derived and executed on the SUT. The model is automatically tuned, based on feedback loops from the SUT. In contrast to WESSBAS, the user behavior is aggregated on transactional level, e.g. a buy transaction, and not on single user interactions.

7.3 Workload Specification

An overview of the workload specification formalism required to understand the paper is given in Section 7.3.1. The WESSBAS-DSL, which is based on this specification, is introduced in Section 7.3.2.

7.3.1 Workload Specification Formalism

The approach described in this paper builds on our previous work on generating probabilistic and intensity-varying workloads for session-based systems (Schulz et al., 2014; van Hoorn/Rohr/Hasselbring, 2008); particularly, the workload modeling formalism that extends the work by Menascé et al. (1999) and Krishnamurthy/Rolia/Majumdar (2006). This section introduces the concepts needed to support the remainder of this paper.

The workload specification formalism (*Workload Model*) consists of the following components, which are detailed below and illustrated in Figure 7.2:

- An *Application Model*, specifying allowed sequences of service invocations and SUT-specific details for generating valid requests.
- A set of *Behavior Models*, each providing a probabilistic representation of user sessions in terms of invoked services and think times between subsequent invocations as Markov chains.
- A *Behavior Mix*, specified as probabilities for the individual Behavior Models to occur during workload generation.
- A *Workload Intensity* that includes a function which specifies the (possibly varying) number of concurrent users during the workload generation execution.

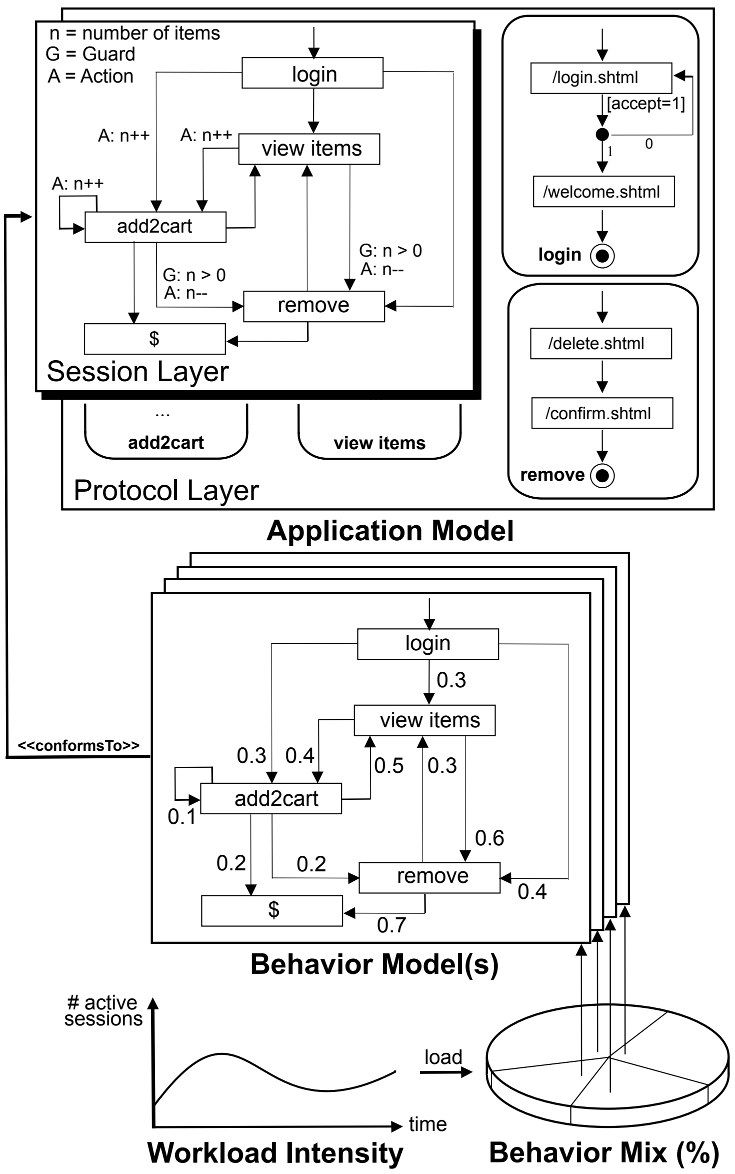


Figure 7.2: *Exemplary Workload Model (without think times in the Behavior Models)*

7.3.1.1 Application Model

The Application Model is a two-layered hierarchical EFSM, consisting of a *Session Layer* and a *Protocol Layer*. Inspired by the work of Krishnamurthy/Rolia/Majumdar (2006) and Shams/Krishnamurthy/Far (2006), the Session Layer is an EFSM in which states refer to system-provided services and allowed transitions among these states/services. These transitions are possibly labeled with GaAs. The EFSM is defined as a 6-tuple (Kalaji/Hierons/Swift, 2009) $(S \cup \{\$\}, s_0, V, I, O, T)$ where $S \cup \{\$\}$ specifies the set of states contained in the application model and $s_0 \in S$ the initial state; V is a finite set of variables; I is a set of input symbols; O is the set of output symbols; and T is a set of possible transitions.

A directed transition $t \in T$ is represented by the 5-tuple $(s_s, i, g_{s,e}, a_{s,e}, s_e)$ in which s_s is the source state of t; i is the input where $i \in I$ and i may have associated input parameters; $g_{s,e}$ is the guard condition which validates if the transition can be executed according to the current variable values; $a_{s,e}$ defines a function on the variable values called action statement, in case the related application transition fires; and finally, s_e is the target state of t.

For each state of the Session Layer, the Protocol Layer contains an associated EFSM, (possibly labeled with GaAs as well) that models the sequence of protocol-level requests to be executed when the Session Layer state is executed.

7.3.1.2 Behavior Models

A Workload Model includes one or more Behavior Models. Each Behavior Model defines probabilistic behavior and think times. Furthermore, each Behavior Model is specified as a Markov Chain and roughly corresponds to the CBMGs introduced by Menascé et al. (1999). Each Behavior Model $\mathcal{B}$ is defined as the tuple $(MS \cup \{\$\}, ms_0, P, TT, f_{tt}, BT)$. MS specifies the set of Markov States contained in the Behavior Model with initial Markov State $ms_0 \in MS$ and exit state \$. Each Markov State is associated with exactly one Application State of the Application Model. $P = [p_{s,e}]$ is a $n \times n$-matrix of transition probabilities, with $n = |MS \cup \{\$\}|$. Think times are specified as an $n \times n$-matrix $TT = [tt_{s,e}]$, with $n = |MS \cup \{\$\}|$. The distribution function f_{tt} specifies the probability distribution of the think times. For instance, the think times may be specified using a Gaussian distribution. BT is a set of transitions in the Behavior Model.

A transition in a Behavior Model $bt \in BT$ is represented by the 4-tuple $(ms_s, p_{s,e}, tt_{s,e}, ms_e)$ in which ms_s is the source Markov State of bt. A matrix entry $p_{s,e} \in P$ defines the probability for a transition from Markov State ms_s to Markov State ms_e. When the probability of $p_{s,e} = 0$ then the transition cannot be executed. A matrix entry $tt_{s,e} \in TT$ defines the think time for a transition from state ms_s to state ms_e. Finally ms_e is the end Markov State of the transition bt.

The Behavior Models can also be defined as *absolute* Behavior Models $\mathcal{AB}$ represented as the tuple $(MS \cup \{\$\}, ms_0, A, STT, BT, f_{tt})$. Then, the matrix $A = [a_{s,e}]$ specifies a $n \times n$-matrix of absolute transition counts. Furthermore, the matrix $STT = [stt_{s,e}]$ represents a $n \times n$-matrix of accumulated think times of the transitions.

The advantage of separating the Application Model and the Behavior Models is that the Protocol Layer and the GaAs for the transitions of the Session Layer have to be specified only once. Otherwise, this information would need to be added to each Behavior Model.

7.3.1.3 Behavior Mix

The Behavior Mix is a set $\{(\mathcal{B}_0, r_0), \ldots, (\mathcal{B}_{m-1}, r_{m-1})\}$, which assigns a relative frequency r_i to the Behavior Model $\mathcal{B}_i$. A tuple $(\mathcal{B}_i, r_i)$ indicates that sessions which correspond to the Behavior Model $\mathcal{B}_i$ are generated with a relative frequency of $r_i \in [0, 1]$. That is, each r_i denotes a probability value and the sum of all values must be 1, corresponding to 100%.

7.3.1.4 Workload Intensity

The Workload Intensity for an experiment is specified in terms of the number of active sessions, i.e., the number of virtual users being simulated concurrently. A generated session is considered active while the workload generator submits requests based on the corresponding probabilistic session model (the exit state of the Behavior Model has not been reached). A function $n : \mathbb{R}_{\geq 0} \to \mathbb{N}$ specifies this number $n(t)$ of active sessions relative to the elapsed experiment time t. Particularly, this allows for generating a varying workload intensity profile, e.g., based on measured workload data.

7.3.1.5 Workload Generation Process

During the workload generation process for a SUT, the model is used as follows (see Figure 7.2):

The Workload Intensity specifies the number of active sessions. For each newly created session, the Behavior Mix determines the user type to be emulated next by selecting the corresponding Behavior Model $\mathcal{B}_i$ based on the assigned relative frequencies r_i. In the selected Behavior Model, a probabilistic sequence of services is generated according to the transition probabilities specified in the related Markov Chain. Furthermore, the GaAs of the Session Layer are taken into account in order to generate valid sequences.

Assume that the Behavior Model $\mathcal{B}_i$ is currently in the Markov state *view_items* (*view_items* $\in MS$) and the current variable n ($n \in V$) has the value one. First, based on the transitions T modeled in the Session Layer from application state *view_items* (*view_items* $\in S$) to the following states, it is validated which guard conditions are satisfied—in the example, the transition from *view_items* to *add2Cart* and *remove*. As the number of items n is one, the guard $G : n > 0$ to transition *remove* is true. The transition to *add2Cart* has no guard and can therefore always be executed. Second, based on the probabilities specified in the matrix P, the next transition is chosen—40% of cases to *add2Cart* and 60% to *remove*. Third, the action(s) on the variable value(s) will be executed. When *remove* is chosen, the value of n is decreased by one; when *add2Cart* is chosen, the value is increased by one. Finally, the think time is taken from the think time matrix TT for this transition. After the think time has elapsed, the Behavior Model moves to the next state and the service is executed according to the specified EFSM of the Protocol Layer.

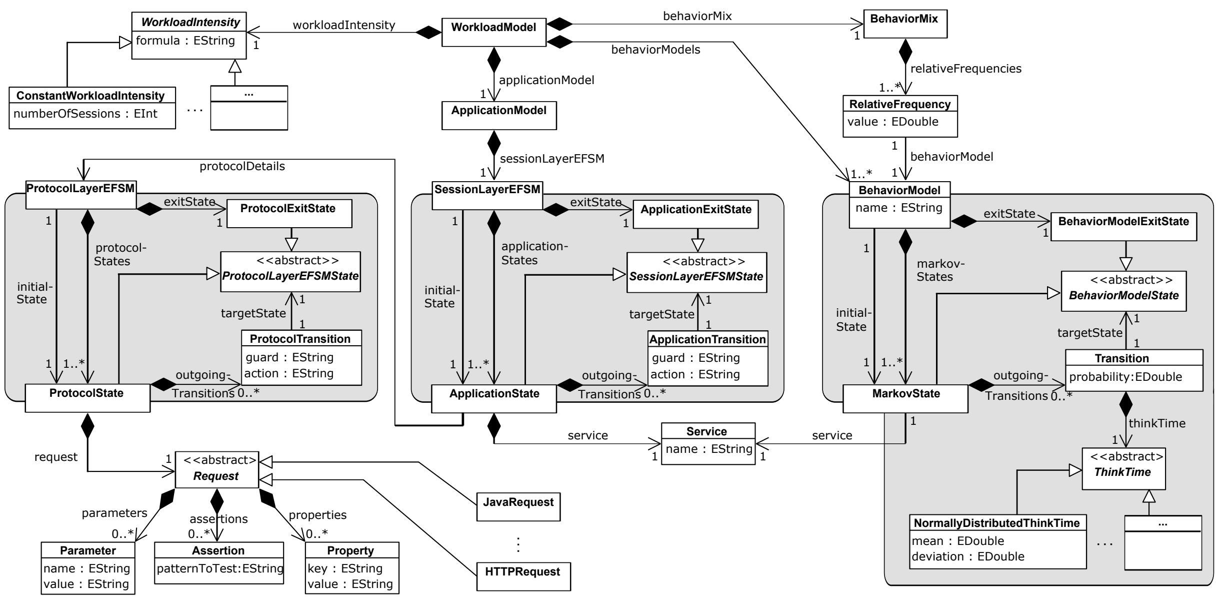


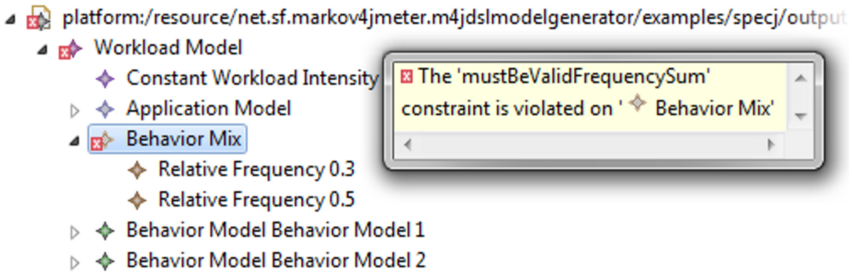
Figure 7.3: WESSBAS-DSL *classes and relationships*

7.3.2 WESSBAS-DSL

The WESSBAS-DSL follows the workload modeling formalism introduced in the previous section and denotes a language for expressing such models. In our approach, the WESSBAS-DSL is used as an intermediate language between the construction of SUT-specific but tool-agnostic workload models on the one side, and the generation of corresponding inputs to load testing tools and performance models on the other side. WESSBAS is implemented as an Ecore-based meta-model using the benefits and tool support of the Eclipse Modeling Framework (EMF) (Steinberg et al., 2009). The meta-model is enriched with constraints (specified in the common Object Constraint Language (OCL)), for checking the validity of model instances. The DSL structure offers a high degree of flexibility and extensibility. The remainder of this section introduces the core concepts.

An overview of the WESSBAS-DSL classes and relationships as a language for the introduced workload specification is presented in Figure 7.3. The parent class Workload Model consists of the Application Model, the Workload Intensity, the Behavior Mix, and Behavior Models.

The representation of the Application Model corresponds to the two-layered structure of that component, including EFSMs for both the Session Layer and the Protocol Layer. States of the Session Layer EFSM, shortly referred to as Application States, are associated with services and with a Protocol Layer EFSM. Services are use cases, e.g., signing on to a system or adding an item to the shopping cart (see Figure 7.2). The states of the Protocol Layer EFSM are associated with specific requests, which might be of type HTTP, Java, BeanShell, SOAP, etc.; the set of currently supported request types can be extended easily by deriving additional subclasses from the common base class. Mention should be made of the difference between properties and parameters of a request: properties correspond to the information that is required for sending a request (e.g., domain, path, or port number


platform:/resource/net.sf.markov4jmeter.m4jdslmodelgenerator/examples/specj/outpu
Workload Model
Constant Workload Intensity
Application Model
Behavior Mix
Relative Frequency 0.3
Relative Frequency 0.5
Behavior Model Behavior Model 1
Behavior Model Behavior Model 2

The 'mustBeValidFrequencySum' constraint is violated on ' Behavior Mix'

Figure 7.4: *Example of a* WESSBAS-DSL *model with a violated constraint (no Behavior Mix frequency sum of 1.0), opened in an EMF Form Editor.*

of a targeted server); parameters denote values to be sent with the request (e.g., input data for a Web form). The transitions in the Session Layer and Protocol Layer EFSMs can be labeled with GaAs. An example can be seen in Figure 7.2. The user action *remove* can only be called when the number of items is greater than zero.

Behavior Models are modeled as Markov Chain, with each (Markov) State also being associated with one service. Thus, each Markov State is assigned exactly to one Application State of the Session Layer. Transitions of the Behavior Models are labeled with probabilities and think times. Currently supported think times are of type Gaussian, that is, they follow a normal distribution, indicating mean and (standard) deviation values as parameters. Other think time implementations can be integrated easily by using the abstract class *ThinkTime*. Exit states are modeled explicitly, and are—in contrast to Markov States—not associated with services as they do not provide a service to the user. Each Behavior Model is associated with a relative frequency to define the Behavior Mix and it is stored as a double value in a dedicated class. These frequencies are contained in the Behavior Mix. The Workload Intensity is stored as a string attribute in the dedicated class *WorkloadIntensity* that also serves as a base class for all types of workload intensities. The Workload Intensity can be specified as a formula to define varying workloads or as a fixed number for constant workloads.

Even though the WESSBAS-DSL is independent of specific performance evaluation tools, it includes all core information required for generating workload specifications that build on the described workload modeling formalism. The implementation of the WESSBAS-DSL as an Ecore meta-model offers the benefits of EMF tools such as EMF Form Editors or serialization support. In particular, WESSBAS-DSL instances can be viewed, validated, and modified in an editor, before being passed as input to any transformation process. For the validation, OCL constraints have been defined with the use of EMF *OCLinEcore* tools. An example of a violated constraint is shown in Figure 7.4.

These constraints ensure that, for example, attributes such as probabilities are valid, state names are unique, and transitions of Behavior Models correspond to those of the Session Layer EFSM. An overview of all implemented OCL constraints is given in Schulz (2014).

7.4 Extracting WESSBAS-DSL Instances

As the manual creation of workload models requires much effort, this section presents the process for extracting WESSBAS-DSL instances automatically based on recorded system logs.

The remainder of this section details the six-step procedure to obtain a WESSBAS-DSL instance, comprising *(i.)* the extraction of a session log from the production system (Section 7.4.1), *(ii.)* the clustering-based extraction of the Behavior Mix (Section 7.4.2), *(iii.)* the extraction of the Workload Intensity (Section 7.4.3), *(iv.)* learning of GaAs (Section 7.4.4), *(v.)* calculation of conditional probabilities (Section 7.4.5), and *(vi.)* the generation of a complete WESSBAS-DSL instance from the Behavior Mix (Section 7.4.6).

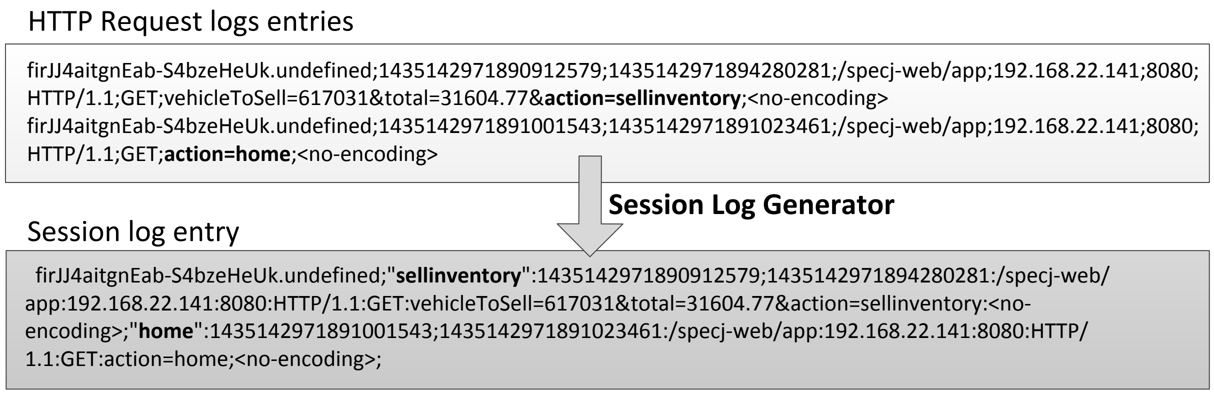
7.4.1 Monitoring and Session Log Generation

The extraction of WESSBAS-DSL instances is based on a so-called *session log* obtained from raw session information, recorded from a running application system. Raw session information is usually provided by *request logs* generated by monitoring facilities, comprising the associated requests to system-provided services with a session identifier and timestamps for the request and completion time. A typical example is the HTTP request log provided by common web servers (Menascé et al., 1999),or tracing information obtained from application-level monitoring tools (van Hoorn/Waller/Hasselbring, 2012).

For each HTTP request, the following information is mandatory to create WESSBAS-DSL instances (example see Figure 7.5): “session identifier”, “request start time”, “request end time”, and “request URL”. In order to further create the Protocol Layer for the generation of executable load tests, the following request information are required as well: “host IP”, “port”,“method” (GET/POST), “protocol”, “parameter with parameter values”, and “encoding”. These Protocol Layer information is not required for the creation of PCM models.

The request information will be transformed to a session log, which can be processed in the next step by the Behavior Mix Extractor. During the transformation, the requests are grouped by the specified session identifier (e.g., session identifier, client IP, or user ID), giving access to the sequence and timing information of subsequent service requests within a session (see Figure 7.5). In each line, the leading number denotes a unique session identifier followed by the sequence of extracted services. A service execution is identified by its assigned name in quotes followed by its start time and end time, and the protocol information.

A service defines a specific user interaction with the system, like clicking a link on a Web page. Each service will later be translated to a service of the WESSBAS-DSL (see Figure 7.3). Thus, the services also represent the states in the Session Layer and the Markov States of the Behavior Models. As the identification of the services is dependent on the respective application, its translation must be specified manually. The user can



```
firJJ4aitgnEab-S4bzeHeUk.undefined;1435142971890912579;1435142971894280281;/specj-web/app;192.168.22.141;8080;
HTTP/1.1;GET;vehicleToSell=617031&total=31604.77&action=sellinventory;<no-encoding>
firJJ4aitgnEab-S4bzeHeUk.undefined;1435142971891001543;1435142971891023461;/specj-web/app;192.168.22.141;8080;
HTTP/1.1;GET;action=home;<no-encoding>
```

```
firJJ4aitgnEab-S4bzeHeUk.undefined;"sellinventory":1435142971890912579;1435142971894280281:/specj-web/
app:192.168.22.141:8080:HTTP/1.1:GET:vehicleToSell=617031&total=31604.77&action=sellinventory:<no-
encoding>;"home":1435142971891001543;1435142971891023461:/specj-web/app:192.168.22.141:8080:HTTP/
1.1:GET:action=home;<no-encoding>;
```

Figure 7.5: *Example HTTP log (recorded with Kieker) and resulting session log*

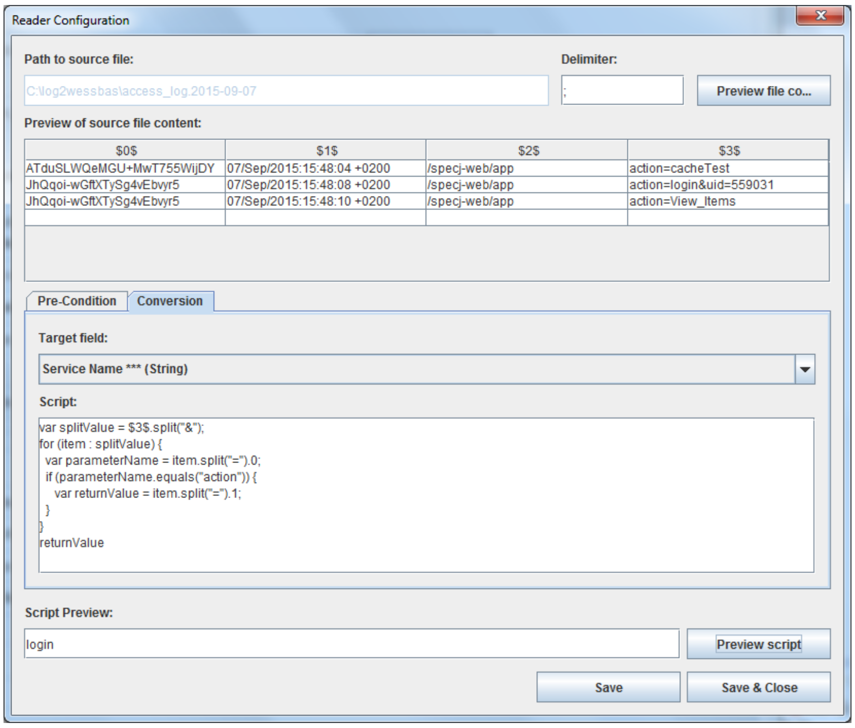
specify that each distinct requested URL is a service. However, there are applications where different URLs represent the same service. In other applications, the same base URL is used and the services can only be distinguished based on submitted parameters. The translation can be defined using the URLs, the parameter names, or corresponding parameter values of the request. For instance, a HTTP request parameter called *action* has the values *sellinventory* or *home* (see Figure 7.5); the values of this parameter can then be used to distinguish the two services.

As each monitoring facility can generate different log formats (e.g., different delimiters or date and time formats), the WESSBAS approach provides tool support for transforming the raw logs to the session log, named the Session Log Generator. The tool enables the user to specify the input raw logs files and then to manually define how the required session data is extracted from the raw logs to the session log. Our approach is independent from specific monitoring solutions, so using the Session Log Generator is advantageous. Furthermore, these rules must only be configured once and can then be reused each time new session logs in the same format are available.

With this tool, the translation of the request data to the service names can be defined. We integrated the Java Expression Language (JEXL)[21] to enable the user to define these translation rules. An example of how to use the Session Log Generator to define a translation rule can be found in Figure 7.6. A HTTP access log is read in by the Session Log Generator. This log consists of session identifiers, time stamps, request URLs, and request parameters for each request. To specify the service name of the resulting session log, the value of the parameter "action" will be identified. For the second row of the log the value "login" is identified.

As users can exit their sessions at any time or can have long inactivity periods, the determination of service requests belonging to a session is required as well. The determination depends on the available session identifier within the raw logs. In case an identifier generated by a Web server is used, no further information has to be specified. Because a session time out is configured in each web server, users with a long inactivity period are automatically assigned to a new session identifier. If client IP addresses or user IDs are used, a threshold for the maximum allowed time between two user requests can be spec-

[21]https://commons.apache.org/proper/commons-jexl/


Reader Configuration

Path to source file: C:\log2wessbas\access_log.2015-09-07

Delimiter: ;

Preview of source file content:

0	1	2	3
ATduSLWQeMGU+MwT755WijDY	07/Sep/2015:15:48:04 +0200	/specj-web/app	action=cacheTest
JhQqoi-wGftXTySg4vEbvyr5	07/Sep/2015:15:48:08 +0200	/specj-web/app	action=login&uid=559031
JhQqoi-wGftXTySg4vEbvyr5	07/Sep/2015:15:48:10 +0200	/specj-web/app	action=View_Items

Target field: Service Name *** (String)

Script:

```
var splitValue = $3$.split("&");
for (item : splitValue) {
 var parameterName = item.split("=").0;
 if (parameterName.equals("action")) {
   var returnValue = item.split("=").1;
 }
}
returnValue
```

Script Preview: login

Figure 7.6: *Exemplary translation rule in the Session Log Generator*

ified (Menascé et al., 1999). In case this threshold is exceeded, the current sequence of service requests is split. Then, the following requests are considered to belong to a new session with a unique session identifier. This threshold can also be defined in the Session Log Generator.

7.4.2 Clustering-Based Behavior Mix Extraction

The Behavior Mix Extractor extracts the Behavior Mix and the corresponding Behavior Models based on the created session log. The Behavior Mix is determined by identifying different groups of customers with similar navigational patterns. As proposed by Menascé et al. (1999), clustering methods can be used to support this task. The identification of different customer groups has several advantages. For example, the system can be optimized upon these navigational patterns. Furthermore, the impact of different Behavior Mixes on the performance can be evaluated, e.g., investigating the performance impact of an increased fraction of a customer group. Lastly, the goal of the clustering is to obtain a relatively small number of clusters to reduce the complexity and to increase the comprehensibility of the resulting Behavior Mix.

In this paper, we focus on clustering with the centroid-based X-means algorithm, which is an improved version of the well-known K-means algorithm (Pelleg et al., 2000). The advantage of X-means over K-means is that it is not mandatory to specify the number of clusters K in advance by the user. The user provides a minimum and a maximum number of resulting clusters and the algorithm determines how many clusters are best suited. The evaluation of K-means clustering is very costly because the results of the K-means must repeatedly be evaluated with different numbers of K (Berkhin, 2006). Furthermore, the

X-means algorithm scales better and the risk of finding local minima is lower. The X-means clustering algorithm is integrated into our proposed approach using the data mining framework Weka (Hall et al., 2009). Other algorithms can be integrated accordingly.

The Behavior Mix Extractor reads the session log file and first transforms each session entry into an *absolute* Behavior Model. This model is composed of a $n \times n$-matrix A defining the transition counts and of a $n \times n$-matrix STT defining the accumulated think times. We remind that we use the workload specification introduced in Section 7.3.1.2. Because Weka cannot handle matrices as clustering input, each matrix A of each *absolute* Behavior Model is transformed into a vector $V = v_1, ..., v_n$ by mapping each value $a_{s,e}$ to a value of the vector by:

$$a_{s,e} \rightarrow v_{(e+((s-1)\cdot n))} \tag{7.1}$$

Think times are not part of the clustering as they have no impact on the navigational patterns. As future work, it could be of interest that the Behavior Models are also clustered using the think times. During the clustering in the first step, a central vector V', called centroid, is determined randomly for each cluster. Each centroid represents a cluster and is the mean of the instances (in our case, sessions represented as transition counts matrices of the *absolute* Behavior Models) $AB_m = [A_m], m = 1, ..., M$ in that cluster. Then, the clustering algorithm iterates several times over the dataset and assigns instances to the nearest cluster centroid until no instance no longer changes the cluster. After each iteration, each centroid V' is recalculated by:

$$v'_i = \frac{\sum\limits_{i=1}^{m} v_i}{m} \tag{7.2}$$

The distance between the instances is calculated using the Euclidean distance metric. During the calculation of a distance, the attributes of the instances (represented as a vector) can be normalized to a value between zero and one. Without data normalization, the attributes with the highest variance drive the clustering. That means, in our case, high transition counts have a high influence on the clustering. In order to figure out the best settings, both the normalized and the non-normalized Euclidean distances will be evaluated in Section 7.7.4.1. Other distance metrics like Manhattan distance or Chebyshev distance could be used as well.

The *relative* Behavior Models are calculated as proposed by Menascé et al. (1999). First, each centroid vector V' is transformed back to a matrix A'. Then the corresponding think time matrix STT' is calculated by accumulating the think times of the single *absolute* Behavior Model instances AB_m within each cluster:

$$stt'_{s,e} = \sum_{i=1}^{m} stt_{i_{s,e}} \tag{7.3}$$

As a result, the centroids represent the *absolute* Behavior Model of the corresponding cluster. Afterwards, these absolute transition count matrices A' are transformed to relative $n \times n$-matrices P, defining the transition probabilities. Furthermore, the matrix STT' will be transformed to the matrix TT representing the mean think time per transition.

$$p_{s,e} = \frac{a'_{s,e}}{\sum\limits_{i=1}^{n} a'_{s,i}} \tag{7.4}$$

$$tt_{s,e} = \frac{stt'_{s,e}}{a'_{s,e}} \tag{7.5}$$

Finally, the relative frequency r of each Behavior Model is calculated by dividing the number of instances m within each cluster by the overall number of session instances in the session log.

7.4.3 Workload Intensity Extraction

The Workload Intensity is automatically analyzed based on the session log and included into the resulting model. The maximum and the average number of concurrent sessions are determined. The user can configure which of these values should be included into the WESSBAS instance. During test execution, this number represents the number of concurrent threads, each starting a new session after the previous session is finished (Barford/Crovella, 1998). Testing the SUT with peak or average loads is sufficient for many application systems.

In order to be able to integrate varying load intensities, for example to test dynamic resource allocations used in virtualized data centers and cloud computing, the LIMBO approach proposed by v. Kistowski/Herbst/Kounev (2014) could be integrated in WESSBAS. The LIMBO approach generates a load intensity model from log files describing the session arrival rate over time using mathematical functions. This meta-model is also implemented using EMF tools and can be combined with the WESSBAS-DSL workload intensity definition. Furthermore, there are already available extensions for JMeter[22] and PCM (Lehrig/Becker, 2014).

7.4.4 Automatic Learning of Guards and Actions

As stated in the previous section, transitions of the Application Layer are optionally labeled with GaAs. As workload specifications might generate invalid paths using solely probabilistic transitions, an important task to be considered is the identification of GaAs. This leads to the fact that errors might occur or that less demand is generated on the

[22]http://se.informatik.uni-wuerzburg.de/tools/limbo/

system resources during load testing, as the user behavior is incorrectly represented. The generated load on the system could be incorrect and performance characteristics, such as CPU utilization or response times, might be different than using correct user behavior (Shams/Krishnamurthy/Far, 2006).

We build on the approach introduced by Beschastnikh et al. (2011), called Synoptic, to learn the GaAs automatically. Beschastnikh et al. define three different kinds of so-called temporal invariants representing relationships between event types (in our case service invocations) that are true over all input traces. Their approach was easily integrated into our WESSBAS framework as the temporal invariants can also be extracted from the same session log file. Three different kinds of invariants are defined and extracted (Beschastnikh et al., 2011):

- *a* Always Followed by *b* (AFby): Whenever the event type *a* appears, the event type *b* always appears later in the same session trace.
- *a* Never Followed by *b* (NFby): Whenever the event type *a* appears, the event type *b* never appears later in the same session trace.
- *a* Always Precedes *b* (AP): Whenever the event type *b* appears, the event type *a* always appears before *b* in the same session trace.

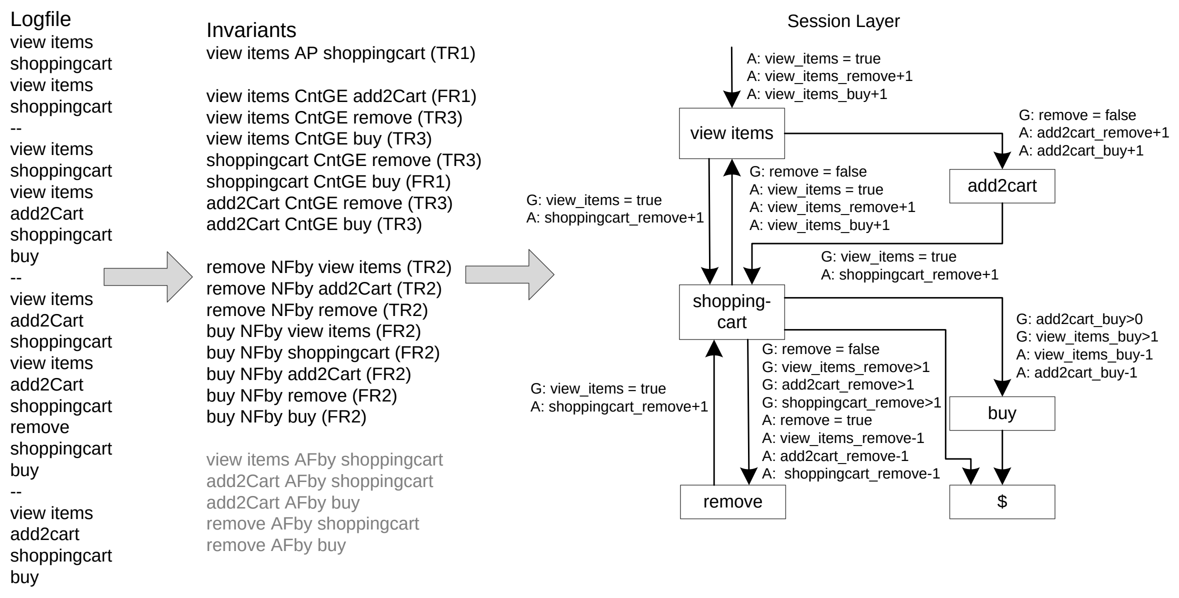
In our approach, we need to identify guards which must be true to execute the associated application transition. This identification is important in order to generate valid user navigations as some user actions can only be executed after other specific user actions have been executed.

We are not only interested in the sequence of user actions, but it is also important to specify how often an user action is executed before another. For instance, the removal of items in a shopping cart is dependent on the number of items added previously to the shopping cart. The user can only execute the user action *remove* as often as there are items in the shopping cart. Therefore, we introduce a new type of temporal invariant, which is a subset of the *AP* invariant:

- Count *a* Greater or Equal as Count *b* (CntGE): For each *AP* invariant the number of executions of *a* is always greater or equal compared to the number of executions of *b* in the same session trace. Additionally, the minimal difference between the execution of *a* and *b* is determined.

Figure 7.7 illustrates a simple example of how the temporal invariants are translated to GaAs. The simple logfile contains four sequences of user actions, each representing a session. From these sessions, 21 temporal invariants are extracted and translated to GaAs of the Session Layer.

Of the four invariant types, *AFby* cannot be used. The condition *AFby* does not mean that *a* must always be executed before *b* as seen in the example (see Figure 7.7). The invariant *add2Cart AFby shoppingcart* cannot be used as guard as the user action *shoppingcart*



```
Logfile
view items
shoppingcart
view items
shoppingcart
--
view items
shoppingcart
view items
add2Cart
shoppingcart
buy
--
view items
add2Cart
shoppingcart
view items
add2Cart
shoppingcart
remove
shoppingcart
buy
--
view items
add2cart
shoppingcart
buy
```

```
Invariants
view items AP shoppingcart (TR1)

view items CntGE add2Cart (FR1)
view items CntGE remove (TR3)
view items CntGE buy (TR3)
shoppingcart CntGE remove (TR3)
shoppingcart CntGE buy (FR1)
add2Cart CntGE remove (TR3)
add2Cart CntGE buy (TR3)

remove NFby view items (TR2)
remove NFby add2Cart (TR2)
remove NFby remove (TR2)
buy NFby view items (FR2)
buy NFby shoppingcart (FR2)
buy NFby add2Cart (FR2)
buy NFby remove (FR2)
buy NFby buy (FR2)

view items AFby shoppingcart
add2Cart AFby shoppingcart
add2Cart AFby buy
remove AFby shoppingcart
remove AFby buy
```

Figure 7.7: *Exemplary translation of temporal invariants to Guards and Actions*

can also be called from the state *view items*. Thus, the user action *add2cart* is not a prerequisite to executing the user action shoppingcart.

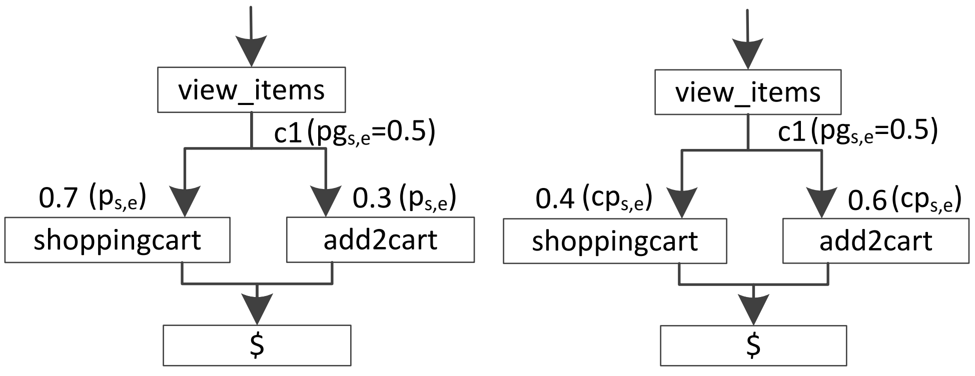
Not all of the remaining temporal invariants are required to generate valid user behavior. Furthermore, the translation of all resulting invariants into GaAs would make the workload specification quite complex. Therefore, we use the following filter rules to check if each temporal invariant is required:

- Filter rule 1 (FR1): Assuming a temporal invariant a to b exists. If state a and state b are directly connected and state b has only one incoming transition from state a and the minimal difference is zero, then the guard is not required. In this case state a, is always called before state b. Furthermore, state b cannot be called more frequently than state a. For instance, *shoppingcart CntEG remove* does not need to be considered, as *remove* can only be called when *shoppingcart* is called.

- Filter rule 2 (FR2): A path from a to b exists. This filter rule is important for the NFby invariant, as Synoptic does not check if a can be followed by b.

For the remaining invariants, we define the following translation rules. The invariants *NFby* and *AP* are translated into Boolean state variables whereas the new invariant *CntGE* is translated to a numeric state variable.

- Translation rule 1 (TR1): Boolean variable a for *AP*
  - Action: If event a is executed, the variable a is set to true.
  - Guard: Each transition to event b validates if a is true.
- Translation rule 2 (TR2): Boolean variable a for *NFby*



(a) Measured probabilities (b) Conditional probabilities

Figure 7.8: *An exemplary Behavior Model with measured probabilities and with conditional probabilities*

- Action: If event *a* is executed, the variable *a* is set to true.
- Guard: If a path from *a* to *b* exists, then each transition to event *b* checks if *a* is false.

- Translation rule 3 (TR3): Numeric variable *a_b* for *CntGE*
  - Action: If event *a* is executed, the value of a variable called *a_b* is increased by one.
  - Guard: Each transition to event *b* first checks whether variable *a_b* is greater than the minimal difference between the execution of *a* and *b*. If yes and the transition is executed, *a_b* is decreased by one.

In our example we identified nine relevant invariants which are translated into GaAs. The precondition for a transition to be executed is that all guards must be true (logical conjunction). A good example is the temporal invariant *add2Cart CntGE remove*. The user action *remove* can only be called when the user has previously added an item to the shopping cart. Thus, the transition from *shoppingcart* to *remove* is only executable when the condition *add2Cart_remove* is greater than one. In this case, the condition must be greater than one, as the minimal difference of request counts between *add2cart* and *remove* is one. We represent the guard as a numeric variable for this type of relationship. When the previously required user action *add2Cart* is executed, the variable *add2Cart_remove* is increased by one. Later, the transition from *shoppingcart* to *remove* validates whether this condition is true or not. If this condition is true, the transition is fired and *add2Cart_remove* is decreased by one.

7.4.5 Calculation of Conditional Probabilities

Because we combine GaAs with probabilities, the calculation of conditional probabilities may be required. The conditional probability is the probability that a transition will be executed given that the corresponding guard condition is true. The conditional probability can be considerably different from the probability $p_{s,e}$ (see Section 7.3.1.2) that we have extracted in the Behavior Mix Extractor for each Behavior Model (see Section 7.4.2).

To exemplify this, assume we have extracted a simple Behavior Model from a session log (see Figure 7.8(a)). This model consists of a transition from *view_items* to *add2cart* that is executed in 30% of cases and of a transition from *view_items* to *shoppingcart* executed in 70% of cases. We assume that the probability that the guard condition $c1$ is true is 50%. When we execute or simulate this Behavior Model, the transition from *view_items* to *add2cart* would be executed in only 15% (i.e., 50% · 30%) of cases and the transition from *view_items* to *shoppingcart* in 85% of cases. As this result is different from the initially measured transition probabilities, the request counts of the extracted workload specification would be different from the request count of the original workload. In the remainder of this section, we propose a heuristic to calculate the conditional probabilities. As future work, we will examine other approaches like Bayesian networks as well.

In the first step, in order to calculate the conditional probability, we have to obtain the probability for each transition that the respective guard condition is true. Let $pg_{s,e}$ be the probability that the guard g from state s_s to state s_e is true.

Based on the session log we can calculate this value by computing the relative frequency of each transition that the corresponding guard conditions is true: from the measured session log we take the sessions belonging to a Behavior Model $\mathcal{B}$, as obtained by the clustering. Then, we interpret each session by iterating the transitions according to the measured state sequence. Within each state, we determine the potential transitions to the next states, according to the Behavior Model. Afterwards, for each transition the value of the guard condition identified in the previous step (see Section 7.4.4) is determined. Then, the next state is chosen according to the state sequence and the corresponding action is executed. This way, we calculate the value of $pg_{s,e}$ for each transition by:

$$pg_{s,e} = \frac{\text{Count } g_{s,e} \text{ is true}}{\text{Count } g_{s,e} \text{ is evaluated}} \tag{7.6}$$

For example, we have 100 sessions for the exemplary Behavior Model of Figure 7.8. Within these sessions, the state *view_items* occurs 100 times. Each time the state *view_items* is examined, we evaluate the guard conditions of the potential transitions to *add2cart* and to *shoppingcart*. Assume that in 50 cases the condition of the transition to *add2cart* was true and in 100 cases the condition of the transition to *shoppingcart* was true. Then, pg for the transition to *add2cart* is 50% (i.e., 50/100) and to *shoppingcart* it is 100% (i.e., 100/100).

In the second step, we calculate the conditional probabilities of all transitions where the probability that the guard condition is smaller than one, as in these cases the probability must be increased. For each transition bt of a Behavior Model $\mathcal{B}$ we calculate the transition probability $cp_{s,e}$ under the condition that the corresponding guard is true according to Kolmogorov (1950) by:

$$cp_{s,e} = \frac{p_{s,e}}{pg_{s,e}}, \forall\{bt \mid 0 < pg_{s,e} < 1\} \tag{7.7}$$

In our example, the conditional probability $cp_{s,e}$ for the transition *view_items* to *add2cart* would be adjusted to 60% (i.e., 30%/50%) as $0 < pg_{s,e} < 1$ (see Figure 7.8(b)).

In the third step, to ensure that the sum of the conditional probabilities from one state to the following states is again 100%, we have to adjust the probabilities of the transitions where $pg_{s,e} = 1$. All transitions from state s to the following states $E = e_1, ..., e_N$ are identified and adjusted by:

$$cp_{s,e} = p_{s,e} \cdot \frac{1 - \sum\limits_{i=1}^{n} cp_{s,e_i}}{\sum\limits_{i=1}^{n} p_{s,e_i}}, \forall\{bt \mid pg_{s,e} = 1\} \tag{7.8}$$

In our example, the percentage of transitions from *view_items* to *shoppingcart* is 70% and the conditional probability of *view_items* to *add2cart* is 60% (see previous step). Therefore, we adjust the probability to 40% (i.e., 70% · (100% - 60%) / (70%)).

If all guard conditions $pg_{s,e}$ from one state to the following states are smaller than one, we have to adjust the probabilities by:

$$cp_{s,e} = cp_{s,e} \cdot \frac{1}{\sum\limits_{i=1}^{n} cp_{s,e_i}} \tag{7.9}$$

The originally calculated probabilities within the Behavior Models are adjusted according to the conditional probabilities. Thus, during the transformation to performance evaluation tools only the calculated conditional probabilities are taken into account.

7.4.6 Generating WESSBAS-DSL Instances

The next task is to transform the extracted Behavior Models, the Behavior Mix, the Workload Intensity, and the GaAs to a valid WESSBAS-DSL instance, which can be further transformed to load generation tools and performance models. Therefore, the WESSBAS-DSL *Model Generator* (Figure 7.1), performs the following steps automatically:

1. Construction of an Application Model, based on SUT-specific states and transitions,
2. integration of the Behavior Mix including the extracted Behavior Models,
3. integration of the Workload Intensity definition,

4. integration of Guards and Actions and conditional probabilities, and
5. extraction and integration of input parameters.

The Wessbas-DSL Model Generator reads the resulting Behavior Models, builds a corresponding Session Layer EFSM, and assigns a Protocol Layer EFSM to each Application State. The transitions of the Session Layer EFSM are set according to the Behavior Models. A transition from service *a* to service *b* is set, when in one of the Behavior Models a corresponding transition with probability greater than zero exists. From each service, a transition to the final state is set, as each session can be canceled by the user at any time.

The structure of our Protocol Layer EFSMs has one Protocol State per EFSM, providing exactly one request being sent in an Application State. A DSL that allows the definition of more complex, protocol-specific EFSMs, e.g., failed user logins, denotes a future work issue. In our case, we extract HTTP requests from the SUT. For other request types, e.g., Java requests, further extensions need to be developed. For each Protocol State, we integrate the information required to create executable load tests (see Section 7.4.1), like the “host IP” and the “port”, and add this information as property to the request type of the Protocol State. We also integrate the used parameters with the associated parameter values. For each parameter of a Protocol State, all parameter values are stored as a list and can later be reused by load test generators.

The integration of Behavior Mix and Behavior Models includes the construction of corresponding Wessbas-DSL elements (see Figure 7.3). Each Behavior Model is created based on the Behavior Models extracted in the previous step. Each available service is mapped exactly to a Markov State. Finally, the transitions are created for all transitions within the Behavior Models with probability greater than zero.

7.5 Generating JMeter Test Plans

A given Wessbas-DSL instance can be transformed into a corresponding JMeter Test Plan. We developed a publicly available extension, called Markov4JMeter (van Hoorn/Rohr/Hasselbring, 2008), for the well-known load generator Apache JMeter, which allows us to define and execute these workload specifications. JMeter supports the generation of workloads of various types of systems, not limited to Web-based systems.

The *Test Plan Generator* (Figure 7.1), reads a serialized Wessbas-DSL instance, as described in Section 7.4.6, from file and constructs a further XMI structure, which can be processed by the JMeter tool. The XMI output is generated via the JMeter API and denotes a JMeter-typical tree structure of Test Plan elements, including Markov4JMeter-specific elements, namely *Markov States* and a *Markov Session Controller*, which are provided by the Markov4JMeter add-on for JMeter (van Hoorn/Rohr/Hasselbring, 2008).

The probabilities and think times of the Behavior Models are defined in external comma-separated value CSV files. These CSV files are read by the Markov-4JMeter extension and consist of the transition probabilities and the think times between the Markov States

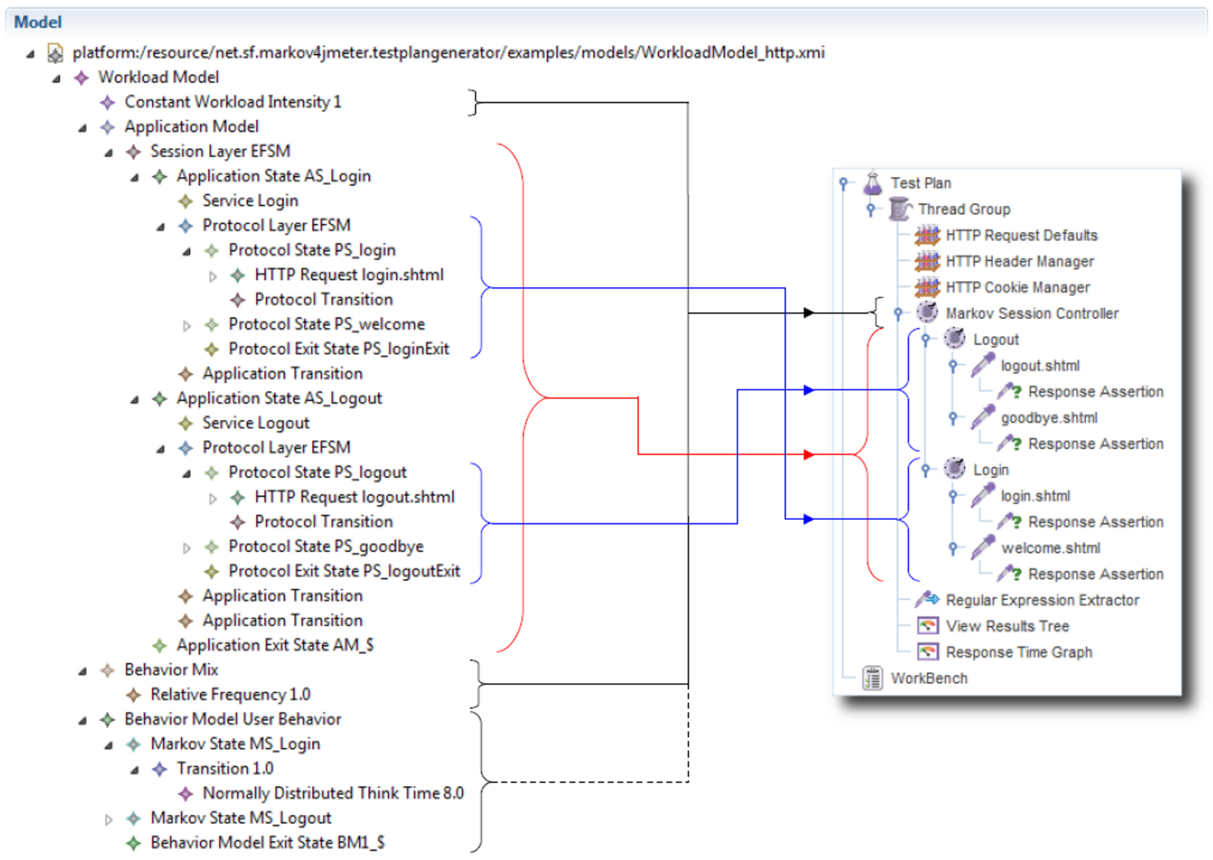


Figure 7.9: *Example mapping of* WESSBAS-DSL *instances to (Markov4)JMeter Test Plan elements*

Table 7.2: *Probabilities and think times of a Behavior Model (see Figure 7.2)*

	login*	view items	add2cart	remove	$
login*	0.0; n(0 0)	0.3; n(3 0.3)	0.3; n(3 0.2)	0.4; n(5 0.9)	0.0; n(0 0)
view items	0.0; n(0 0)	0.0; n(0 0)	0.4; n(4 0.4)	0.6; n(2 0.8)	0.0; n(0 0)
add2cart	0.0; n(0 0)	0.5; n(5 0.8)	0.1; n(4 0.1)	0.2; n(4 0.2)	0.2; n(7 0.9)
remove	0.0; n(0 0)	0.3; n(2 0.5)	0.0; n(0 0)	0.0; n(0 0)	0.7; n(5 1.0)

represented as a matrix (see Table 7.2). For instance, the probability of the transition *add2cart* to *remove* is 20% with a mean think time of 4 seconds and a standard deviation of 0.2 seconds. As the normal distributed think times can be below zero, Markov4JMeter automatically handles negative values as zero. These files were also automatically created by the JMeter Test Plan Generator.

On start of the transformation process, WESSBAS-DSL input models are validated with respect to the OCL constraints discussed in Section 7.3.2. The core transformation process builds on a mapping between WESSBAS-DSL concepts and (Markov4)JMeter Test Plan elements. An overview of the underlying mappings is given in Table 7.3.

A Session Layer EFSM in the WESSBAS-DSL is mapped to a corresponding set of Markov States in JMeter. Each Markov State includes its individual set of outgoing transitions with GaAs, for defining the validity of state execution sequences. For each guard and action parameter, a so-called User Parameter is created. In contrast to User Defined Variables, User Parameters are specific for each thread. User Parameter and User Defined Variables are Test Plan elements which are provided by JMeter. The name of a Markov State in the resulting JMeter Test Plan corresponds to the name of the state's associated service in the WESSBAS-DSL instance. Protocol Layer FSMs are modeled as child elements of Markov States in the tree-structured result and they are constructed with the use of JMeter controllers and samplers according to their related WESSBAS-DSL structure.

The values of each request parameter are created as a User Defined Variable with the parameter name and a list of the measured parameter values. As default setting, the parameter values are randomly chosen during load execution. However, there are also parameter values which cannot be reused (e.g., identifiers generated during load test execution). A limitation of the JMeter Test Plan Generation process is that the values of these parameters cannot be generated automatically. Thus, these parameter values must be identified and specified manually by the load tester. For instance, when it is necessary to specify an item for a delete request on the protocol level. These identifiers can either be generated randomly or extracted using regular expressions or XPath extractors. In case Web site are tested these IDs can be extracted from the HTML code.

The Workload Intensity is stored as a formula string in the *Session Arrival Controller* sub-component of a Test Plan's (unique) Markov Session Controller. That controller additionally includes a table for Behavior Mix frequencies, to be filled with appropriate values of the input WESSBAS-DSL instance. Behavior Models are stored separately in CSV files, which are referred by the frequency table of the Markov Session Controller. In addition to the Test Plan elements that result from the core transformation process for a given WESSBAS-DSL instance, several JMeter elements are added to a generated Test Plan by default. This step is required for making a Test Plan's structure accessible for the JMeter tool and providing additional functionality, such as handling of HTTP session cookies. Figure 7.9 shows the mapping of WESSBAS-DSL instances to JMeter Test Plan elements. Currently, the Test Plan structure is predefined, targeting HTTP-based tests only; an appropriate mechanism for specifying alternative structures, particularly for different types of requests, denotes a future work issue.

Table 7.3: *Mapping of* WESSBAS-DSL *concepts to (Markov4)JMeter elements*

WESSBAS-DSL	*Markov4JMeter Elements*
Session Layer FSM	Markov States (+ outgoing transitions)
Protocol Layer FSMs	JMeter Elements (Markov State children)
Workload Intensity	MSC (Session Arrival Controller)
Behavior Models	MSC (frequency table) → CSV-files
Behavior Mix	MSC (frequency table)
Input Parameter	User Defined Variables
Guards and Actions	User Parameters

MSC = Markov Session Controller

7.6 Transformation to Performance Models

This section explains the proof-of-concept transformation of Wessbas-DSL instances to workload specifications of the Palladio Component Model (PCM). First, Section 7.6.1 gives a short overview of PCM, followed by the description of how the system-specific parts of performance model are generated (see Section 7.6.2). Finally, Section 7.6.3 depicts how Wessbas-DSL instances are transformed to workload specifications of PCM.

7.6.1 Palladio Component Model

PCM is a domain-specific modeling language that allows the prediction of acQoS like response times, utilization, and throughput (Becker/Koziolek/Reussner, 2009). It is composed of five complementary model types. The central model type is the *Repository Model*, which models the software components, component operations, and the relations between them. The components can provide an interface to offer operations to other components or require interfaces to access operations from other components. The modeled components are assembled in a *System Model* to represent the application system. Resource containers (e.g., servers) and their associated hardware resources are modeled in the *Resource Environment Model*, whereas the *Allocation Model* defines the allocation of assembled components to the resource container. The *Usage Model* defines the workload of the system.

7.6.2 Generation of Performance Models

As our proposed approach focuses on the generation of PCM workload specifications, the system-specific parts of the model need to be created in a separate step. Since manual modeling requires too much effort, approaches that automatically extract PCM instance from design specification or running applications, e.g., (Brosig/Huber/Kounev, 2011; Brunnert/Vögele/Krcmar, 2013) can be used to generate the system-specific part of the SUT.

We use the approach proposed by Brunnert/Vögele/Krcmar (2013) to generate the system-specific parts of the performance model. A Java EE Servlet filter is used to collect data about the requests to Web components (i.e. Servlets, Java Server Pages (JSP)). The data includes the component invocations, the relations between them, and CPU resource demands for each request. Afterwards, the performance model is created and the mean CPU demand per component invocation is integrated. We create the performance model on the level of requests to the web components only and do not split further into other components like EJBs. Thus, the resulting model is very simple. For each simulated request the average measured CPU time will be used for the performance prediction. Further details on the performance model generation approach are presented in Brunnert/Vögele/Krcmar (2013).

7.6.3 Transformation

The PCM Usage Model offers only basic support for modeling complex workloads: that is the Usage Model does not allow the modeling of arbitrary usage flows. Each element can only have one incoming and one outgoing edge. Branches can be modeled with branch actions composed of multiple branch transitions, but it is not possible to interconnect elements of different branch transitions with each other. Thus, control flows like nested loops or connections between elements of one branch transition with elements of other branch transition cannot be modeled. Furthermore, the Usage Models grow in complexity for larger workloads, due to the lack of concepts enabling reuse. Consequently, we cannot transform the WESSBAS-DSL solely to the Usage Model. We generate parts of the workload specification into the Repository Model (cf. (Vögele et al., 2014)) as it does offer this kind of structuring. This violates the clear separation of the PCM models but reduces the complexity of the transformation considerably. Furthermore, using this approach we do not need to extend the PCM meta-model.

During the transformation, elements of the WESSBAS-DSL are mapped to elements of PCM as summarized in Table 7.4. The Protocol Layer and the input parameter cannot be modeled in PCM; this information is only required for load generators.

The GaAs and the probabilities cannot be modeled independently from each other in PCM. Therefore, the Behavior Models and the Session Layer are combined by modeling the GaAs of the Session Layer transitions to the transitions of the Behavior Model.

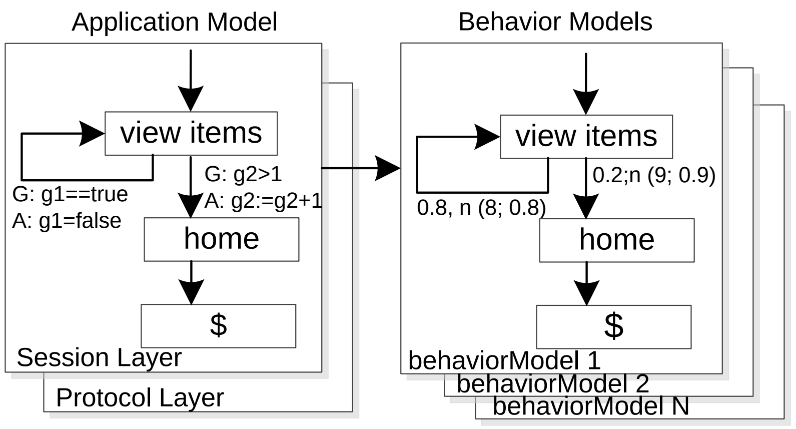
First, the PCM Repository Model generated by the performance model generator in the previous step is loaded and for each Behavior Model of the WESSBAS-DSL a new component with a corresponding interface used to represent the relationships between the components is generated. Furthermore, for each Markov State of a Behavior Model, a component operation as RDSEFF (Becker/Koziolek/Reussner, 2009) is created. An RDSEFF describes the performance relevant behavior of component operation in a way similar to UML activity diagrams. The values of the guard conditions are passed using input parameter.

An example, can be found in Figure 7.10. The Workload Model in Figure 7.10(a) will be first transformed to the PCM repository model (see Figure 7.10(b)). *Component1* with the operations *view_items* and *home* is a component of the SUT already generated by the performance model generator. For *behaviorModel1*, a new component and an interface

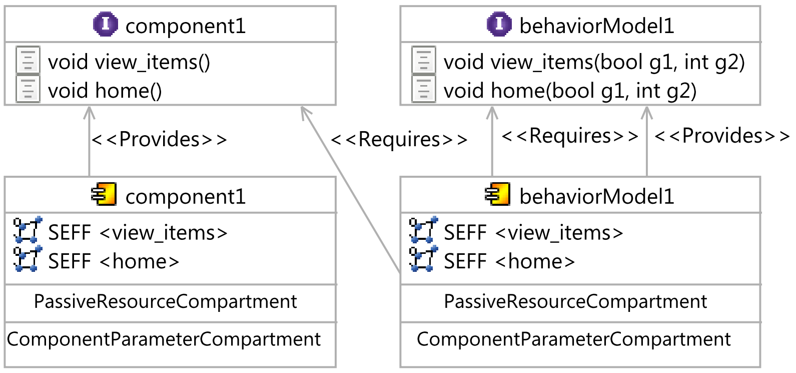
Table 7.4: *Mapping of* WESSBAS-DSL *concepts to PCM Model elements*

WESSBAS-DSL	*PCM Model Elements*
Session Layer FSMs	Repository Model (Basic Component, RDSEFF)
Protocol Layer FSMs	not required
Workload Intensity	Usage Model (Open / Closed Workload)
Behavior Models	Repository Model (Basic Component, RDSEFF)
Behavior Mix	Usage Model (Branch)
Input Parameters	not required
Guards and Actions	Parameter

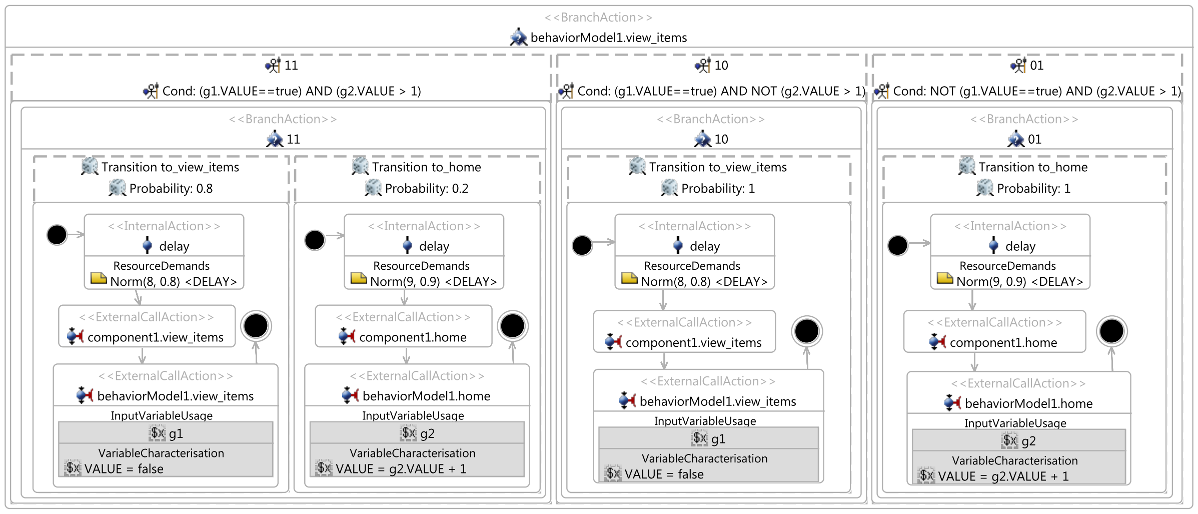
RDSEFF = Resource Demanding Service Effect Specification



(a) Workload Model example



(b) PCM Repository Model example



(c) RDSEFF of behaviorModel1.view_items

Figure 7.10: *Exemplary transformation to PCM*

are created with the RDSEFFs *view_items* and *home*. The Behavior Model component requires the interface of *component1* as this component provides the system operations, that are called during the transitions. Furthermore, *behaviorModel1* requires its own interface, as operations from the Behavior Models call themselves.

The transitions of the current Markov State to its successors are represented within each RDSEFF. In this way, the allowed sequence of service invocations is controlled by the Markov States themselves. Each RDSEFF consists of one branch. Within this branch, guards (if existing) represented as input parameters are first evaluated using guarded branch transitions. Afterwards, for each guarded branch transition, a branch with probabilistic branch transitions to the next Markov States is modeled. In case a guarded branch transition is false, the probability of the remaining transitions must be adjusted to 100%. As PCM cannot recalculate the probabilities based on the results of the conditions dynamically, all possible guard combination outcomes are calculated first for each RDSEFF. We exclude the case where all conditions are false, as in this case, no transition can be chosen and the execution terminates. Since no overlapping branching conditions are allowed in PCM, we ensure that only one of the conditions is true.

An RDSEFF example can be found in Figure 7.10. As depicted in this figure, the RDSEFF for the Markov State *view_items* of the generated Behavior Model component *behaviorModel1* has three guarded branch transitions representing the possible guard combination. Assuming both guard conditions are true, the branch transition *to_view_items* has a probability of 80% and the branch transition *to_home* has a probability of 20%. In cases where guard $g2 > 1$ is false, the transition *to_home* cannot be executed. As a result, the probability of the transition *to_view_items* is increased to 100% (respectively for transition *to_view_items*).

Each resulting branch transition specifies the call probability and contains three different actions:

1. The think time of this transition is modeled as specified in the Wessbas-DSL using an *InternalAction* as a normal distribution with mean and deviation. In our example, the think time is specified as a normal distribution with mean and deviation. The normal think time can be less than zero as we use normal distributed think times. Therefore, zero is used when the value is negative.

2. The matching operation of the modeled system components is called as an *ExternalCallAction*. To identify the corresponding system operation, we use name mapping between the name of the system operation and the name of the Markov State. Only the operations of components that provide external system calls will be matched with the Markov State names. In the transition *to_view_items* of our example the operation *view_items* of *component1* is called as it has the same name as the next Markov State *view_items*.

3. The RDSEFF of this Behavior Model component representing the next Markov State is called as *ExternalCallAction*. The values of the guard conditions are adapted according to the corresponding action; in the *to_view_items* transition of our example, the *view_items* state is called again and the value of *g1* is set to true. In the *to_home* transition, the state *home* is called and the value of *g2* is increased by one. In this way, the RDSEFFs of the Behavior Model component call themselves until a RDSEFF without successor is reached. Then, no further call is modeled, and the sequence terminates. In our example, *home* does not have a successor as there is only a transition to the exit state.

After creating the Behavior Model components in the Repository Model, each newly created component is allocated to the System Model and correspondingly to the Allocation Model. A new Usage Model is generated with one probabilistic branch representing the Behavior Mix. For each created Behavior Model component, a branch transition with the relative frequency as call probability is created. The initial Markov State of the Behavior Model is called within this transition. Finally, the Workload Intensity is modeled as closed workload with *(i.)* the population representing the number of active sessions and *(ii.)* the think time between the end and the start of a new session.

7.7 Evaluation

During evaluation, we apply our proposed extraction approach and tooling to the industry-standard benchmark SPECjEnterprise2010[23] and to the World Cup 1998 access logs (Labs, 1998). This serves as an investigation of *(i.)* the representativeness of the extracted workload specifications (quantitative) and *(ii.)* the practicality of the approach and tooling support (qualitative).

7.7.1 Research Questions and Methodology

We particularly investigate the following five research questions in order to evaluate our proposed approach:

- *RQ 1: How accurately do the clustering results match the input Behavior Mix?*

  The accuracy of the clustering is evaluated based on the fraction of misclassified sessions over all classifications of the clustering for benchmark runs using different Behavior Mix settings (see Section 7.7.4.1). To answer this question, classified sessions are required.

- *RQ 2: What is the impact of the clustering results on the workload characteristics of the executed and predicted workload?*

  First, JMeter and PCM instances are extracted. The JMeter test plans are executed on the SUT and the PCM instances are simulated. Afterwards, the impact of the clustering on the workload characteristics is evaluated based on: *(i.)* three session-based metrics namely, session length distribution (as number of requests per sessions), session duration, and number of distinct session types; *(ii.)* a request-based metric, namely the relative invocation frequency of all request types. Conclusions about the arrival rates of requests can be drawn by looking at the invocation frequencies of requests (see Section 7.7.4.2 and Section 7.7.4.3).

[23]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.

- *RQ 3: How accurately do the performance characteristics of the production system / SUT match the performance characteristics using the generated and predicted workload?*

  The accuracy of the performance characteristics using the generated and predicted workload is evaluated based on CPU utilization, response times, and heap usage. The heap usage is only evaluated for the measured and extracted workload, as it cannot be modeled and predicted using PCM (see Section 7.7.4.4).

- *RQ 4: How accurately do the workload and performance characteristics match when applying different workload settings to the extracted workload?*

  In test environments, it is often of interest to evaluate the impact of different workload settings like increasing workload intensity or different Behavior Mixes. Therefore, we evaluate workload and performance characteristics when these two workload settings are changed. We use the extracted workload from RQ 2 and RQ 3 and change the Workload Intensity and mix. The results will then be again compared with measured characteristics extracted from the original workload (see Section 7.7.4.5).

- *RQ 5: What is the impact of GaAs on the workload and performance characteristics?*

  The impact of the GaAs will be evaluated by executing workloads with and without the use of GaAs (see Section 7.7.4.6).

Ideally, these questions should be answered using logs of a real-world system to obtain production workloads with corresponding performance measurements and a test environment for load testing. Some non-synthetic logs of real-world system are publicly available and have been used by researchers. However, we do not have performance measurements of these systems as well. Furthermore, we have no access to test environments of these systems to evaluate the extracted JMeter Test Plans. Thus, we can use these publicly available logs only to evaluate RQ2. Using synthetic logs imposes a threat to external validity and performance measurements would also be not available. As a result, lab experiments under controlled conditions is the best option for us, as we are able to evaluate all RQs. Therefore, we select an industry standard benchmark that includes a representative workload profile.

To evaluate the approach with non-synthetic logs, we use the World Cup 1998 access logs. As the sessions of these logs are not classified and as we do not have access to the Web application to analyze performance characteristics, we can use these logs only for the evaluation of RQ 2. We extract a WESSBAS-DSL instance from the logs and transform it to a JMeter Test Plan and to a PCM instance. As the World Cup Web site is no longer available, we developed and instrumented a mock-up Website that has no functionality and accepts all kinds of requests. We execute the extracted JMeter Test Plan on this Web site, which enables us to measure the workload characteristics of the extracted JMeter Test Plan. The system-specific part of the PCM Model is modeled manually and consists of one system component and default resource demands (see Figure 7.10).

An instrumented version of SPECjEnterprise2010 is used for the evaluation of all five RQs. Using this application we are able to measure workload and performance characteristics.

We executed the application with four different Behavior Mixes to obtain session logs. Based on these logs, the clustering is executed and evaluated. For the Behavior Mix extraction we applied different configurations of the X-means clustering. Afterwards, a WESSBAS-DSL instance is automatically generated from the obtained Behavior Models as described in Section 7.4.6. The resulting GaAs are shown in the Appendix (Table 7.10). The transformation from the instances to JMeter Test Plans is performed according to Section 7.5. The transformation to workload specifications of PCM is applied as described in Section 7.6. The extracted workload is executed on the same SPECjEnterprise2010 deployment in order to evaluate the workload and performance characteristics. Hence, the same session log analysis infrastructure as used for measuring the workload can be applied.

7.7.2 Fifa World Cup 1998 Access Logs

In order to evaluate RQ 2 with non-synthetic access logs, we used the World Cup 98 Web site logs (Arlitt/Jin, 2000; Labs, 1998). The logs were recorded by a typical web server over a period of about three months. Each log entry contains a unique client identifier which may also be a proxy. To identify client sessions, we used a common timeout value of 30 minutes (Filieri/Grunske/Leva, 2015) as threshold between two user requests (see Section 7.4.1). During a session, the clients move from one page to another following navigation links. The URLs of the Web site follow the form "/language/category/subcategory/page_name" where category and subcategory are not always used. For instance "/english/competition/statistics.htm", "/english/index. html", and "/english/history/past_cups/uruguay30.html" are typical URLs.

The application consists of over 32,000 pages (Filieri/Grunske/Leva, 2015). We grouped the links into the categories and subcategories resulting in 25 groups each of which corresponds to a service. When no category and subcategory is used (e.g., "/english/index.html") we use the term "mainLevel" as category.

7.7.3 SPECjEnterprise2010 Deployment

The SPECjEnterprise2010 industry benchmark deployment is used for the evaluation of the proposed approach. SPECjEnterprise2010 is a Java EE application representing a business case combining customer relationship management (CRM), supply chain management (SCM), and manufacturing. It includes a workload specification and a dataset which is needed for the execution of load tests. The workload is generated by the Faban Harness and Benchmark Driver.[24] The benchmark consists of three different application domains; namely, *Orders domain* (CRM), *Manufacturing domain*, and *Supplier domain* (SCM). The *Orders domain* (CRM) provides a Web-based user interface representing a standard e-commerce application with product information and a shopping cart. It drives the demand to the *Manufacturing domain*, which simulates production across different manufacturing plants. The task of the *Supplier domain* (SCM) is to order new parts for

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

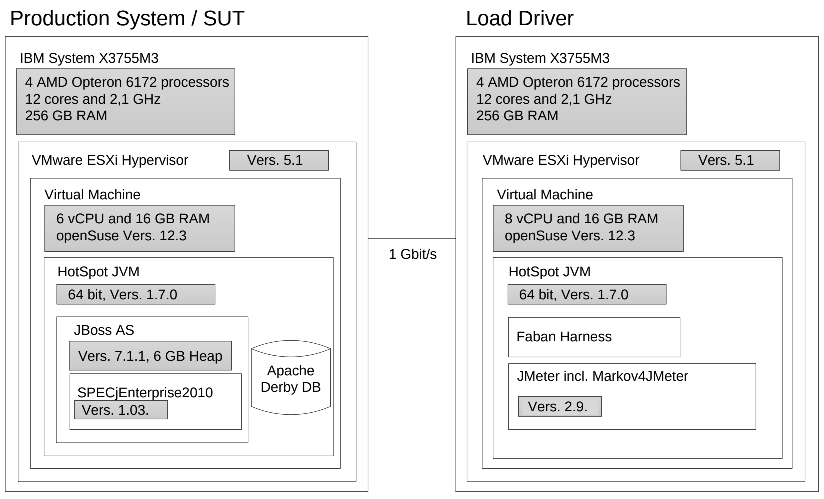


Figure 7.11: *Hardware and software infrastructure*

the Manufacturing domain. In this work, we consider only the Orders domain, which represents a typical Web-based application providing e-commerce functionality to customers; in this case automobile dealers. Using this application, customers are able to purchase and sell cars, to manage their accounts and dealership inventory, and to browse the catalogue of cars. The Orders domain runs independently from the other two domains, as they are mainly intended to be used as (Web-)service by other applications. It represents the production system / SUT.

7.7.3.1 Hardware Infrastructure

The SUT and the Dealer Driver are deployed on separate virtual machines (VM), linked by a 1 GBit/s network (see Figure 7.11). The SUT is deployed on an IBM System X3755M3 server with 6 virtual CPUs and 16 GB RAM. The Dealer Drivers also run on an IBM System X3755M3 server VM with 8 virtual CPUs and 16 GB RAM. For the JMeter load test, we used 3 VMs (800 User) and 4 VMs (1200 User). The application server is JBoss 7.1.1. using the Java EE 6 full profile with 6 GB heap allocated. As persistence layer, an Apache Derby DB is used running in the same JVM as the JBoss application server. Both systems use openSUSE operating system in version 12.3 and are executed on a 64-bit OpenJDK 1.7.0 Server Java VM in version 1.7.0.

7.7.3.2 Workload Description

SPECjEnterprise2010 defines three different transaction types which are executed by automobile dealers: *Browse (B)*, *Manage (M)*, and *Purchase (P)*. Within the transaction type Browse, the benchmark driver navigates to the catalogue of available cars and browses the catalogue for a constant number of thirteen times. Manage describes a scenario during which open orders are canceled and vehicles are sold. In the more complex transaction type Purchase, orders are placed and immediately purchased or deferred. The shopping cart is either cleared or items are removed one by one until only one item remains. Each

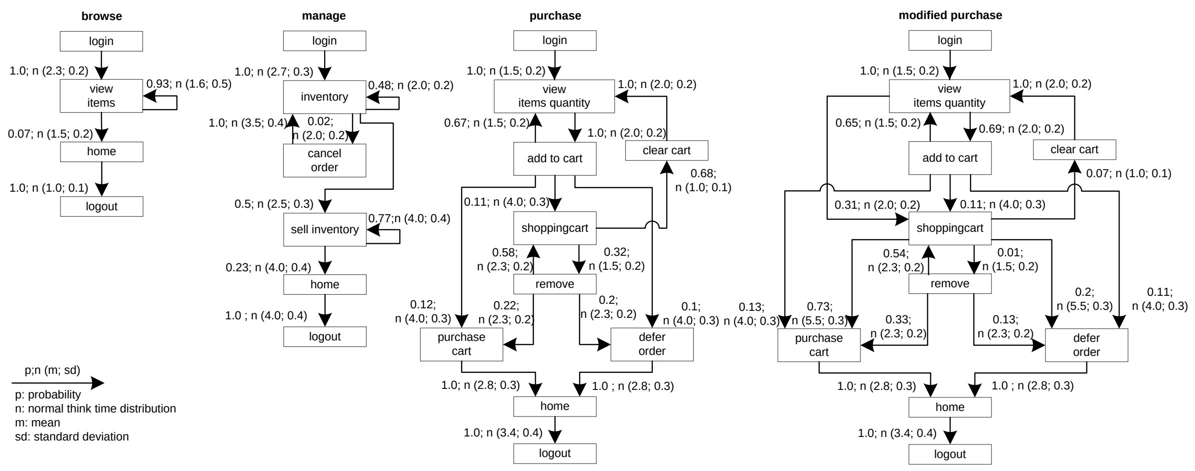


Figure 7.12: *SPECjEnterprise2010 transactions Browse, Manage, and Purchase as Behavior Models. The transaction Modified Purchase is used for the evaluation of RQ 5 (see Section 7.7.4.6). The probabilities are rounded to two decimal places and the mean and standard deviation of the think times to one decimal place.*

of these transaction types corresponds to a sequence of HTTP requests. The workload in the Faban dealer driver is not defined in a probabilistic way and only a few of the HTTP requests are generated in a probabilistic way.

SPECjEnterprise2010 defines a total of 13 different HTTP request types, using a request parameter called *action*. We additionally split the request type called *View_Items* into two different request types as it executes two different use cases resulting in different resource demands; one request type is *View_Items* and the other is *View_Items_Quantity*. In the first use case, *View_Items* is called to browse the catalogue of available cars. In the second use case, only one specific item of the catalogue is selected.

Within the original dealer driver, no think times are defined between the execution of the HTTP actions, i.e., each HTTP action is executed directly after its previous request has been completed. Therefore, we added think times between these actions as Gaussian distribution with mean and standard deviation. The think times are randomly specified between mean values of one to four seconds. Figure 7.12 depicts the structure of the three transaction types as Behavior Models, obtained by applying our WESSBAS extraction approach including the transition probabilities and the specified think times.

In the original benchmark workload, automobile dealers log in to the system, execute *multiple instances* of the three transactions types, and log out. Each of the three transaction types is executed with a specified probability. The standard transaction mix is 50% Browse, 25% Manage, and 25% Purchase. We modified the dealer driver such that *each* transaction starts with a login and ends with a logout. In this way, each transaction corresponds to a unique session and the transaction mix corresponds to the Behavior Mix. As a result, the transaction types define the different navigational patterns.

Table 7.5: *Clustering results*

		X-means (min 3 cluster, max 3 cluster)								X-means (min 2 cluster, max 20 cluster)									
		ED				NED				ED			NED						
TM	T	C1	C2	C3	MC	C1	C2	C3	MC	C1	C2	MC	C1	C2	C3	C4	MC	N
50	B	10,346	0	0		10,346	0	0		10,346	0		0	10,346	0	0		
25	M	0	0	5,230	3.03%	0	0	5,230	0%	0	5,230	24.64%	0	0	199	5,031	0.96%	19,890
25	P	0	4,468	625		0	5,093	0		0	5,093		5,093	0	0	0		
25	B	0	0	5,077		5,077	0	0		0	5,077		0	0	0	5,077		
25	M	0	5,081	0	16.48%	0	0	5,081	0%	5,081	0	24.91%	182	4,899	0	0	0.89%	20,395
50	P	6,875	3,362	0		0	10,237	0		10,237	0		0	0	10,237	0		
25	B	0	5,092	0		5,092	0	0		5,092	0		0	0	0	5,092		
50	M	10,058	0	0	2.99%	0	0	10,058	0%	0	10,058	24.72%	326	9,732	0	0	1.62%	20,125
25	P	586	0	4,389		0	4,975	0		0	4,975		0	0	4,975	0		
34	B	0	0	6,917		0	0	6,917		0	6,917		0	0	0	6,917		
33	M	6,818	0	0	4.1%	6,818	0	0	0%	6,818	0	32.9%	6,581	237	0	0	1.16%	20,470
33	P	840	5,895	0		0	6,735	0		6,735	0		0	0	6,735	0		

7.7.3.3 Benchmark Execution and Monitoring

Four different transaction mixes are used to evaluate the proposed approach. For each mix, one of the transaction types is executed with a probability of 50% and the other two with 25% each. Additionally, a mix is chosen where the proportions of the transaction types are equal. A load of 800 concurrent users is executed, resulting in a moderate CPU utilization of the SUT of approximately 30%. Each benchmark run is executed for twelve minutes after a three minute ramp-up phase and before a three minute ramp-down phase. We extract the system-specific parts of the performance model (as described in Section 7.6.2) using the original workload mix once. This part of the performance model will be reused during the evaluation.

The SUT is instrumented using Kieker (van Hoorn/Waller/Hasselbring, 2012) to obtain the raw session information. Each request is recorded and afterwards transformed to a session log (see Section 7.4.1). During the transformation, we only take complete sessions during steady state into account; meaning, sessions starting with a login request after the ramp-up phase and ending with a logout request before the ramp down phase. Thus, incomplete sessions are removed. In order to be able to evaluate the clustering results of the transaction types, the name of the transaction type is added as an additional parameter to the login HTTP action.

We use the same procedure to predict the performance with PCM. However, as PCM does not provide a unique session identifier for interrelated requests, we cannot remove incomplete sessions during steady state. As a result, the predicted request counts are a little bit higher than the measured ones.

7.7.4 Evaluation Results

7.7.4.1 Accuracy of Clustering

The evaluation of clustering accuracy (RQ 1) is split into two steps. In the first step, the accuracy of the clustering is determined based on the assumption that the number of resulting clusters is known in advance. For this reason, the number of resulting clusters is fixed to three. As the number of clusters is usually not known in advance, we let the X-means algorithm determine that number in a second step. Since the seed value for the random selection of the initial centroids can have a high impact on the clustering results, multiple clustering runs are executed with different seed values between one and twelve. Afterwards, the run with the lowest sum of squared error value (Pelleg et al., 2000) is selected.

The results of the clustering are presented in Table 7.5. For each TM, the clustering shows for each T the cluster (C_x) to which a transaction is assigned, and the percentage of misclassified (MC) transactions. The left side of the table shows the results of exactly three predefined clusters (step one); the right side shows the results of letting X-means determine the number of clusters between two and twenty (step 2). The number of transactions (N) clustered for each transaction mix is around $20,000$.

The results of using exactly three clusters indicate that the clustering with use of NED is able to cluster all transactions correctly (100%), resulting in the Behavior Models shown in Figure 7.12. The clustering using ED without normalization classifies the transactions Browse and Manage correctly, whereas a fraction of transactions of type Purchase is assigned mistakenly to the same cluster as the Manage transactions. In the second transaction mix, the fraction of Purchase transactions is higher than in the other mixes. Hence, the percentage of misclassified transactions is high (16.48%). As a result, the clustering using ED is not able to cluster all transactions correctly, although each transaction comprises unique states.

The clustering without predefining the correct number of clusters always results in two clusters using ED and four clusters using NED. As clustering with ED always merges transactions of type Purchase and Manage, the percentage of misclassified transactions is between 25% and 33%. It is assumed that the transaction type with the lower number of instances merged within one cluster counts as misclassified. The clustering using NED always correctly classifies Browse and Purchase transactions whereas Manage transactions are always split into two clusters. Hence, the percentage of misclassified transactions is very low (around 1%) for all transaction mixes.

Transactions of type Browse seem to be homogeneous in a way that they are clustered correctly among all clustering runs. This can be explained as Browse transactions are executed with a constant number of actions without probabilistic behavior. NED is better suited for clustering the different transaction types than the non-normalized version. The normalization has the effect that high transaction counts, and therefore also the session lengths, have a lower impact on the clustering. Thus, the structure of the transactions in terms of the number of different HTTP requests grows in significance. As each of the three transaction types consists of different HTTP request types (except for *login*, *home* and *logout*), the clustering results are significantly better.

7.7.4.2 Accuracy of World Cup 1998 Workload Characteristics

We investigate research question RQ 2, by analyzing the impact of the clustering results on server-side session-based and request-based metrics (mentioned in Section 7.7.1) for the original measurements with the corresponding metrics obtained by executing extracted workload specifications using JMeter and PCM. In this section we present the results of the non-synthetic World Cup 1998 logs. In Section 7.7.4.3 the accuracy of the workload characteristics of SPECjEnterprise2010 is presented.

We analyzed the logs of day 42 of the World Cup in English language. We identified 53,644 sessions and in total 511,824 page requests. The logs are analyzed as described in Section 7.7.2 and a Wessbas instance is created. During the transformation no GaAs could be identified, as the Website is created in way that each Web site can be accessed by all other Web sites. We clustered the logs using X-means clustering with a minimum of 2 clusters and a maximum of 20 clusters with NED distance metric resulting in four clusters. The relative frequency of each request type per cluster can be found in Table 7.6.

The average session lengths within the cluster range from 7.58 (C4) to 13.28 (C2) requests and the number of sessions per cluster range from 3,447 (C3) to 21,205 (C2). The clusters differentiate primarily in the four request groups *mainlevel*, */teams*, */news*, and */competition*, which make up 86% of all requests.

The users of the first cluster search mainly for teams and navigate less on the main level. In the second cluster, the users navigate on the main level and on teams' pages. In the third cluster, the users request news pages and fewer on pages from the request type teams. The fourth cluster contains mainly users browsing on the main level and on pages of category competition. Users of the fourth cluster have the lowest session length. Overall, we can see that the clustering is able to identify different kinds of user groups.

The Wessbas instance is then transformed into a JMeter Test Plan and executed against the mock-up Web site (see Section 7.7.1) to measure the workload characteristics. Moreover, a PCM instance will be generated and simulated. The results are presented in the following.

Session-Based Metrics.

The session-based statistics are only compared against JMeter metrics as PCM does not generate unique identifiers for interrelated user actions. The evaluated session statistics can be found in Table 7.7.

The mean values and the 0.95 confidence interval indicate that both distributions are very similar. The number of distinct sessions of the extracted workload is with 27,548 also similar to 22,605 of the original workload. The difference in the number of distinct

Table 7.6: *World Cup 1998 Logs: Relative frequency of each request category per cluster*

Category	C1	C2	C3	C4
mainlevel	11.08%	31.79%	35.22%	55.67%
/teams	60.96%	48.10%	14.56%	11.68%
/news	14.03%	8.22%	18.07%	8.26%
/competition	5.34%	5.81%	8.02%	16.11%
/enfetes	0.96%	1.28%	3.05%	1.43%
/playing	0.98%	0.76%	3.27%	1.11%
/playing/download	0.61%	0.35%	2.77%	0.59%
/history/past	1.00%	0.52%	2.46%	0.62%
/tickets	0.71%	0.50%	2.29%	0.79%
/playing/mascot	0.52%	0.46%	2.14%	0.66%
/venues	0.39%	0.30%	1.31%	0.49%
/venues/cities	0.82%	0.32%	1.15%	0.36%
/history	0.38%	0.28%	1.00%	0.33%
/help	0.24%	0.23%	0.74%	0.37%
/hosts/cfo	0.23%	0.18%	0.77%	0.28%
/member	0.16%	0.21%	0.63%	0.34%
/hosts/suppliers	0.49%	0.07%	0.66%	0.34%
/venues/venues	0.46%	0.23%	0.59%	0.23%
/history/history	0.19%	0.14%	0.51%	0.16%
/individuals	0.31%	0.16%	0.29%	0.05%
/hosts/fifa	0.02%	0.05%	0.16%	0.05%
/hosts/sponsors	0.04%	0.02%	0.15%	0.04%
/history/reading	0.02%	0.02%	0.10%	0.03%
/hosts/fff	0.02%	0.01%	0.06%	0.02%
/playing/rules	0.01%	0.00%	0.01%	0.00%
# sessions	3,447	8,492	21,205	20,500
avg session length	8.43	13.28	10.12	7.58

Table 7.7: *Summary statistics of session lengths*

	Min.	Q_1	*Med.*	*Mean*	$CI_{0.95}$	Q_3	*Max.*	*N*
Orig.	1	4	6	9.54	[9.37, 9.72]	11	1,972	53,644
NED-4	1	3	7	9.51	[9.43, 9.58]	13	91	53,326

sessions comes from the fact that within the original workload, sessions with a very high number of user requests have been measured; at maximum 1,972 in the original workload compared to 91 in the extracted workload. This can be explained as the unique client identifier in the original non-synthetic access logs could also include requests from a proxy comprising multiple users. In contrast, in the extracted workload the session lengths is evenly distributed.

Summarizing, the session-based metrics are similar and differentiate mainly by the fact that the original workload contains very long sessions caused by proxies.

Request Counts.

In Table 7.8, the request counts of the measured (Orig.) are compared to the request counts of the executed (JMeter) and simulated (PCM) workloads. For the request counts, an almost exact match can be found for the 25 request groups. The deviation in the form of the sum of squared errors is zero. Thus, the server-side request counts are representative compared to the original workload.

Table 7.8: *Absolute and relative (Rel.) request counts*

Category	Orig.	JMeter	PCM	Rel.
mainlevel	201,167	199,569	199,165	0.39
/teams	121,354	120,043	119,871	0.24
/news	64,966	64,151	64,006	0.13
/competition	50,354	50,134	49,135	0.10
/enfetes	10,503	10,249	10,256	0.02
/playing	9,875	10,043	9,866	0.02
/playing/download	7,429	7,449	7,265	0.01
/history/past	7,096	7,066	6,813	0.01
/tickets	6,922	6,708	6,791	0.01
/playing/mascot	6,299	6,134	6,495	0.01
/venues	4,014	3,928	3,929	0.01
/venues/cities	3,633	3,661	3,726	0.01
/history	3,088	2,983	3,081	0.01
/help	2,505	2,381	2,449	0.00
/hosts/cfo	2,340	2,333	2,296	0.00
/member	2,165	2,174	2,203	0.00
/hosts/suppliers	2,164	2,168	2,061	0.00
/venues/venues	2,008	2,063	2,009	0.00
/history/history	1,557	1,400	1,633	0.00
/individuals	950	863	974	0.00
/hosts/fifa	489	509	471	0.00
/hosts/sponsors	428	458	394	0.00
/history/reading	288	264	301	0.00
/hosts/fff	189	222	153	0.00
/playing/rules	41	49	31	0.00
sum	511,824	507,002	505,374	1.00

7.7.4.3 Accuracy of SPECjEnterprise2010 Workload Characteristics

In this section, we also evaluate RQ 2, but this time with the synthetic logs generated by SPECjEnterprise2010 which serves to increase the external validity. The session-based statistics are again compared only against JMeter metrics (see Section Section 7.7.4.2). We present the results of the original benchmark Behavior Mix (25% P, 50% B, and 25% M), using the X-means clustering algorithms results with 2 (ED), 3 (NED), and 4 (NED) clusters (entries for the bottom TM in Table 7.5).

Session-Based Metrics.

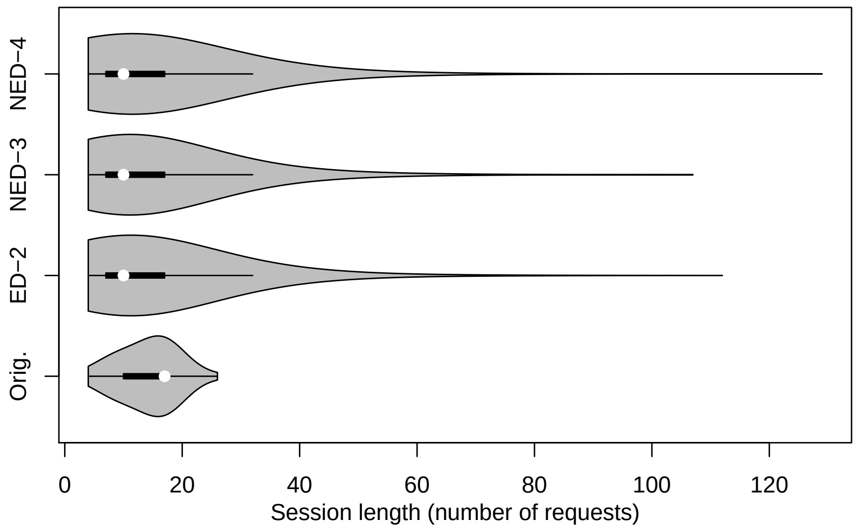
Statistics about the session length distributions of the original and the three synthetic workloads are shown in Figure 7.13. Figure 7.13(a) depicts the four distributions as violin plots. Looking only at the mean values and the 0.95 confidence interval (Figure 7.13(b)), one may conclude that the session length distributions of the three synthetic workloads exactly match the distribution of the original workload. However, particularly the violin plot (Figure 7.13(a)) indicates that the synthetic distributions are similar but differ considerably from the original workload.

The quartile-based statistics in Figure 7.13(b) confirm this observation. The same observation can be made for the distribution of session durations in seconds (Figure 7.13(c) and Figure 7.13(d)). Very long sessions for the synthetic workloads are generated. While for the original workload the longest sessions comprise 26 requests, the synthetic sessions reach maximums of 112, 107, and 129.

In the original workload, we identified 78 distinct sessions. The number of distinct sessions in the synthetic workloads is considerably higher, namely 1056 (2 clusters), 1004 (3 clusters), 960 (4 clusters). The relatively low number of distinct session types is caused by the fact that the original SPECjEnterprise2010 workload contains few probabilistic elements, which are all bounded in the number of maximum iterations (cf. Section 7.7.3.2). For instance, the *view_items* request in the browse transaction is executed exactly thirteen times. Hence, the maximum number of possible distinct sessions is countable. In our previous work (van Hoorn et al., 2014) the number of distinct sessions is around 2,000. The number of distinct sessions is lower in this paper as GaAs reduce the number of invalid sessions.

The described session length distributions of the synthetic workloads imply the high number of distinct sessions. Inspecting the structure of the synthetic sessions, we observed the following recurring patterns: *(i.) sell inventory+*, *(ii.) inventory+*, *(iii.) view items+*, *(iv.)* (*view items quantity*, *add to cart*)+, *(v.)* (*view items quantity*, *add to cart*, *shopping cart*, *clear cart*)+. These patterns can be explained by the corresponding transitions with high probabilities already indicated by the probabilities of the original workload depicted in Figure 7.12.

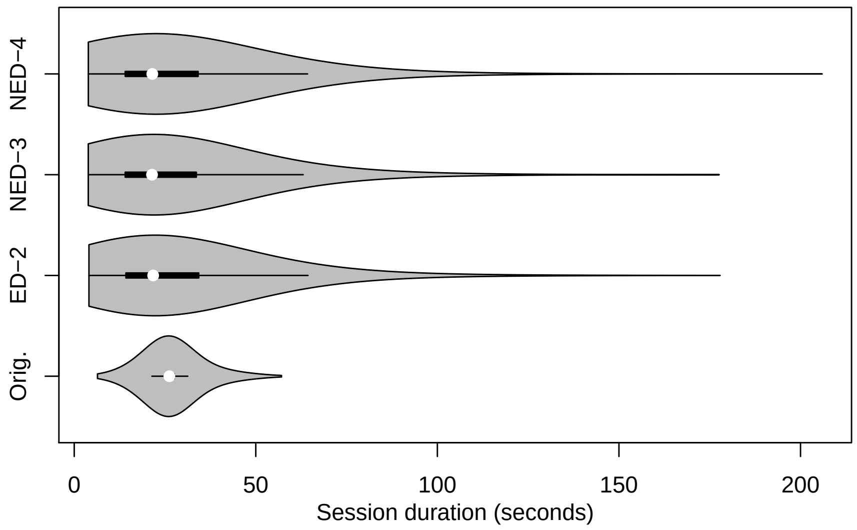
Considering the setting for SPECjEnterprise2010, the following conclusions can be drawn about the impact of the clustering results on the session-based metrics session length and number of distinct session types. Firstly, no statistically significant differences between



(a) Violin plot of session lengths

	Min.	Q_1	*Med.*	*Mean*	$CI_{0.95}$	Q_3	*Max.*	*N*
Orig.	4	10	17	14.18	[14.12, 14.24]	17	26	19,890
ED-2	4	7	10	13.96	[13.81, 14.11]	17	112	20,119
NED-3	4	7	10	13.88	[13.73, 14.02]	17	107	20,358
NED-4	4	7	10	13.96	[13.82, 14.11]	17	129	20,299

(b) Summary statistics of session lengths



(c) Violin plot of session durations

	Min.	Q_1	*Med.*	*Mean*	$CI_{0.95}$	Q_3	*Max.*	*N*
Orig.	6.40	25.09	26.20	27.05	[26.95, 27.14]	27.56	57.08	19,890
ED-2	4.05	14.23	21.74	26.97	[26.71, 27.24]	34.29	177.90	20,119
NED-3	3.87	14.03	21.43	26.63	[26.37, 26.89]	33.64	177.60	20,358
NED-4	3.88	14.03	21.50	26.79	[26.53, 27.05]	34.13	206	20,299

(d) Summary statistics of session durations (in seconds)

Figure 7.13: *800U-50B/25P/25M: Session length and duration statistics for the original workload (Orig.) and the synthetic workloads (ED-2, NED-3, NED-4)*

	Request	*Orig.*	*ED-2*	*NED-3*	*NED-4*	*Rel.*
1	add to cart	20,625	21,474	21,129	21,217	0.07
2	cancel order	191	198	176	168	0.00
3	clear cart	1,932	21,29	2,011	1,976	0.01
4	defer order	2,236	2,228	2,218	2,312	0.01
5	home	19,371	20,119	20,358	20,299	0.07
6	inventory	10,034	10,273	10,136	10,064	0.04
7	login	19,890	20,119	20,358	20,299	0.07
8	logout	19,372	20,119	20,358	20,299	0.07
9	purchase cart	2,682	2,780	2,873	2,795	0.01
10	remove	923	660	675	723	0.00
11	sell inventory	21,949	22,703	21,854	21,653	0.08
12	shopping cart	2,855	2,789	2,686	2,699	0.01
13	view items	139,370	133,766	136,529	137,723	0.49
14	view items quantity	20,625	21,474	21,129	21,217	0.07

(a) Absolute and relative (*Rel.*) counts (JMeter)

	Request	*Orig.*	*ED-2*	*NED-3*	*NED-4*	*Rel.*
1	add to cart	20,625	22,416	22,466	21,936	0.07
2	cancel order	191	217	165	208	0.00
3	clear cart	1,932	2,094	2,222	2,062	0.01
4	defer order	2,236	2,425	2,379	2,275	0.01
5	home	19,371	21,131	21,190	20,990	0.07
6	inventory	10,034	10,703	10,656	10,932	0.04
7	login	19,890	21,128	21,190	20,997	0.07
8	logout	19,372	21,128	21,190	20,997	0.07
9	purchase cart	2,682	2,806	2,919	2,840	0.01
10	remove	923	711	713	692	0.00
11	sell inventory	21,949	23,867	23,552	23,807	0.08
12	shopping cart	2,855	2,808	2,939	2,755	0.01
13	view items	139,370	146,637	146,903	148,698	0.49
14	view items quantity	20,625	22,425	22,472	21,930	0.07

(b) Absolute and relative (*Rel.*) counts (PCM)

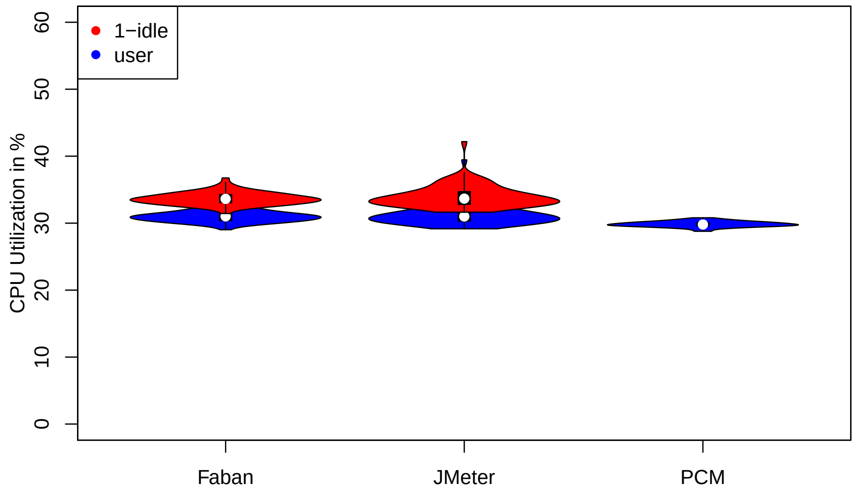
Figure 7.14: *800U-50B/25P/25M: Request count statistics*

the synthetic workloads for two, three, and four clusters in the summary statistics from Figure 7.13 can be observed. Secondly, both the session length distributions and the number of distinct sessions deviate from the characteristics of the original workload. Thirdly, the deviation of the session length distributions is mainly caused by a number of synthetic long sessions. Lastly, the mean value shows no statistically significant difference.

Request Counts.

Figure 7.14 depicts statistics about the frequency of invoked requests using JMeter and PCM, based on the absolute numbers of requests to the 14 SPECjEnterprise2010 request types. We compared the request counts of the original workload with the request counts of the three different clustering settings executed with JMeter and simulated with PCM.

As in Section 7.7.4.2, an almost exact match of the relative frequencies could be observed. Hence, from the server-perspective, the synthetic SPECjEnterprise2010 is representative in terms of the distributions of requests.



(a) Violin plots

	Mean	± $CI_{0.95}$	*Std. dev.*	*Median*	*N*
Faban (1-idle)	33.67	± 0.22	0.92	33.62	72
Faban (user)	31.06	± 0.21	0.89	31.02	72
JMeter (1-idle)	33.99	± 0.38	1.63	33.66	72
JMeter (user)	31.36	± 0.37	1.60	31.01	72
PCM	29.84	± 0.10	0.41	29.80	72

(b) Summary Statistics

Figure 7.15: *800U-50B/25P/25M: CPU utilization statistics*

7.7.4.4 Accuracy of Performance Metrics

In this section, performance characteristics of the SUT using the original workload are compared with resulting performance characteristics using the extracted and simulated workload (RQ 3). We analyze the resulting CPU utilization, server-side response times per request type and the heap usage (only Faban compared to JMeter). The results using the original benchmark Behavior Mix (50% B, 25% P, and 25% M) with 3 (NED) clusters are presented.

CPU Utilization.

Figure 7.15(a) illustrates the resulting CPU utilization using Faban Harness, JMeter, and PCM as violin plots. We measured the CPU utilization every 10 seconds using the Linux command line tool System Activity Reporter (SAR)[25]. The CPU utilization for the load driver is split into overall CPU utilization (1-idle) and user CPU utilization. As illustrated in Figure 7.15(b), the original workload using Faban resulted in a mean CPU utilization of 33.67% (1-idle) and 31.06% (user). The mean CPU utilization of JMeter is almost similar with 33.99% (1-idle) and 31.36% (user). However, the standard deviation using JMeter is higher. The predicted CPU utilization using PCM is 29.84%. This is a prediction error of 11.4% in relation to the overall utilization and of 3.92% compared to the user CPU utilization. The difference can be explained by the fact that the used performance model generator (Brunnert/Vögele/Krcmar, 2013) neglects the system utilization. The deviation of the predicted CPU demands is very low.

[25]http://linux.die.net/man/1/sar

Server-side Response Times.

The resulting server-side response times in milliseconds per request type can be found in Figure 7.16. The predicted mean response times using PCM are similar to the response times using Faban, but indicate very low response time deviations. As the generated performance model simulates average CPU demands per request type, the low deviation was expected (compare Section 7.6.2).

Comparing the response times of Faban with JMeter, the mean response times and the deviation are similar except for *purchase cart* and *cancel order*. The mean response times of *purchase cart* requests is with 16.85 ms considerably higher than the mean response times using the original workload (10.21 ms). Furthermore, the deviation is higher because the number of purchased items using JMeter can be considerably higher than in the original workload. In the original workload, a maximum of five items is purchased. As the extracted workloads are generated in a probabilistic way, the number of *add to cart* executions before the *purchase cart* request is not limited.

The reason why the mean response time of the *cancel order* requests is lower is similar. Before the request is executed, the original workload checks if open orders exist. As JMeter generates these requests in a probabilistic way, it is possible that no open orders exist. Thus, the response times and the deviation are lower. This could be manually fixed by adding an additional guard condition into JMeter, which checks if open orders exist. However, this kind of conditions cannot be extracted in an automatic way.

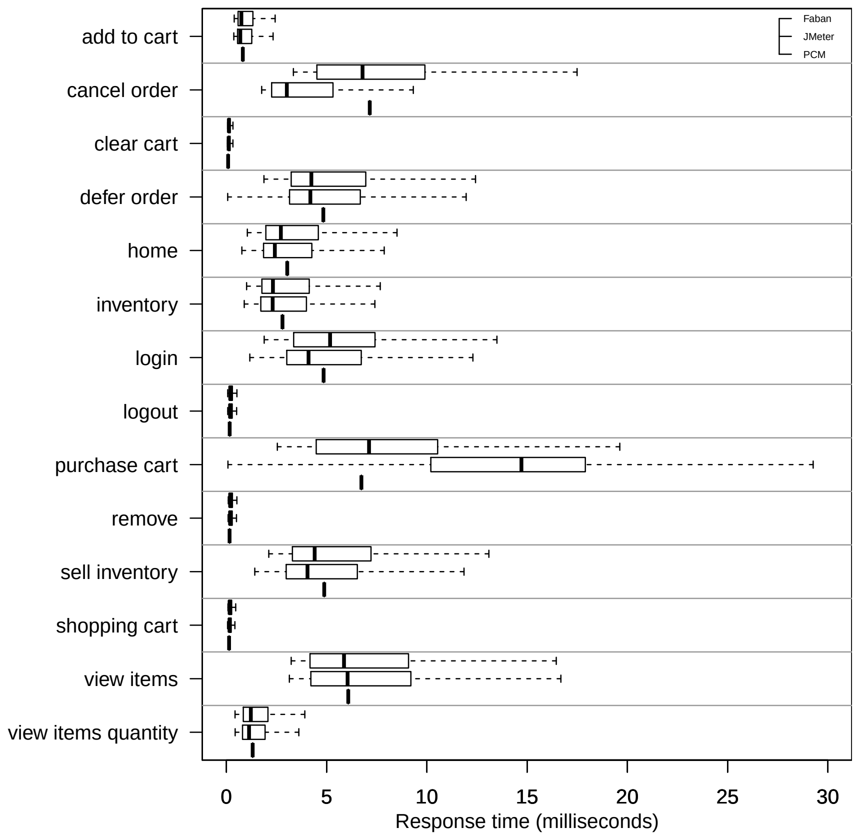
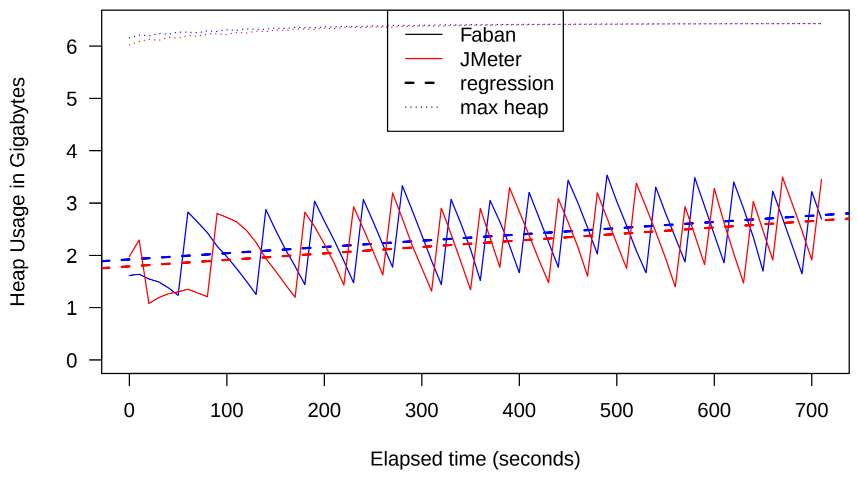


Figure 7.16: *800U-50B/25P/25M: Server-side response time statistics*



(a) Usage over time

	Mean	$\pm$ $CI_{0.95}$	*Stddev.*	*Median*	*N*
Faban	2.35	$\pm$ 0.15	0.64	2.35	72
JMeter	2.23	$\pm$ 0.16	0.66	2.23	72

(b) Summary Statistics

Figure 7.17: *800U-50B/25P/25M: Memory usage statistics (with Faban initialization)*

Heap Usage.

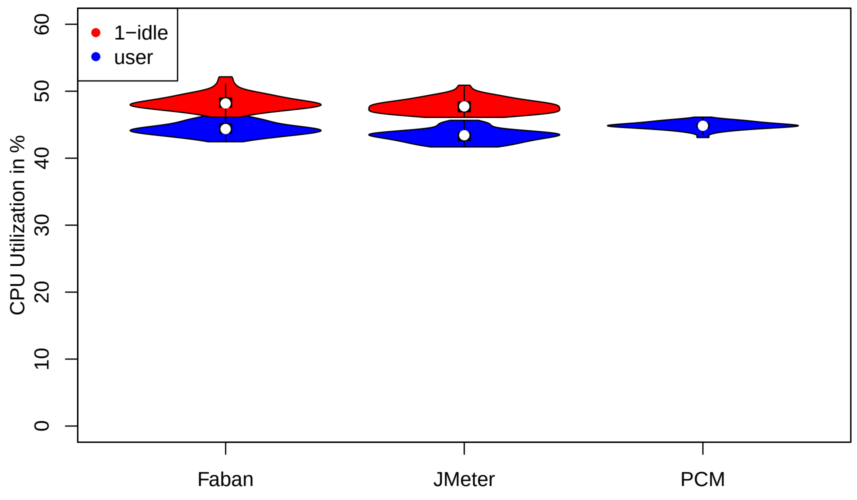
We additionally analyzed the heap usage of the original workload compared to the extracted workload (see Figure 7.17). As the Faban benchmark driver executes several read and write operations on the database (Faban initialization) before the ramp-up phase, the heap usage increases by approximately one gigabyte. In order to make the heap usage comparable, we also executed the Faban initialization phase before we started JMeter. As shown in Figure 7.17(b) the resulting mean heap usage of Faban (2.35 GByte) and JMeter (2.23 GByte) are very similar. Additionally, the regression lines run in parallel (see Figure 7.17(a)) on the same level.

7.7.4.5 Accuracy of Changing Workload Settings

In this section we describe scenarios in which the settings of the extracted workloads are changed (RQ 4). These workloads are executed (respectively simulated) and then again compared with the workload and performance characteristics from a Faban run. In this way, we evaluate the accuracy of the extracted workload models under changed settings. In the first scenario we increase the Workload Intensity only and in the second we increase the intensity and change the workload mix. In the following we present the relevant analysis.

Increasing Workload Intensity.

For the first scenario, we conducted the same experiment as before (standard benchmark mix, 3 (NED) clustering) but increased the Workload Intensity from 800 users to 1200 users. We first analyzed the workload characteristics. As the session-based and request-



(a) Violin plots

	Mean	$\pm$ $CI_{0.95}$	*Stddev.*	*Median*	*N*
Faban (1-idle)	48.39	$\pm$ 0.29	1.22	48.18	72
Faban (user)	44.48	$\pm$ 0.27	1.13	44.36	72
JMeter (1-idle)	47.77	$\pm$ 0.24	1.04	47.74	72
JMeter (user)	43.35	$\pm$ 0.23	0.98	43.43	72
PCM	44.85	$\pm$ 0.14	0.58	44.84	72

(b) Summary Statistics

Figure 7.18: *1200U_50B_25P_25M: CPU utilization statistics*

based metrics are almost identical to the run with 800 users (except for the higher number of sessions and requests) we will not present these metrics here.

The CPU utilizations increased by approximately 15% compared to the run with 800 users (Figure 7.18). The statistics show that the mean CPU utilizations of Faban (48.39%) and JMeter (47.77%) are again quite similar. The relative prediction error of PCM compared to the overall utilization of Faban is 7.3% and 0.01% for the user CPU utilization. Thus, the prediction error decreases compared to the run with 800 users.

The resulting server-side response times of Faban and JMeter increase by approximately 30% (Figure 7.19). Again, the response times are similar except for *cancel order* and *purchase cart*. The relative error for *purchase cart* requests is increased to 94% compared to Faban. The predicted response times are only increased by approximately 2% and are somewhat lower than the response times caused by the Faban load driver.

Increasing Workload Intensity and Changing Behavior Mix.

In the second scenario, we additionally changed the Behavior Mix in a way that the proportion of the transaction types are of almost equal size (34% B, 33% P, and 33% M).

Figure 7.20 shows that the mean session length decreases slightly to a mean of 13.30 compared to the original workload mix (14.18). In contrast, the mean session duration increases slightly from 27.05 to 27.26 seconds. Again, metrics generated by Faban and JMeter are very similar. By comparing the request count statistics (Figure 7.21) it can

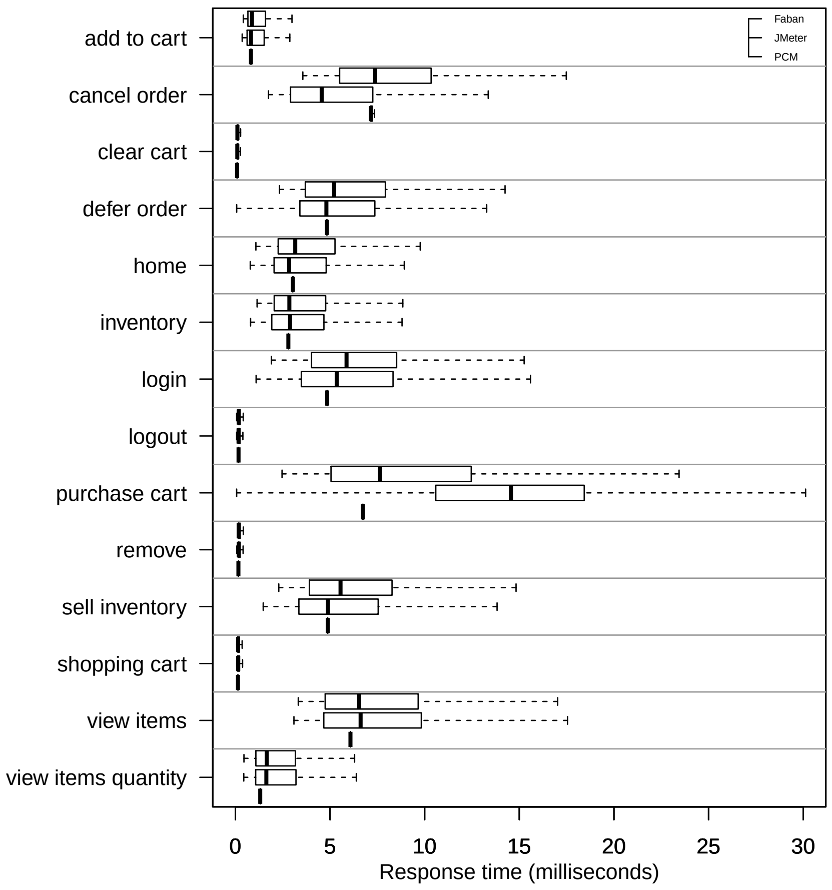


Figure 7.19: *1200U-50B/25P/25M: Server-side response time statistics*

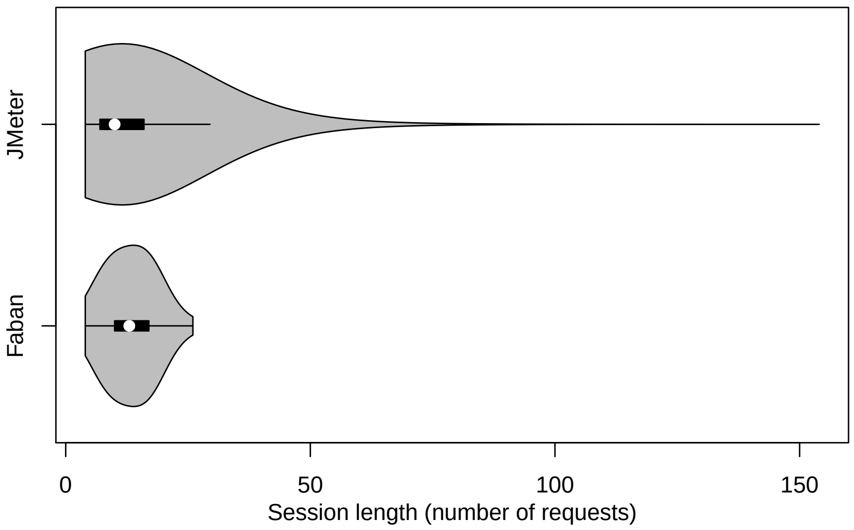
be seen that the relative error compared to the overall number of requests is again zero. This can be seen as the same bars are used for Faban, JMeter, and PCM.

The CPU utilization using the extracted workload decreases to 41.17% (1-idle) compared to the first scenario. The CPU utilization is very similar to the original workload 42.28% (see Figure 7.22). The prediction error of PCM increases to 12.7% compared to the overall utilization and 4.1% compared to the user CPU utilization. The response times are almost the same as in the first scenario and are therefore not shown (but are included in the supplementary material).

As a result, we can see that the workload and performance characteristics of the extracted workload and the simulated workload are comparable to the original workload when settings are changed in terms of Workload Intensity and Behavior Mix.

7.7.4.6 Impact of Guards and Actions

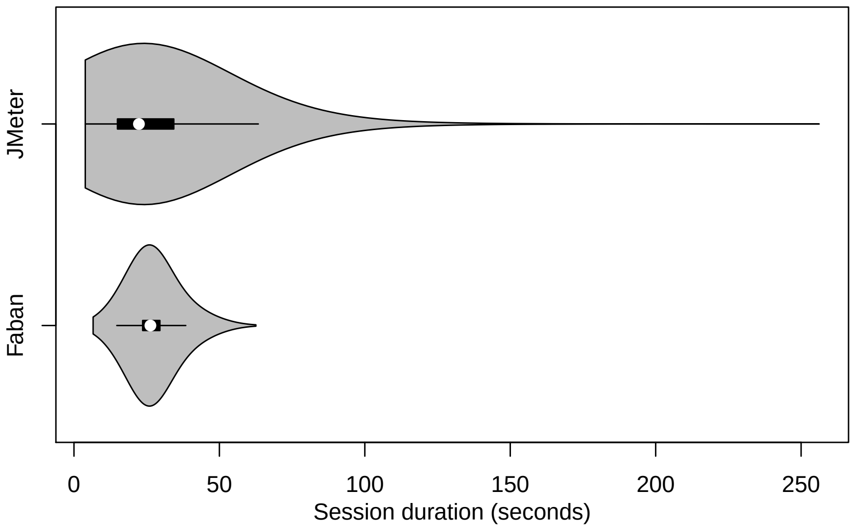
In the previous experiments the GaAs do not have a high impact on the workload characteristics. This results from the fact that the transactions of SPECjEnterprise2010 are designed in a way that no invalid user sequences are allowed. The only invalid user sequence can occur in the Purchase transaction. In this transaction, more items can be



(a) Violin plot

	Min.	Q_1	*Med.*	*Mean*	$CI_{0.95}$	Q_3	*Max.*	*N*
Faban	4	10	13	13.30	[13.25, 13.36]	17	26	29,558
JMeter	4	7	10	13.20	[13.08, 13.31]	16	154	29,746

(b) Summary statistics

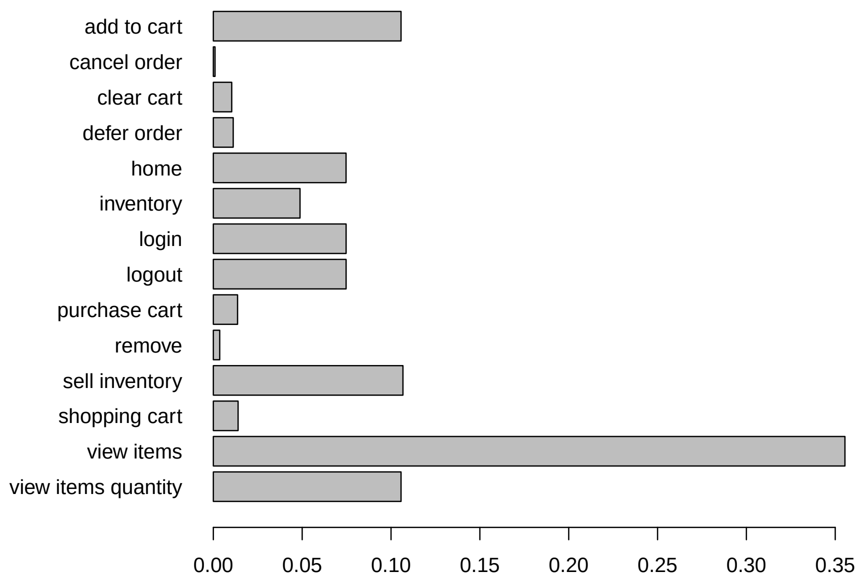


(c) Violin plot

	Min.	Q_1	*Med.*	*Mean*	$CI_{0.95}$	Q_3	*Max.*	*N*
Faban	6.58	23.62	26.28	27.26	[27.17, 27.35]	29.55	62.55	29,558
JMeter	3.88	14.95	22.33	27.40	[27.19, 27.60]	34.32	256.20	29,746

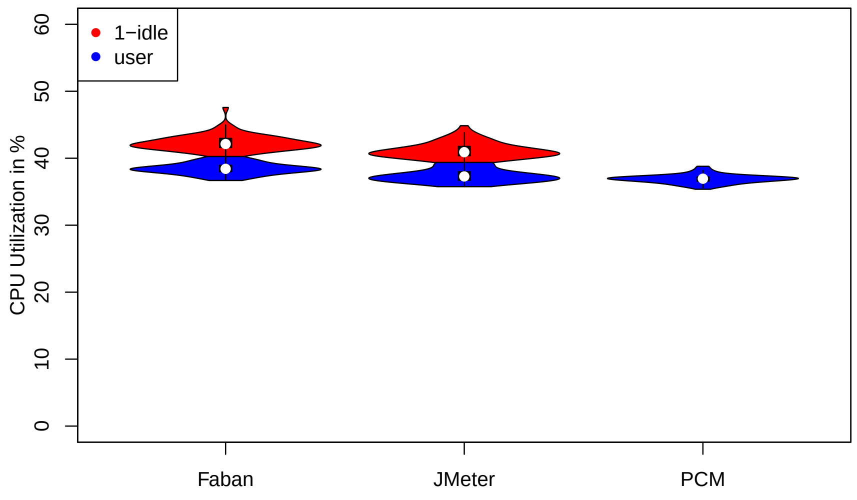
(d) Summary statistics

Figure 7.20: *1200U-34B/33P/33M:*



(a) Relative counts (common to JMeter, Faban and PCM)

Figure 7.21: *1200U-34B/33P/33M: Request count statistics*



(a) Violin plots

	Mean	$\pm$ $CI_{0.95}$	*Stddev.*	*Median*	*N*
Faban (1-idle)	42.28	$\pm$ 0.27	1.16	42.12	72
Faban (user)	38.49	$\pm$ 0.22	0.92	38.42	72
JMeter (1-idle)	41.17	$\pm$ 0.27	1.16	40.91	72
JMeter (user)	37.46	$\pm$ 0.24	1.04	37.28	72
PCM	36.91	$\pm$ 0.15	0.64	36.95	72

(b) Summary Statistics

Figure 7.22: *1200U-34B/33P/33M: CPU utilization statistics*

removed from the shopping cart than items have previously been added to the shopping cart.

Thus, we manually modify the Purchase transaction in a way that it represents a more challenging scenario. In order to achieve this, we added artificial sessions to the session log extracted with 800 users (50% B, 25% M, 25% P). These sessions contain adapted sequences of user requests. During these user requests the *purchase cart* request is never called when the shoppingcart is empty. These sessions comprise three new transitions: from *view_item_quantity* to *shoppingcart*, from *shoppingcart* to *purchase cart*, and from *shoppingcart* to *defer order*. Afterwards, we generated a new Wessbas-DSL instance, again including the learning of GaAs and the calculation of the conditional probabilities (see Section 7.4.5). The resulting *Modified Purchase (MP)* transaction can be found in Figure 7.12. As the GaAs do not have any impact on the Browse and Manage transactions, we set the proportion of Modified Purchase to 100%. Afterwards, a new JMeter Test Plan is generated.

To validate the impact of the automatically learned GaAs, we execute two experiments with JMeter:

1. *withGAA*: The workload is executed with the new settings using GaAs and calculated conditional probabilities (see Section 7.4.5).
2. *withoutGAA*: The guards and action are removed and the originally measured transition probabilities are included. Then, the experiment is executed again.

Request Counts.

The request counts of the experiments are depicted in Table 7.9. The relative frequencies of the requests are almost exactly the same for the two experiments *withGAA* and *withoutGAA*. This indicates that the combination of GaAs and conditional probabilities lead to the same request count distribution.

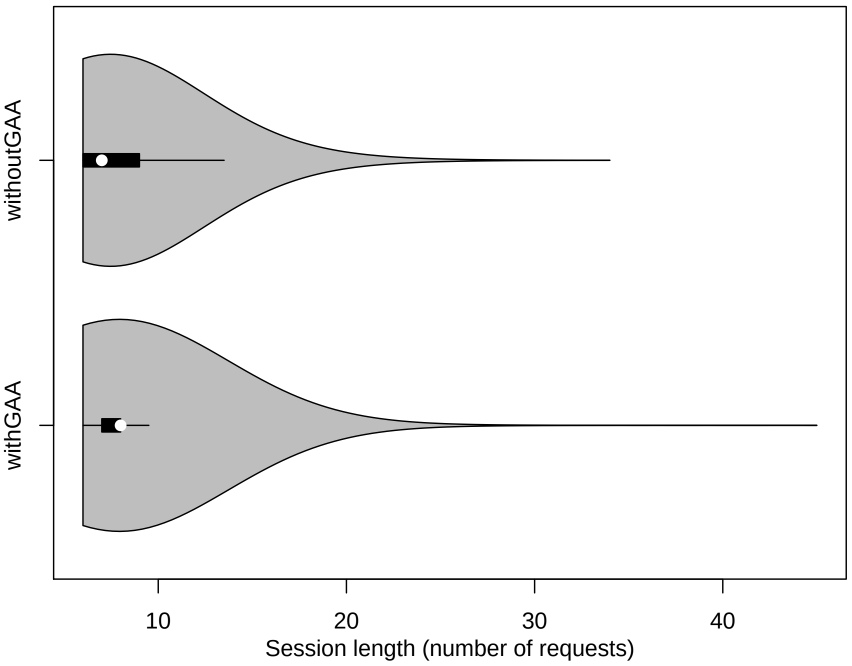
We also included the request counts that would result when experiment *withGAA* is executed using the originally measured transition probabilities (*withGAA (OP)*) and not the conditional probabilities. In this case, the request counts would be considerable different from the originally measured request counts. Especially, the proportion of requests to *add to cart* is with 21.9% considerably different from the originally measured proportion (16.2%). The results of this experiment emphasize that the calculation of conditional probabilities is required.

Session-Based Metrics.

As we can see in Figure 7.23, the mean session length of *withGAA* match with 7.92 exactly the mean session length of *withoutGAA*. The main difference is the number of distinct sessions. The experiment *withGAA* resulted in 182 distinct sessions and *withoutGAA* resulted in 333. This can be explained because in *withoutGAA* invalid sequences like *login*, *view items quantity*, *shopping cart*, *purchase cart* occur quite often. The GaAs prevent that a *purchase cart* request is called without having called *add to cart* before.

Table 7.9: *800U-0B/100MP/0M: Request count statistics (JMeter)*

	Request	*withGAA*		*withoutGAA*		*withGAA (OP)*	
1	add to cart	38,596	0.162	38,720	0.162	57,762	0.219
2	clear cart	1,188	0.005	1,224	0.005	946	0.004
3	defer order	8,571	0.036	8,582	0.036	9,682	0.037
4	home	30,079	0.126	30,251	0.126	30,048	0.114
5	login	30,079	0.126	30,251	0.126	30,048	0.114
6	logout	30,079	0.126	30,251	0.126	30,048	0.114
7	purchase cart	21,508	0.090	22,137	0.092	20,212	0.077
8	remove	155	0.001	168	0.001	122	0.000
9	shopping cart	21,923	0.092	21,669	0.090	17,064	0.065
10	view items quantity	56,191	0.236	56,328	0.235	68,629	0.260

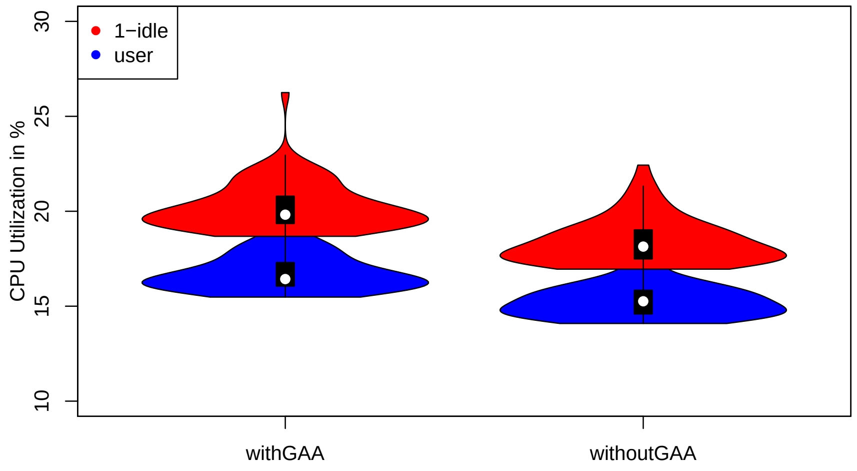


(a) Violin plots for session lengths

	Min.	Q_1	*Med.*	*Mean*	$CI_{0.95}$	Q_3	*Max.*	*N*
withoutGAA	6	6	7	7.92	[7.89, 7.95]	9	34	30,251
withGAA	6	7	8	7.92	[7.90, 7.95]	8	45	30,079

(b) Summary statistics of session lengths

Figure 7.23: *800U-0B/100MP/0M: Session length statistics for withGAA compared to withoutGAA*



(a) Violin plots

		Mean	$\pm$ $CI_{0.95}$	*Stddev.*	*Median*	*N*
	JMeter (1-idle)	20.26	± 0.30	1.26	19.82	72
withGAA	*JMeter (user)*	16.81	± 0.27	1.13	16.43	72
	JMeter (1-idle)	18.40	± 0.28	1.18	18.14	72
withoutGAA	*JMeter (user)*	15.42	± 0.26	1.09	15.26	72

(b) Summary Statistics

Figure 7.24: *800U-0B/100MP/0M: CPU utilization statistics*

CPU Utilization.

Figure 7.24 illustrates that the mean overall CPU utilization of *withoutGAA* generated by JMeter decreases by 9.18% from 20.26% *withGAA* to 18.40%. This can be explained as in the experiment *withoutGAA* the action *purchase cart* is often called when no items are in the shopping cart.

Server-side Response Times.

The results of the server-side response times (see Figure 7.25) confirm the results so far. The response times of the request type *defer order* and *purchase cart* are higher in the experiment *withGAA*. Especially, the mean response time of *purchase cart* increased significantly from 10.5 ms *withGAA* to 14.7 ms *withoutGAA*.

To summarize, the GaAs can have a high impact on performance evaluation results, depending on the control flow of the user actions. Using the conditional probabilities in combination with the GaAs the workload characteristics are similar to the originally measured workload characteristics. Only the number of distinct sessions is lower as invalid user sequences are not possible. We evaluated the impact of the GaAs only against JMeter as the performance model generator only uses average CPU values per request type and does not consider parametric dependencies like the number of items in the shopping cart. This results in the fact that the simulated CPU demands are the same regardless of whether items are in the shopping cart or not. Therefore, using a PCM model that considers this parametric dependencies, would result in similar results.

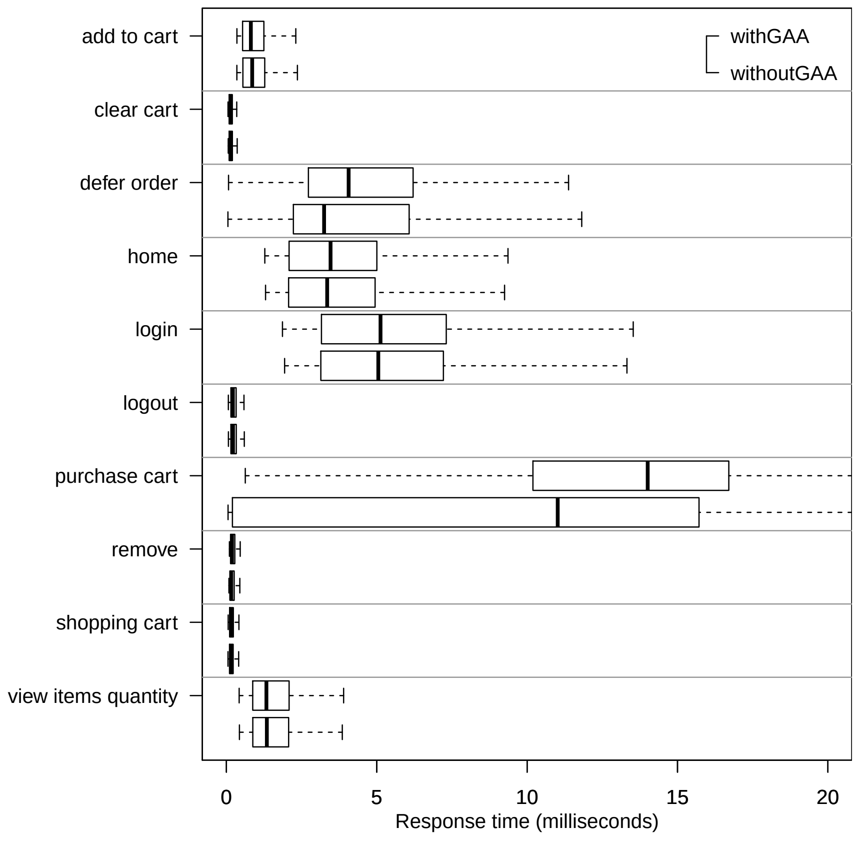


Figure 7.25: *800U-0B/100MP/0M: Server-side response time statistics*

7.7.5 Threats to Validity

A threat to external validity (Wohlin et al., 2012) is that we only selected one common load generator tool and one architecture-level performance evaluation tool for the evaluation. We claim that we can use the WESSBAS-DSL for other performance evaluation tools as well, which enable the specification of probabilistic workloads and GaAs. It might be that extensions of these tools are required, as described in the case of JMeter. In our future work, we will evaluate the use of other tools as well.

Another threat to validity is that we modified the dealer driver of SPECjEnterprise2010 whereby we assigned (see Section 7.7.3.2), exactly one session to a transaction. In real world applications, users will usually not behave in this way. Instead, users will execute multiple transactions in one session, they will leave sessions without logging out of the system, or they take long breaks between user actions and reach session time outs. However, the way we split the transactions into sessions, we are able to evaluate the impact of different clustering settings on the accuracy of the results. This way we found out that X-means is easier to use than K-means and that NED is better suited to identify different transactions than ED. To overcome this threat, we also applied the clustering setting with the lowest classification error to the non-synthetic access logs of the World Cup 1998 Web site. Hence, we could show that the WESSBAS approach can also be applied to real world application.

A threat to construct validity is that our selected workload settings do not drive the SUT in overload situations. We chose moderate CPU utilization between 30% and 50% as in many production systems CPU utilizations are often in this moderate load situation.

To ensure conclusion validity we used multiple statistical metrics like absolute counts, relative proportions, means, medians, and standard deviations. Furthermore, we used violin and bar plots to visualize the distribution of the measurement results. As in the example in Figure 7.13(c), the mean values can be similar but the deviations significantly differ from each other.

As we introduced the concept of GaAs in the workload model, the memoryless property of the Markov chains is lost. Therefore, to ensure that the average behavior extracted from the session logs is kept, we calculated and added conditional probabilities to the Behavior Models. In the evaluation of Section 7.7.4.6 we demonstrated that this behavior can be preserved for the SPECjEnterprise2010 workload. However, in future work it must be evaluated if this is generally applicable for all workload types.

7.7.6 Assumptions and Limitations

During our experiments a performance model generator is used (Brunnert/Vögele/Krcmar, 2013) to create the system-specific parts of the performance model in an automated way. We were able to use this generator as it is designed for generating performance models for Java EE applications. Furthermore, the prediction accuracy of the generated model has previously been evaluated. This type of generator is not available for all session-based systems and performance models. Alternatively, the system-specific part must be modeled manually.

Within the Workload Model we assume that the loop counts follow a geometric distribution whenever a loop in a session is modeled using a memoryless loop exit transition (such as the number of *view_items* requests in the *Browse* transaction type (see Figure 7.12)). Therefore, the distribution does not necessarily match to the distribution measured in the log files. Alternatively, for each possible loop within a Behavior Model the distribution would need to be determined and integrated into the Workload Model. This would improve the accuracy of the Workload Model but would make it more complex as there can be many different loops within a Behavior Model. It is also much more complex to consider the distribution for each loop during the transformation to performance evaluation tools. These tools must be able to handle different distributions. Furthermore, when distributions other than the geometric distribution should be used, these must be modeled explicitly. As future work, the effect of different distributions for loop counts on the workload and performance characteristics would be interesting to investigate.

One limitation of our approach is that still manual effort is needed to create the Wessbas-DSL instances and executable load tests (see Figure 7.1). This includes the identification of use cases during session log creation (Section 7.4.1), the handling of generated parameter values during the test case creation (Section 7.5), and, if required, the examination of preconditions (Section 7.7.4.4) to prevent inaccurate user behavior. However, using our approach, the effort to extract workload specifications and to generate load tests is significantly reduced. In the example of SPECjEnterprise2010 we only needed to add five regular expression extractors to the JMeter Test Plan to extract required parameter values generated during load generation. Furthermore, we had to create a mechanism to store the items added to the *shopping cart* in order to know which items can be removed in the *remove* action.

The effort for a user to adopt our approach is low when the performance evaluation tools Apache JMeter or PCM are used. The required log files can be also extracted using common monitoring tools or HTTP request logs from web servers. When other tools are used first new transformations of the WESSBAS-DSL to these tools must be implemented.

Another limitation of our approach is that the order of events and the minimum and maximum number of executions is not controlled using probabilistic workloads. As we can see in Section 7.7.4.4, the number of items in the shopping cart has a high impact on the response times of the purchase cart action.

7.8 Conclusion and Future Work

The specification and generation of representative workloads is a core task for many performance evaluation activities. However, obtaining representative workload specifications is still a big challenge. In response to this challenge, we present our WESSBAS approach for the systematic extraction and specification of probabilistic workloads for session-based systems. We also include transformations to the load testing tool Apache JMeter and to the performance model PCM. To address the challenge of specifying workloads for different performance evaluation tools, we first introduced a domain-specific language that describes the structure of a workload in a generalized way. We demonstrated how groups of customers with similar behavioral patterns can be identified using clustering algorithms. Furthermore, inter-request dependencies are learned in an automatic way and conditional probabilities are calculated. This is the first approach to present a holistic process from runtime data to the executable load tests and performance predictions.

The evaluation with the industry-standard benchmark SPECjEnterprise2010 and the World Cup 1998 access logs demonstrated the practicality and high accuracy of the proposed approach. The session-based characteristics, like session length and the number of distinct sessions, deviate from the measured logs in case of SPECjEnterprise2010. However, using the non-synthetic World Cup logs, the session-based characteristics are similar as well. The invocation frequencies for requests match with almost 100%. Furthermore, performance characteristics in terms of CPU utilization, response times and heap usage are, with a few minor exceptions, similar to the original executed workload. The approach is applicable for all session-based systems and requires no detailed knowledge about workload extraction.

In our future work, we will investigate the prioritization and selection of load test cases using the generated performance models (Vögele et al., 2014). Moreover, we plan to implement the transformation between the WESSBAS-DSL instances and PCM in a bidirectional way. The advantage of testing Wessbas-DSL instances and PCM in a bidirectional way is that the test cases are analyzed and selected within PCM and corresponding load test scripts can be generated using the Wessbas-DSL. Furthermore, we plan to implement the transformation from Wessbas to PCM in a transformation language such as Henshin[26], as it additionally provides tools to verify the transformation correctness. Moreover, we plan to integrate approaches for the generation of varying workload intensities (v. Kistowski/Herbst/Kounev, 2014).

[26]http://www.eclipse.org/henshin

7.9 Appendix

Table 7.10: *Resulting Guards and Actions*

From State	**To State**	**Guards**	**Actions**
login	View items	login	N/A
login	inventory	login	inventory = true
login	View items quantity	login	View_Items_quantitydeferorder +1; View_Items_quantitypurchasecart +1; View_Items_quantityshoppingcart +1; View_Items_quantityclearcart +1; View_Items_quantityremove +1
View items quantity	Add_to_Cart	N/A	Add_to_Cartdeferorder +1; Add_to_Cartpurchasecart +1; Add_to_Cartshoppingcart +1; Add_to_Cartclearcart +1; Add_to_Cartremove +1
Add_to_Cart	defer order	login && View_Items_quantitydeferorder >0 && Add_to_Cartdeferorder >0	View_Items_quantitydeferorder -1; Add_to_Cartdeferorder -1
Add_to_Cart	purchasecart	login && View_Items_quantitypurchasecart >0 && Add_to_Cartpurchasecart >0	View_Items_quantitypurchasecart -1; Add_to_Cartpurchasecart -1
Add_to_Cart	shoppingcart	login && View_Items_quantityshoppingcart >0 && Add_to_Cartshoppingcart >0	View_Items_quantityshoppingcart -1; Add_to_Cartshoppingcart -1;
Add_to_Cart	View items quantity	login	View_Items_quantitydeferorder +1; View_Items_quantitypurchasecart +1; View_Items_quantityshoppingcart +1; View_Items_quantityclearcart +1; View_Items_quantityremove +1
shoppingcart	clearcart	login && View_Items_quantityclearcart >0 && Add_to_Cartclearcart >0	View_Items_quantityclearcart -1; Add_to_Cartclearcart -1
shoppingcart	remove	login && View_Items_quantityremove >1 && Add_to_Cartremove >1	View_Items_quantityremove -1; Add_to_Cartremove -1
remove	defer order	login && View_Items_quantitydeferorder >0 && Add_to_Cartdeferorder >0	View_Items_quantitydeferorder -1; Add_to_Cartdeferorder -1
remove	purchasecart	login && View_Items_quantitypurchasecart >0 && Add_to_Cartpurchasecart >0	View_Items_quantitypurchasecart -1; Add_to_Cartpurchasecart -1
remove	shoppingcart	login && View_Items_quantityshoppingcart >0 && Add_to_Cartshoppingcart >0	View_Items_quantityshoppingcart -1; Add_to_Cartshoppingcart -1
clearcart	View items quantity	N/A	View_Items_quantitydeferorder+1; View_Items_quantitypurchasecart+1; View_Items_quantityshoppingcart+1; View_Items_quantityclearcart +1; View_Items_quantityremove +1
inventory	sellinventory	login && inventory	N/A
inventory	cancelorder	login	N/A
inventory	inventory	login	inventory = true
cancelorder	inventory	N/A	inventory = true
sellinventory	home	login	N/A
sellinventory	sellinventory	login && inventory	N/A
View items	home	login	N/A
View items	View items	login	N/A
Additionally for RQ5			
View items quantity	shoppingcart	login && View_Items_quantityshoppingcart >0 && Add_to_Cartshoppingcart >0	View_Items_quantityshoppingcart-1;Add_to_Cartshoppingcart-1
shoppingcart	purchasecart	login && View_Items_quantitypurchasecart >0 && Add_to_Cartpurchasecart >0	View_Items_quantitypurchasecart-1;Add_to_Cartpurchasecart-1
shoppingcart	defer order	login && View_Items_quantitydeferorder >0 && Add_to_Cartdeferorder >0	View_Items_quantitydeferorder-1;Add_to_Cartdeferorder -1

Chapter 8

Modeling Complex User Behavior with the Palladio Component Model

Authors	Vögele, Christian[1] (voegele@fortiss.org) Heinrich, Robert[2] (robert.heinrich@kit.edu) Heilein, Robert[4] (heilein@in.tum.de) van Hoorn, André[3] (andre.van.hoorn@acm.org) Krcmar, Helmut[4] (krcmar@in.tum.de) [1]fortiss GmbH, Munich, Germany [2]Karlsruhe Institute of Technology, Karlsruhe, Germany [3]Institute of Software Technology, University of Stuttgart, Stuttgart, Germany [4]Chair for Information Systems, Technical University of Munich (TUM), Garching, Germany
Outlet	Symposium zur Software-Performance 2015 (SSP 2015)
Status	Accepted
Individual Contribution	Problem and scope definition, construction of the conceptual approach, experiment design, execution and result analysis, paper writing, paper editing

Table 8.1: *Bibliographic details for P5*

Abstract The specification of workloads is required in order to evaluate performance characteristics of application systems using performance prediction approaches like the Palladio Component Model (PCM). One of the biggest challenges in workload modeling is to ensure that the modeled user behavior adequately resembles the real user behavior. However, PCM offers limited support to model such complex user behavior. Furthermore, reusing modeled activities is not possible. To overcome these limitations, workarounds are required. In order to avoid these workarounds, we extend the meta-model of the PCM Usage Model. We evaluate the extended PCM Usage Model by integrating it into our previous work on automatic extraction of workload specifications. Based on HTTP web logs, recorded from the standard industry benchmark SPECjEnterprise2010, instances of a domain-specific language (DSL) for modeling workload specifications are extracted. Afterwards, these instances are transformed to the extended PCM Usage Model. The evaluation shows that workload characteristics of the simulated workload match the measured workload with high accuracy.

8.1 Introduction

In order to evaluate the performance of application systems using model-based prediction approaches like the Palladio Component Model (PCM)(Becker/Koziolek/Reussner, 2009), the modeling of workloads is required. Workloads describe the user behavior and workload intensities in terms of the number of requests to the system under test (SUT) (Goševa-Popstojanova et al., 2006). One of the biggest challenges in workload modeling is, that these models must be representative compared to the real workload (Feitelson, 2015). This is especially important to ensure that the predicted performance in terms of response times, resource utilization, and throughput using performance models match the measured performance with high accuracy.

Within PCM, workloads are modeled with the Usage Model. The Usage Model is a domain- specific modeling language allowing to specify workload intensities (i.e., the number of concurrent users), user behavior (i.e., the control flow graph of user system calls), and parameters passed with the system calls (Becker/Koziolek/Reussner, 2009). However, modeling complex user scenarios with the Usage Model has limitations, which makes the workload modeling often difficult or even unfeasible. In response to these limitations we extend the Usage Model.

These extensions result in several advantages. First, the modeling of realistic and complex user behavior is possible. Thus, all kind or usage flows extracted from running applications can be modeled without using workarounds like applied in van Hoorn et al. (2014). Second, modeling of business processes (BP) is enabled. BPs are a set of one or more linked activities where each activity itself is composed of one ore more linked steps (WFMC Terminology, 1999). Steps are either performed completely by a human actor or performed completely by an information system (IS) (Heinrich et al., 2015). By introducing reusability concepts, BP activities must only be modeled once and can then be reused by other activities.

To summarize, the contribution of our proposed approach comprising the following elements: *(i.)* The extension of the PCM Usage Model, including *(ii.)* the evaluation using the WESSBAS approach (van Hoorn et al., 2014; Vögele/van Hoorn/Krcmar, 2015) against the industry-standard benchmark SPECjEnterprise2010. WESSBAS introduces a domain-specific language (DSL) for modeling workload specifications, an automatic extraction of DSL instances from session logs and a transformation from this DSL into load test scripts and performance models.

8.2 Limitations of PCM Usage Model

The Usage Model meta-model (PCM Version 3.4.1) can be found in Figure 8.2. The dashed rectangles represent the new elements and are explained in the next section. We use the example of the SPECjEnterprise2010 purchase transaction (see Figure 8.1) to explain the limitations. SPECjEnterprise2010 is a Java EE industry benchmark representing an application of an automobile manufacturer whose main users are automobile dealers.

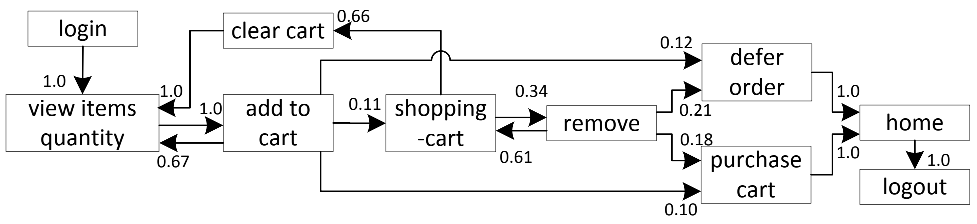


Figure 8.1: *Probabilistic representation of the SPECjEnterprise2010 purchase transaction type*

The Orders domain of this benchmark represents a web-based e-commerce application and enables customers purchasing and selling cars (Purchase), managing their accounts and inventory (Manage), and browsing the catalogue of available cars (Browse). Within the purchase transaction, orders are placed and immediately purchased or deferred. The shopping cart is either cleared or items are removed one by one.

The following limitations can be observed. First, an *AbstractUserAction* can only have zero or one successor and zero or one predecessor (see Figure 8.2). This prevents the modeling of usage behavior, which is representative compared to the real usage behavior. *(i.)* There is no possibility to model backward-edges like from *view items quantity* to *add to cart*. *(ii.) Loops* can be modeled using the *Loop* element. A *Loop* element is a container, within the elements are looped as often as specified. However, when more than one edge leaves the loop, the *Loop* element cannot be used. For instance, the loop *view items quantity*, *add to cart*, *shoppingcart*, *clear cart* cannot be modeled using this *Loop* element, as more than one option is available to leave the loop, e.g., from *add to cart* to *purchase cart* and from *shoppingcart* to *remove*. *(iii.) Branches* are containers as well comprising of multiple *BranchTransitions*. Elements from one *BranchTransition* cannot be linked to elements of another *BranchTransition*. For example, the transitions from the user action *remove* to *purchase cart* or *defer order* cannot be modeled as these actions reside in different branches.

Second, a *UsageScenario* cannot be called by another *UsageScenario*. Within a Usage Model, multiple *UserScenarios* can be modeled. However, there is no possibility that a *UserScenario* calls another *UserScenario* as they are running independent from each other, specifying their own workloads. Therefore, usage flows (c.f. activities) cannot be modeled once and reused by other activities, which is especially important in BP modeling.

Third, only probabilistic *BranchTransitions* can be specified. Thus, the concept of guards and actions (GaA) to control the usage flow cannot be applied. A guard is a condition which must hold true in order to enable a transition. If the transition is executed, an action can change the value of a guard variable (Shams/Krishnamurthy/Far, 2006). GaA can have an impact on the length of a simulated user session or on the number of simulated requests. For example, a *purchase* action can only be executed when items are added to the *shopping cart* before. Thus, the concept of probabilistic conditions, which is a combination of probabilistic and guarded *BranchTransitions*, must be introduced.

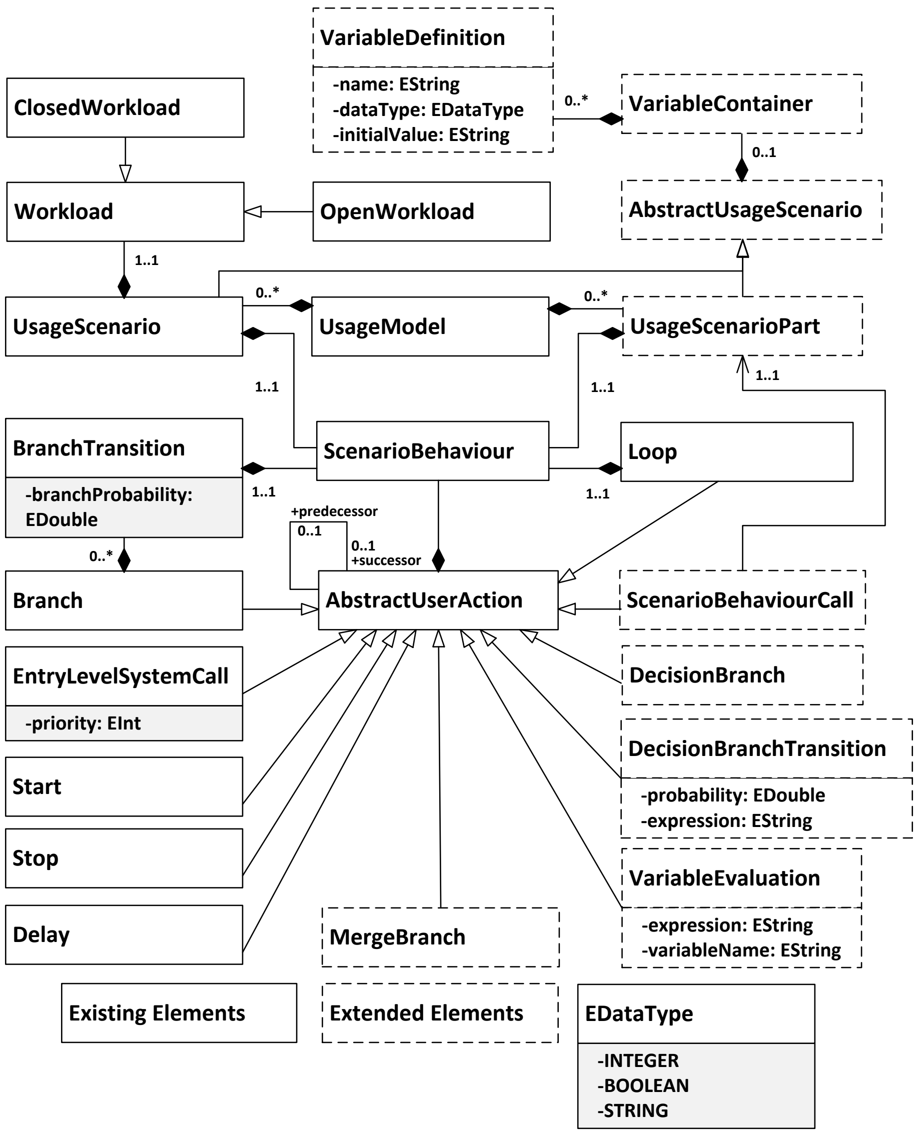


Figure 8.2: *Extended PCM Usage Model meta-model*

8.3 Extension of PCM Usage Model

In this section, the proposed new elements of the PCM Usage Model (dashed rectangles) are explained.

Modeling Complex User Behavior: In order to model representative user behavior the limitation of having only one successor and respectively one predecessor must be overcome. Therefore, we introduce two new *AbstractUserActions*: *MergeBranch* and *DecisionBranch*. A *MergeBranch* has one or many incoming edges and only one outgoing edge. Whereas a *DecisionBranch* has one incoming edge and one or many outgoing edges. Using these new elements, the *AbstractUserActions Branch* and *Loop* are not required anymore.

Reusability: To enable the modeling of reusable activities, we introduce the elements *UsageScenarioPart* and *ScenarioBehaviourCall*. A *UsageModel* can have multiple *Us-*

ageScenarioParts which are similar to *UsageScenarios*, except that they have no workload definition. Both inherit from *AbstractUsageScenario*. Due to the fact that a *UsageScenarioPart* is a reusable scenario it can be called by other *UsageScenarios* and therefore does not need a workload definition. The *UsageScenarioPart* can be called with support of the new element *ScenarioBehaviourCall*. When a *ScenarioBehaviourCall* is called, it executes the linked *UsageScenarioPart* until a *Stop* action is reached. Afterwards, it continues with the superior *UsageScenario*.

Probabilistic Conditions: To model probabilities and GaA, each outgoing edge from the *DecisionBranch* has a new superordinate element called *DecisionBranchTransition*. Within this element a probability must be set. Additionally, a guard condition can be specified. In case the guard is false the edge will be ignored and the probabilities of the other edges are extrapolated to one. Thus, the probabilities are dynamically calculated during runtime. To set an action, the *VariableEvaluation* element must be integrated into the usage flow. The expression will be evaluated and the result is written to the variable defined in the field *variableName*. These variables must be defined in the *VariableContainer*, which is a container for variables used within a *UsageScenario*. Within the *VariableContainer*, multiple variables can be specified with the *VariableDefinition* element. This element defines variables with the attributes *name*, *dataType* (Integer, Boolean or String), and *initialValue*.

8.4 Evaluation

In this section, the accuracy of the extended Usage Model is evaluated. We first extracted standard HTTP web logs from a running SPECjEnterprise2010[27] deployment. The benchmark run was executed with 800 users, a duration of twelve minutes (720 seconds), three minutes ramp up and ramp down phase, and the original benchmark transaction mix (25 % Purchase, 50 % Browse, and 25 % Manage). Afterwards, we used the WESSBAS approach to generate instances of a domain-specific language (DSL) for modeling workload specifications based on these web logs (van Hoorn et al., 2014). Then, we modified the transformation explained in (Vögele/van Hoorn/Krcmar, 2015) to generate workload specifications using the extended PCM Usage Model. We generated the Usage Model once with and once without GaA.

The accuracy of the extracted workload specification are evaluated by comparing the number of simulated requests for the different HTTP request types with the originally measured request counts to the SUT. The result of the measured request counts (MRC) and simulated request counts (SRC) per HTTP action can be found in Table 8.2. Further, for each simulation run the relative prediction error (PE) of the SRC compared to the MRC is given.

[27]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.

Table 8.2: *Evaluation Results*

		Orig.	without GaA		with GaA	
	Request	MRC	SRC	PE%	SRC	PE%
1	add to cart	21,376	20,766	2.94%	21,490	0.53%
2	cancel order	342	350	2.29%	285	20.00%
3	clear cart	2,043	2,005	1.90%	2,194	6.88%
4	defer order	2,273	2,237	1.61%	2,249	1.07%
5	home	19,409	19,039	1.94%	19,009	2.10%
6	inventory	19,960	19,452	2.61%	19,609	1.79%
7	login	19,913	19,514	2.04%	19,527	1.98%
8	logout	19,194	18,838	1.89%	18,812	2.03%
9	purchase cart	2,811	2,716	3.50%	2,728	3.04%
10	remove	947	901	5.11%	736	28.67%
11	sell inventory	43,375	42,741	1.48%	42,089	3.06%
12	shopping cart	2,991	2,906	2.92%	2,932	2.01%
13	view items quantity	21,300	20,706	2.87%	21,408	0.50%
14	view items	67,886	66,518	2.06%	65,112	4.26%
	$\sum$	243,820	238,689	2.15%	238,180	2.37%

The evaluation shows that the simulated request counts match the measured request counts with high accuracy. The maximum prediction error without GaA is 5.11% for the request type *remove*. With GaA the prediction errors are slightly higher. The maximum prediction error is again for the request type *remove* with 28.67%. This was expected as GaA do not allow the execution of *remove* when there are no items in the shopping cart anymore.

8.5 Related Work

We group the related work into approaches for modeling complex user behavior and into approaches for the extraction of workload specifications based on system traces. Due to space limitations we give representative examples.

Modeling complex user behavior: An approach for modeling complex user behavior from a business process perspective is proposed by Heinrich et al. (2015). The PCM Usage Model meta-model has been extended by user behaviors from a business process perspective. These behaviors can also be none system interactions, like e.g. a user mixing chemicals. In contrast, the proposed meta-model extension in this paper puts focus on modeling complex usage flows.

Extraction of workload specifications based on system traces: The iObserve approach exploits observed method traces for generating the states and transitions of behavioral models and the corresponding usage intensity (Heinrich et al., 2014). Further, another approach for the automatic generation of PCM workload specifications from log files can be found in Vögele/van Hoorn/Krcmar (2015). Due to the limitations of the Usage Model (see Section 8.2) large parts of the workload specification are modeled within the PCM Repository Model. The evaluation of this approach showed, that the prediction results match the measured workload with high accuracy. However, the clear separation of concerns of PCM is violated.

8.6 Conclusion and Future Work

This paper presents an extension of the PCM Usage Model in order to model complex user behavior. Additionally, we introduce a concept to reuse activities, which is a key requirement for business process modeling. The evaluation using WESSBAS-DSL instances extracted from standard HTTP web logs of the Java EE benchmark SPECjEnterprise2010 demonstrates, that PCM workload specifications can be generated, which match the measured workload with high accuracy. As future work, we plan to enable the modeling of asynchronous communications and session abandonments.

Chapter 9

Multi-Objective Optimization of Load Test Designs using Performance Models

Authors	Vögele, Christian[1] (voegele@fortiss.org) Krcmar, Helmut[2] (krcmar@in.tum.de) [1]fortiss GmbH, Munich, Germany [2]Chair for Information Systems, Technical University of Munich (TUM), Garching, Germany
Outlet	ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)
Status	Submitted
Individual Contribution	Problem and scope definition, construction of the conceptual approach, prototype development, experiment design, execution and result analysis, paper writing, paper editing

Table 9.1: *Bibliographic details for P6*

Abstract Load tests are designed by selecting workloads that validate if given performance objectives are met. Only after the execution and analysis of load test results it can be assessed if the given load test design is able to fulfill the objectives. This process often requires several iterations as a large number of different workload alternatives exists. This results in high costs and a high duration of the load testing process. In response to this challenge, this work proposes the use of performance models before load test execution to derive suitable workloads that achieve (possibly multiple) given performance objectives. First, workload specifications are extracted in an automatic way from an application system and transformed into a performance model. Second, workload design options and performance objectives are configured by a performance engineer. Third, different workload alternatives are simulated and assessed based on these objectives. A multi-objective optimization using an evolutionary algorithm is applied to evaluate these alternatives. The resulting candidates are presented to the performance engineer to make trade-off decisions. Finally, executable load test scripts are generated based on the selected solution. The feasibility of the approach is validated in a case study using the industry Java EE benchmark SPECjEnterprise2010.

9.1 Introduction

Due to the fact that an increasing amount of business transactions are supported by application systems the development of fast and reliable systems is required. These applications have to satisfy non-functional performance requirements like given throughput and response times. To validate whether these requirements can be met, the performance must be continuously evaluated.

Load testing is a commonly used method to evaluate the performance of applications systems (Avritzer et al., 2002). Many tools exist which enable load generation. However, these tools do not support the selection of appropriate load test designs. Load tests are designed by selecting test cases and input parameter derived from usage scenarios, the proportion by which each test case is executed as well as the number of expected users executing requests. In current practice, test experts derive test designs manually by for instance selecting a set of the most probable test cases and by estimating the workload intensity (Meier et al., 2007). This approach is time consuming and error prone (Shams/Krishnamurthy/Far, 2006) and can lead to totally useless load testing results.

Many tools exist which enable load execution. However, these tools do not support the selection of appropriate load test designs. Load tests are designed by selecting test cases and input parameter derived from usage scenarios, the proportion by which each test case is executed as well as the number of expected users executing requests. In current practice, test experts derive test designs manually by for instance selecting a set of the most probable test cases and by estimating the workload intensity (Meier et al., 2007). This approach is time consuming and error prone (Shams/Krishnamurthy/Far, 2006) and can lead to totally useless load testing results.

Mainly two different goals are followed when load tests are designed Jiang/Hassan (2015): *(i.)* designing the workload in a way that it is representative compared to the real workload in the field or *(ii.)* designing the workload in a way that it causes load related problems. The WESSBAS[1] approach introduced in our previous work Vögele et al. (2016) enables to extract representative workloads. However, this approach has no mechanism to select load tests based on given performance objectives. There can be many different performance objectives depending on the type of system under test. One common performance objective is to detect workloads that lead to load related performance problems like high response times. Further objectives can be to reduce the number of needed test cases or to generate workloads that saturates the system to predefined utilization levels. Before the execution of load tests and the analysis of the testing results it is difficult to verify whether the selected test design is able to satisfy given testing objectives. It becomes even harder, when multiple testing objectives should be considered at the same time which are conflicting. For example, in a highly distributed software system the data dependencies can be very complex. Therefore, the number of test cases should be reduced as the provisioning of test data for many test cases causes a lot of effort. However, at the same time the chosen test design should represent the real workload as accurately as possible. Such conflicting trade-off decisions are very difficult to solve manually.

[1]WESSBAS is an acronym for *Workload Extraction and Specification for Session-Based Application Systems*

Performance models are a promising solution to support the selection of load test designs before the execution of load tests (Woodside/Franks/Petriu, 2007). Performance models in combination with a simulation engine allow to derive performance predictions for a given workload (Koziolek, 2010), e.g. response times or resource utilization. Thus, the impact of different workloads (respectively load test designs) on the system can be analyzed before load test execution. As the number of possible workload settings to be evaluated is very high and multiple performance objectives should be met simultaneously, we propose the use of a multi-objective genetic algorithm.

The selection of load test designs using performance models comes with several advantages. First, the integration of software development and operations (cf. DevOps) is supported (Brunnert et al., 2015). The extracted workload during operations can be used for measurement-based (Bulej/Kalibera/Tůma, 2005) and model-based (Brunnert/Krcmar, 2014) continuous integration approaches in order to detect performance regressions during development. Furthermore, the approach can be used to continuously select suitable test designs. Using the same test design for each load test can lead to the effect that always the same parts of the system are saturated (Woodside/Franks/Petriu, 2007). Thus, there is the danger that new bottlenecks are not identified. Another advantage is that this approach can be combined with architecture optimization approaches (Koziolek et al., 2011). Architecture solutions that are derived using these approaches can be further evaluated using different workloads. Finally, using our proposed approach performance engineers save time and effort as test designs are derived in an automatic way.

As starting point for the optimization we use a performance model instance modeled with the PCM (Becker/Koziolek/Reussner, 2009). Using PCM allows the generation of different workload candidates in an easy way. We use an architecture-level performance models as it permits to model system architecture, execution environment, and workload specification separately from each other (Brosig/Huber/Kounev, 2011). The performance model instance is automatically extracted from the system under test (SUT) (Brunnert/Vögele/Krcmar, 2013). Furthermore, the workload specification of the performance model will be generated using the Wessbas[29] approach introduced in our previous work (Vögele et al., 2016). The performance engineer can configure how the workload design can be varied and which performance objectives should be achieved. Afterwards, the multi-objective optimization will identify workload candidates fitting best to the configured objectives until a stop criterion is fulfilled. These candidates are presented to the performance engineer who can choose the candidate fitting best to his performance requirements. Finally, based on the selected candidate executable load test scripts are generated. Figure 9.1 gives an overview of the proposed approach, called Load Test Design Selector (within dashed lines), integrated into the Wessbas approach. Details on the Wessbas approach are explained in Section 9.3.

[29] Wessbas is an acronym for *Workload Extraction and Specification for Session-Based Application Systems*

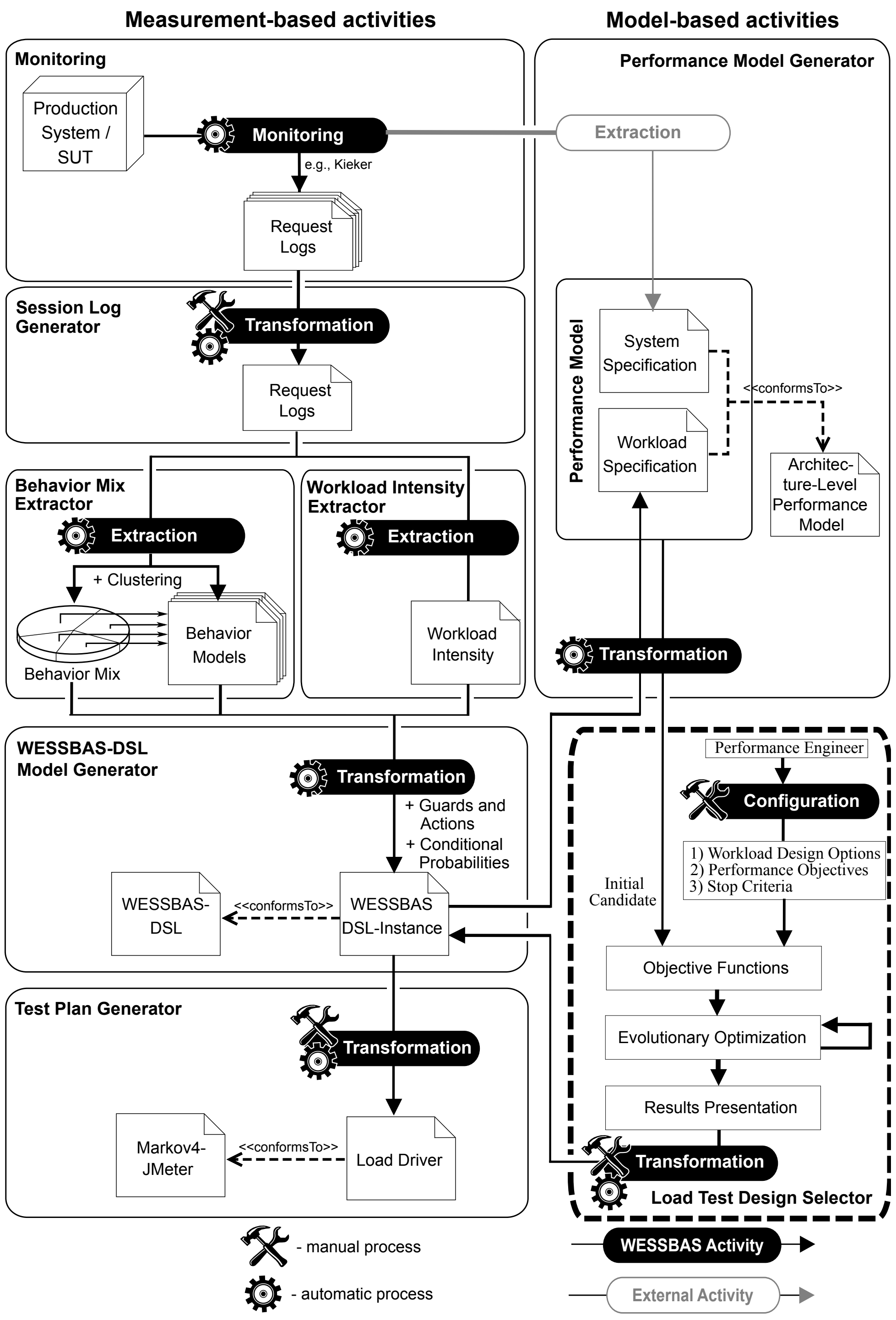


Figure 9.1: *Load Test Design Selector integrated into the* WESSBAS *approach (based on Vögele* et al. (2016)*)*

To summarize, the contribution of this paper comprises the following elements:

1. Approach for automatically deriving load test designs from performance models that optimized for (possibly conflicting) performance objectives using multi-objective optimization techniques..

2. Integration with the WESSBAS approach to automatically deriving performance models and executable load tests from the chosen test design.

3. Comprehensive evaluation against the industry-standard benchmark SPECjEnterprise2010.

To the best of our knowledge this is the first approach that derives executable load tests from runtime data that are optimized to achieve multiple given performance objectives. The developed tools, models, and results of this paper are publicly available online (Vögele/Krcmar, 2016).

The remainder of this paper is structured as follows: Section 9.2 provides an overview of related work. In Section 9.3, the used workload formalism required to understand the paper is introduced. Furthermore, the transformation of these workload specifications to performance models and to load testing tools is presented in this section. The description of the load test design selection process is depicted in Section 9.4. The evaluation of the proposed approach can be found in Section 9.5. Section 9.6 summaries limitations and assumptions of our approach and is followed by the illustration of the threads to validity in Section 9.7. Finally, Section 9.8 details conclusions of our work and presents suggestions for future work.

9.2 Related Work

Performance evaluation (Koziolek, 2010; Becker/Koziolek/Reussner, 2009), load testing (Jiang/Hassan, 2015; Avritzer et al., 2002) and multi-objective metaheuristic optimization (Coello/Lamont/Van Veldhuisen, 2007) are the primary topics of our work. Based on these topics we group related approaches into *(i.)* feedback-based load generation techniques *(ii.)* genetic algorithm techniques, and *(iii.)* architecture optimization techniques using performance models.

9.2.1 Feedback-based Load Generation Techniques

Feedback-based load generation techniques adapt the workload generation dynamically based on the system feedback in order to achieve a given performance objective.

An similar approach to ours that combines performance prediction with load testing is introduced by Barna/Litoiu/Ghanbari (2011a) and Barna/Litoiu/Ghanbari (2011b). This approach first models an application systems as a two-layered queuing model. Then based on this model workload mixes and workload intensities are identified that saturates hardware and software bottlenecks. Bottlenecks are defined as resources where the requests are queued and delayed because the processing capacity limits of that resource. The approach implements a feedback loop by using the results of the executed load test to calibrate the performance model.

FOREPOST (Grechanik/Chen/Xie, 2012) is an approach that uses machine learning techniques to support load testing. It utilizes online monitoring of test executions to classify system inputs by learning from test execution logs. These classes are then used for generating further test inputs that lead to intensive computations. This way the number of test cases is reduced to those tests that are meaningful for the detection of performance problems.

Another approach that used feedback-based learning is proposed by Bayan/Cangussu (2008) and Bayan/Cangussu (2006). A set of inputs is identified which drives the system to the desired level of stress. When the goal is to detect memory leaks important parameters that impact the memory usage are identified. While these approaches focus on the selection of system inputs our approach selects suitable test cases with performance models.

Zhang/Cheung (2002) present an approach that generates test cases for stress testing of multimedia systems. The authors first model the multimedia systems with petri nets. Based on that model all possible user action sequences are generated by conducting reachability analysis. Linear programming is used to identify the sequences of user actions, which can trigger performance problems.

Segall/Tzoref-Brill (2015) propose an approach that uses a combinatorial testing engine in order to generate combinations of levels of resource utilization on the subcomponent of cloud environments. Based on data gathered during test execution of these systems the combinatorial testing engine adjusts the tests cases.

An approach that uses models to find suitable workload candidates for testing elasticity properties of cloud system is proposed by Gambi/Filieri/Dustdar (2013). First, the behavior of the system is modeled as a Labeled Transition System (LTS) where states capture the current resources allocation and transition labels describe the frequency of transitions occurrence during the execution of the test suite. The models are used to find workload candidates that are likely violating the elasticity properties of the system. The LTS are continually extracted and refined based on execution logs.

In contrast, our proposed approach enables to optimize against multiple performance objectives simultaneously using performance models. Also new objectives like the test coverage and the number of used test cases is taken into account. Furthermore, the used performance models and workloads are extracted automatically by combining it with the Wessbas approach (Vögele/van Hoorn/Krcmar, 2015).

9.2.2 Genetic Algorithm Techniques

Genetic Algorithms are search algorithms that mimic the process of natural evolution for locating optimal solutions towards a specific goal.

Di Penta et al. (2007) use genetic algorithm techniques to derive loads that causes SLA or QoS (Quality of Service) requirement violations in service-oriented systems. The authors use a genetic algorithm that is proposed by Canfora et al. (2005), in order to identify risky workflows within a service. The response time for the risky workflows should be as close to the SLA (high response time) as possible.

Briand/Labiche/Shousha (2005) propose to develop a methodology that helps to identify performance scenarios that stress the system in order to force performance failures or SLA violations. It combines the use of external aperiodic events (ones that are part of the interface of the software system under test, i.e., triggered by events from users, other software systems or sensors) and internally generated system events (events triggered by external events and hidden to the outside of the software system) with a genetic algorithm. These approaches derive workloads by direct measurements on real systems and are not using performance models.

9.2.3 Architecture Optimization Techniques using Performance Models

Aleti et al. (2013) provide a comprehensive review on software architecture optimization methods. Several approaches exist that optimize the architecture of a system using performance models, e.g. (Litoiu/Barna, 2013; Koziolek et al., 2011).

An architecture optimization approach which is similar to our approach is presented by Koziolek et al. (2011). The approach searches the design space of possible architecture candidates based on software architectures modeled with the Palladio Component Model. Pareto-optimal candidates are identified against the criteria performance, reliability and cost.

Another approach similar to our approach is presented by Willnecker/Krcmar (2016). The authors propose to use performance models and evolutionary algorithms to find optimal deployment topologies either to minimize response times or to maximize resource utilization. In contrast to our approach we keep the system architecture as it is and evaluate workload candidates.

9.3 Workload Specifications for Load Testing and Model-based Performance Prediction

As manual modeling and specification of workload specifications is very time consuming we integrate the Load Test Design Selector with the Wessbas approach. Thus, we are able to automate the workload specification and extraction for load testing and performance prediction (see Figure 9.1). In the following we briefly introduce the required concepts of the Wessbas approach needed to support the remainder of this paper. We explain the used workload specifications formalism (Section 9.3.1), the extraction and generation of Wessbas-DSL instances (Section 9.3.2), the transformation to performance models (Section 9.3.3), and the generation of executable load tests (Section 9.3.4).

The Wessbas approach specifies and extracts representative workloads for session-based application systems. Based on a DSL, called Wessbas-DSL, system- and tool-agnostic modeling of these workload specifications is enabled. Recorded session logs are used as a basis for the automatic extraction of Wessbas-DSL instances. Different groups of customers showing similar navigational patterns are identified during the creation of these instances using clustering algorithms. Additionally, inter-request dependencies (GaAs) among the execution of requests are automatically learned. These dependencies come from the fact that the execution of requests often depends on the result of previous requests. The combination of probabilities and GaAs requires the calculation of conditional probabilities, which are also determined in an automatic way. Finally, protocol information required to generate executable load tests are integrated into the Wessbas-DSL. Further details on the Wessbas approach can be found in (Vögele et al., 2016; Vögele/van Hoorn/Krcmar, 2015; van Hoorn et al., 2014).

9.3.1 Workload Specification

The workload formalism used in the Wessbas approach builds on work on the generation of probabilistic and intensity-varying workloads for session-based systems (Schulz et al., 2014; van Hoorn/Rohr/Hasselbring, 2008); particularly, the workload modeling formalism that extends the work by Menascé et al. (1999) and (Krishnamurthy/Rolia/Majumdar, 2006). The workload specification (*Workload Model*) consists of the following components:

- An *Application Model*, specifying allowed sequences of service invocations and SUT-specific details for generating valid requests. The Application Model includes a Session Layer and a Protocol Layer. The Session Layer specifies the allowed sequences of service invocations and SUT-specific details to generate valid requests as extended finite state machine (EFSM). The Protocol Layer models the sequence of protocol-level requests to be executed on the real system.
- A set of *Behavior Models*, each providing a probabilistic representation of user sessions in terms of invoked services and think times between subsequent invocations as Markov chains. Transitions between Markov States are labeled with think times and call probabilities.
- A *Behavior Mix*, specified as probabilities for the individual Behavior Models to occur during workload generation. The Behavior Models represent different customer groups, e.g., Behavior Models for heavy users and/or occasional buyers.
- A *Workload Intensity* that includes a function which specifies the (possibly varying) number of concurrent users during the workload generation execution.

Figure 9.2 illustrates an exemplary workload model. During the workload generation process for a SUT, the model is used as follows:

The Workload Intensity specifies the number of active sessions. For each newly created session, the Behavior Mix determines the user type to be emulated next by selecting

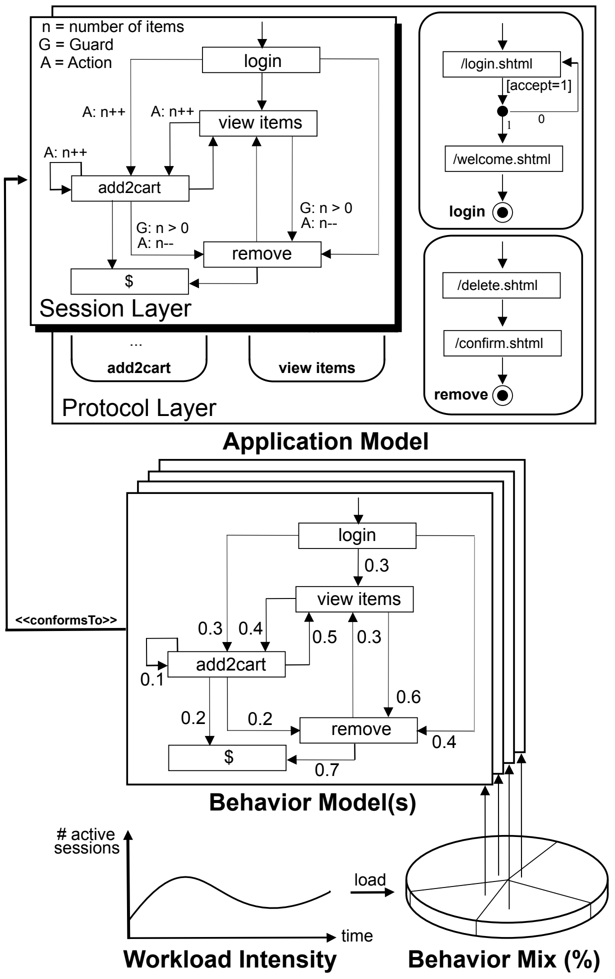


Figure 9.2: *Exemplary Workload Model (without think times in the Behavior Models)* *Vögele* et al. (2016)

the corresponding Behavior Model based on the assigned relative frequencies. In the selected Behavior Model, a probabilistic sequence of services is generated according to the transition probabilities specified in the related Markov Chain. Furthermore, the GaAs of the Session Layer are taken into account in order to generate valid sequences.

Assume that the Behavior Model is currently in the Markov state *view_items* and the current variable n has the value one. First, based on the transitions modeled in the Session Layer from application state *view_items* to the following states, it is validated which guard conditions are satisfied—in the example, the transition from *view_items* to $add2Cart$ and *remove*. As the number of items n is one, the guard $G : n > 0$ to transition *remove* is true. The transition to $add2Cart$ has no guard and can therefore always be executed. Second, based on the transition probabilities, the next transition is chosen—40% of cases to $add2Cart$ and 60% to *remove*. Third, the action(s) on the variable value(s) will be executed. When *remove* is chosen, the value of n is decreased by one; when $add2Cart$ is chosen, the value is increased by one. Finally, the think time is taken for this transition. After the think time has elapsed, the Behavior Model moves to the next state and the service is executed according to the specified EFSM of the Protocol Layer.

9.3.2 WESSBAS-DSL

The Wessbas-DSL follows this workload modeling formalism and denotes a language for expressing such models. In our approach, the Wessbas-DSL is used as an intermediate language between the construction of SUT-specific but tool-agnostic workload models on the one side, and the generation of corresponding inputs to load testing tools and performance models on the other side. Wessbas is implemented as an Ecore-based meta-model using the benefits and tool support of the Eclipse Modeling Framework (EMF) (Steinberg et al., 2009). The DSL structure offers a high degree of flexibility and extensibility.

Wessbas-DSL instances are extracted in an automatic way based on recorded session logs from the SUT. These logs are recorded by monitoring the load (e.g. production load) of the SUT (see Figure 9.1). With support of the *Session Log Generator* the raw logs are transformed to the session log which is the input format for the *Behavior Mix Extractor*. The Behavior Mix Extractor extracts the Behavior Mix and the corresponding Behavior Models based on the created session log. The Behavior Mix is determined by identifying different groups of customers with similar navigational patterns. During the extraction the different Behavior Models are identified with support of clustering algorithms. Currently supported clustering algorithms are K-means and X-means. Additionally, the Workload Intensity is extracted using the *Workload Intensity Extractor*. Finally, the extracted Behavior Models, the Behavior Mix, the Workload Intensity, and the GaAs are transformed to a valid Wessbas-DSL instance, which can be further transformed to performance models (Section 9.3.3) and load generation tools (Section 9.3.4).

9.3.3 Transformation to Performance Models

In order to apply the proposed approach a performance model of the system under test is required. In this work the Palladio Component Model (PCM) will be used as meta-model (Becker/Koziolek/Reussner, 2009). PCM is a modeling language enabling the prediction of quality-of-service attributes (QoS) like response times, CPU utilization, and throughput (Becker/Koziolek/Reussner, 2009). PCM is composed of five complementary model types. The central model type is the *Repository Model*. It models the software components, component operations, and the relations between them. The modeled components are then assembled into a *System Model* to represent the application system. Resource containers (e.g., servers) and their associated hardware resources are modeled in the *Resource Environment Model*, whereas the *Allocation Model* defines the allocation of assembled components to the resource container. The *Usage Model* defines the workload of the system.

As manual modeling requires too much effort we automatically derive the performance models. The system specific parts of the performance model, all models except the usage model, are generated in an automatic way using the approach proposed by Brunnert/Vögele/Krcmar (2013). In this approach PCM instances are extracted from running Java EE applications.

The Usage Model is generated using the WESSBAS approach introduced in our previous work (Vögele et al., 2015). PCM offers limited support to model complex workload specification with the PCM usage model. Furthermore, reusing modeled activities is not possible. To overcome these limitations, workarounds are required. In order to avoid these workarounds, we extended the meta-model of the PCM Usage Model (Vögele et al., 2015). We then adapted the transformation from WESSBAS-DSL instances to PCM Usage Models proposed in Vögele/van Hoorn/Krcmar (2015) to the transformation to the extended PCM Usage Model. The resulting PCM Usage Model consists of the Behavior Models, Behavior Mix, and the Workload Intensity (see also example in Section 9.3.5). The Application Model is merged into the Behavior Models by modeling the GaAs of the Session Layer transitions to the transitions of the Behavior Model. The Protocol Layer is not required in PCM. As simulation engine we use the discrete-event simulation engine SimuCom (Becker/Koziolek/Reussner, 2009). The usage of other simulation engines will be evaluated in our future work.

9.3.4 Generating JMeter Test Plans

A given WESSBAS-DSL instance can be transformed into a corresponding load generation tool. We used the publicly available extension, called Markov4JMeter (van Hoorn/Rohr/Hasselbring, 2008), for the well-known load generator Apache JMeter, which allows us to define and execute the workload specifications defined in Section 9.3.1. JMeter supports the generation of workloads of various types of systems, not limited to Web-based systems.

The *Test Plan Generator* (Figure 9.1), reads a WESSBAS-DSL instance, as described in Section 9.3.2, from file and constructs a further XMI structure, which can be processed by the JMeter tool. The XMI output is generated via the JMeter API and denotes a JMeter-typical tree structure of Test Plan elements, including Markov4JMeter-specific elements, namely *Markov States* and a *Markov Session Controller*, which are provided by the Markov4JMeter add-on for JMeter (van Hoorn/Rohr/Hasselbring, 2008).

9.3.5 Example

In the following example we illustrate the concepts of the proposed approach. First, the approach requires an architecture-level performance model. Though we use PCM as performance model our approach can also be integrated with other architecture-level performance models. A simplified example of a PCM instance in UML-like notation can be found in Figure 9.3. Like explained in Section 9.3.3, the workload specific part is generated using the WESSBAS approach (see Section 9.3.3) and the system-specific part using the approach proposed by Brunnert/Vögele/Krcmar (2013).

The simple exemplary system consists of two servers (Application Server, Database Server) each containing one deployed component and a processing rate of 2.3 GHz. *componentA* consists of four services *login*, *add2cart*, *view_items* and *remove*; *componentB* of the services *getItem* and *getCustomer*. In PCM for each service an abstract behavioral description called service effect specification RDSEFF is specified. RDSEFFs model the abstract control flow of the services. They consist of internal actions (i.e., resource demands accessing the underlying hardware) and external calls (i.e., accessing connected components) (Becker/Koziolek/Reussner, 2009). For reasons of simplicity we do not show the RDSEFF models in our example.

The workload specification of our example system consists of a closed workload, specifying the population size and the think times between the arrivals of new customer (Workload Intensity). We remind that we use the workload specification introduced in Section 9.3.1. Also an open workload specifying the arrival rate of new users could be used. Using the Wessbas approach the usage model is modeled in a way that the first probabilistic branch represents the Behavior Mix. Thus, the call probability to each Behavior Model is represented. PCM enables to vary the workload design options population size respectively arrival rate (Workload Intensity) and the Behavior Mix in an easy way.

Simulating the PCM model using these workload settings results in a set of performance metrics exemplary illustrated in Table 9.2. In this example each metric represents the values which are necessary for the assessment of the configurable performance objectives which will be explained in (Section 9.4.1.2). Changing the workload can lead to totally different prediction results. Thus, the impact of the chosen workload on the performance of the system is difficult to assess without using prediction results or by executing the load test on a test system. As the design space is very huge the performance engineer cannot assess all available combinations.

Workload Design Options	Value
Workload Intensity	800
Behavior Mix	0,5; 0,25; 0,25

Performance Metric	Value
CPU Utilization - App Server [%]	34%
CPU Utilization - DB Server [%]	45%
Response Time - Default Usage Scenario [s]	0.003
Throughput - Default Usage Scenario [req/s]	623
Number of Test Cases [#]	3
Coverage [%]	100%
Representativeness Error [SSE]	156

Table 9.2: *Exemplary PCM Prediction Results*

9.4 Load Test Design Selector

This section explains the *(i.)* selection of load test designs from performance models and *(ii.)* the generation of executable load test cases. To select test designs from performance models in an automatic way, we propose an optimization process using multi-objective optimization techniques. The performance model extracted by the Wessbas approach is used as initial performance model and the process searches for workload candidates that

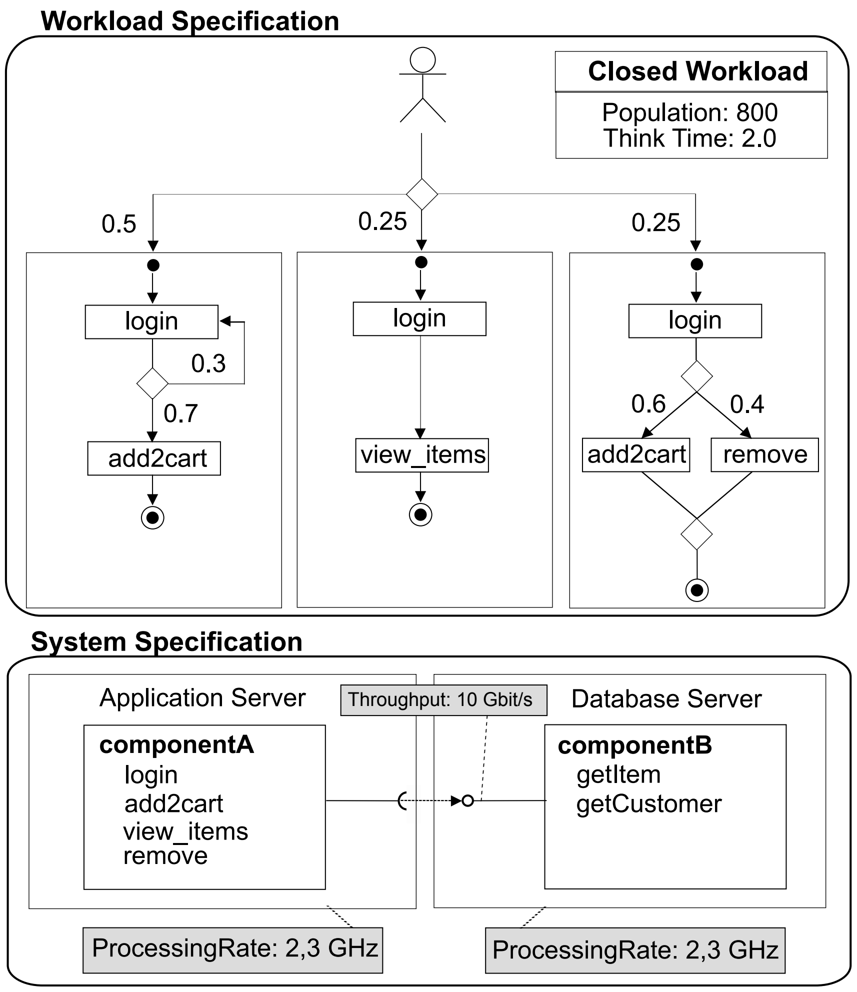


Figure 9.3: *Simplified PCM example without think times and Guards and Actions*

fulfill given performance objectives. From the resulting candidates performance engineers can chose suitable solutions fitting to their trade-off decisions. First, the configuration of the optimization process will be explained (see Section 9.4.1). Afterwards, the evolutionary optimization process will be executed (see Section 9.4.2) and the resulting candidates transformed to executable load test scripts (see Section 9.4.3).

9.4.1 Configuration of Optimization Process

This section explains the configuration of the optimization process. It comprises the configuration of workload design options (Section 9.4.1.1), the testing objectives (Section 9.4.1.2) as well as the stop criteria (Section 9.4.1.3) for the optimization process. This configuration must be specified manually by the performance engineer and will be used during the evolutionary optimization process.

9.4.1.1 Workload Design Options

As explained in Section 9.3.1 a workload in PCM consists of the Behavior Models, Behavior Mix, and the Workload Intensity. As the goal is to find workloads that achieve the performance objectives in an exploratory way the performance engineer can specify which parts of the workload can be adjusted during the optimization process.

In our approach we enable to vary the Workload Intensity and the Behavior Mix of the Workload Model. As the extracted Behavior Models represent the measured user behavior we will not enable to adjust the Behavior Models. In this way the sequence of user actions will be preserved. Further, the Design Space (DS) of the possible workload candidates would also increase considerably. Thus, in our case the degree of freedom is two as we enable to vary the Workload Intensity and the Behavior Mix.

In case the performance engineer is interested in finding workloads saturating specific resources or SLAs (stress tests or scalability tests) the Workload Intensity and the Behavior Mix must be varied as input for the optimization process (Barna/Litoiu/Ghanbari, 2011b). In this case the Workload Intensity must be varied as well as the level of stress has to be determined. Using load levels as expected in the field (operational profile tests) (Avritzer/Weyuker, 1995) only the Behavior Mix must be adjusted as the Workload Intensity should be configured like measured or expected in production.

The DS of all workload combinations can be very high. According to Barna/Litoiu/Ghanbari (2011a) the total number of possible workload combinations when the Workload Intensity and the Behavior Mix can be adjusted is:

$$Mixes = \sum_{n=1}^{n_{max}} \binom{n+b-1}{b-1} \tag{9.1}$$

n is the number of users (Workload Intensity) and b is the number of test cases (Behavior Models). Assuming we have a workload candidate with 200 users (Workload Intensity) and 5 Behavior Models. The number of workload combinations would be 2.872.408.790 (see (Barna/Litoiu/Ghanbari, 2011a)). As the number of combinations is too high to explore (even for relatively simple workloads) we use a multi-objective genetic algorithm approach.

9.4.1.2 Performance Objectives

In this configuration step the performance objectives used to assess the workload design candidates are chosen. As stated before, often more than one criterion should be optimized at the same time which are conflicting with a high probability. We predefined a set of objectives which can be chosen by the test engineer fitting to their testing objectives, whereby at least one criterion must be chosen. For each objective the performance engineer can set the following options:

1. The performance engineer can chose which of the predefined objectives should be considered during the load test design selection. At least one objective must be selected.

2. For each selected objective it must be configured if the objective should be maximized or minimized (Sign of objective). For example, workload candidates which maximize the CPU utilization should be derived.

3. A minimum and/or a maximum threshold for the selected objective can be defined. For example, as the system behavior becomes unpredictable when the CPU is too high (around 80%) the maximum limit should be configured. Further, if the engineer is only interested in workloads which are higher than a predefined threshold the minimum utilization can be configured.

The available performance objectives are explained in the following:

Resource Utilization: The resource utilization is the most important objective. Often it must be evaluated which workload is necessary to stress a specific resource type (Barna/Litoiu/Ghanbari, 2011a; Zhang/Cheung, 2002; Briand/Labiche/Shousha, 2005). Each resource type that is modeled in the used performance model can be chosen as testing objective. The common resources in PCM are CPU, HDD and network. When multiple resource container are modeled in PCM for each container the resource utilization can be configured separately. Currently only the resource type CPU utilization is supported in our approach.

Response Time: Response times can be configured as performance objective as well. When response time SLAs are predefined the question which workloads (Workload Intensity and Behavior Mix) would violate this SLA is of high importance (Barna/Litoiu/Ghanbari, 2011a; Grechanik/Chen/Xie, 2012; Di Penta et al., 2007). The average predicted response time of the simulated system requests will be taken as input for the evolutionary optimization process. Thus, no user think times are taken into account.

Throughput: Throughput can also be an important metric for the selection of workload candidates as SLAs can also include throughput requirements. The number of system level requests initiated by the PCM usage model per second will be taken as input for the evolutionary optimization process.

Number of Test Cases: The number of test cases can have an high impact on the load testing effort. The maintenance effort increases as the test cases must be updated for each new software release. Further, for each test case test data must be provided. The number of test cases is defined as the number of Behavior Models with an execution probability greater than zero.

System Coverage: Chosen workloads can result in very low test coverage and the system can be tested always in the same saturation regime (Woodside/Franks/Petriu, 2007). Therefore, achieving a high test coverage can be an objective of a test design. The system coverage in our work is defined as the percentage of software components which are called by the selected test cases. Each component must be called at least once to be considered. The coverage can be defined on method (Service Effect Specification (SEFF)) or component level (PCM - Basic Component). For each candidate the percentage of called SEFFs or components is calculated.

Representativeness Error: An important requirement is that the workloads should be as representative as possible to the measured workload (Feitelson, 2002; Lutteroth/ Weber, 2008; Draheim et al., 2006). Therefore, one possible test goal is to ensure that the proportion of requests per request type is as similar as possible to the proportion of the initially measured workload. The representativeness error of a workload compared to the measured workload will be determined by calculating the Sum of Squared Error (SSE) for the proportions of the request counts. For each workload candidate the sum of squared error will be calculated as follows:

$$SSE = \sum_{i=1}^{n}(rpm_i * 100 - rps_i * 100)^2 \tag{9.2}$$

rpm_i is defined as the measured proportion of request type (RT) i and rps_i as the simulated proportion of this request type. An example can be found in Table 9.3. The measured request proportion (MRP) and the simulated request proportion (SRP) of two experiments are presented. For each simulated distribution the SSE compared to the measured distribution will be calculated. In the example the request proportion SRP1 is more representative than SRP2 as the SSE value is lower. The lower the value the more representative the executed workload compared to the measured workload.

RT	MRP	SRP1	SSE1	SRP2	SSE2
login	10%	14%	16	15%	25
view_items	20%	16%	16	20%	0
add2cart	20%	14%	36	12%	64
shoppingcart	30%	28%	4	23%	49
home	10%	14%	16	15%	25
logout	10%	14%	16	15%	25
sum	100%	100%	104	100%	188

Table 9.3: *Exemplary calculation of representativeness error*

9.4.1.3 Stop Criteria

Finally, the stop criteria for the optimization process must be configured. The stop criterion defines when the optimization process will terminate and present the results to the performance engineer. At least one stop criteria must be chosen. In case multiple criteria are chosen the optimization process stops when one of the criteria is fulfilled.

The following stop criteria are defined:

1. Duration of optimization process
2. Number of simulations
3. Number of iterations of the genetic algorithm
4. Specific number of candidates fulfilling minimum criteria is found
5. Specifying the minimum coverage metric C of the Pareto front of the new optimization run compared to Pareto front of the previous run

The coverage metric C used in Koziolek et al. (2011) is based on Zitzler et al. (2003) has the goal to define the efficiency of the found Pareto front from a new optimization run. This metric compares two Pareto fronts A and B by first calculating the common Pareto front P as $A \cup B$. Afterwards, the coverage $C(A, B)$ of A is calculated as the proportion of candidates from A in the Pareto front $P : \frac{|A \cap P|}{P}$. When the share of $C(A, B)$ is higher than 50% then A is the better front as more candidates are in P.

9.4.2 Evolutionary Optimization Process

Having defined a configuration the evolutionary optimization can be executed. We propose an automated optimization process that takes the configuration and an initial performance model as input and searches for workload candidates optimizing the given performance objectives. The optimization process search for Pareto-optimal candidates. Based on these candidates, performance engineers can select candidates for these objectives.

The selected performance objectives must first be translated into a fitness function used during the optimization process (Section 9.4.2.1). Afterwards, the optimization is executed until the stop criteria is reached (Section 9.4.2.2). Finally, the results are presented to the performance engineer (Section 9.4.2.3).

9.4.2.1 Fitness Functions

Each performance objective must be translated into a fitness function that is used during the optimization process. In case no thresholds are configured for a specific objective the optimization process tries to minimize / maximize the value of the functions.

In case a performance objective has to fulfill a predefined threshold or must be within a predefined value range the fitness function has to be adapted for this objective. In this case not the predicted value of a simulation (e.g. CPU utilization) should be maximized or minimized. The fitness function is adapted in a way that the function has the maximum when the metric is in the predefined range. As an example the CPU utilization of the resulting workload candidates should be within the range of 40% to 50% utilization. Then the fitness function must be adapted that the function has the maximum value within this range. In case the CPU utilization of a candidate is lower/higher than these thresholds, the value of the fitness function decreases. The higher the deviation of the predicted value is compared to the targeted range, the higher the penalty. We use the following functions to calculate the fitness function.

First the value m is calculated as the mean value for the given threshold range.

$$m = (threshold_{max} + threshold_{min})/2 \tag{9.3}$$

For instance if the minimum threshold is 40% and the maximum threshold 50%, the value m is 45%. Afterwards, the relative deviation d of the predicted value x to the value m is calculated. In case the value of x is within the threshold range the deviation is zero.

$$d = \begin{cases} 0, & \text{if x in threshold range} \\ \frac{m-x}{m}, & \text{if x not in threshold range} \end{cases} \tag{9.4}$$

Finally, we use the normal distribution to calculate the value of the function based on the value of d. We use the normal distribution with a mean value of zero and a standard deviation of 0.4. The lower the value of the standard deviation, the higher is the value of the normal distribution within the threshold. Thus, the penalty of values outside the threshold is higher. Figure 9.4 illustrates the fitness function of the CPU utilization for the threshold range 40% to 50%.

9.4.2.2 Evolutionary Optimization

The search problem of the evolutionary optimization will be formulated as a MOP. A MOP is defined as a problem where the goal is to optimize different maximization or minimization fitness functions simultaneously (Coello/Lamont/Van Veldhuisen, 2007). In this work a combination of maximization and minimization functions for the fitness functions $Q = q_1, \ldots, q_k$ will be applied (see Section 9.4.1.2). A MOP can be defined as:

$$\underset{c \in DS}{min/max}[f_{q_1}(c), f_{q_2}(c), ..., f_{q_k}(c)] \tag{9.5}$$

k is the number of fitness functions selected in the configuration process. The vector c is a n-dimensional decision variable vector $c = (c_1, ..., c_n)$ from some universe Ω. Each decision variable of the vector can be mapped to a design decision of the performance model. In our case it represents a workload candidate consisting of a Behavior Mix and a Workload Intensity. Depending on the configuration the Workload Intensity is varied as well or is assumed constant. The approach presented in this work searches for multiple

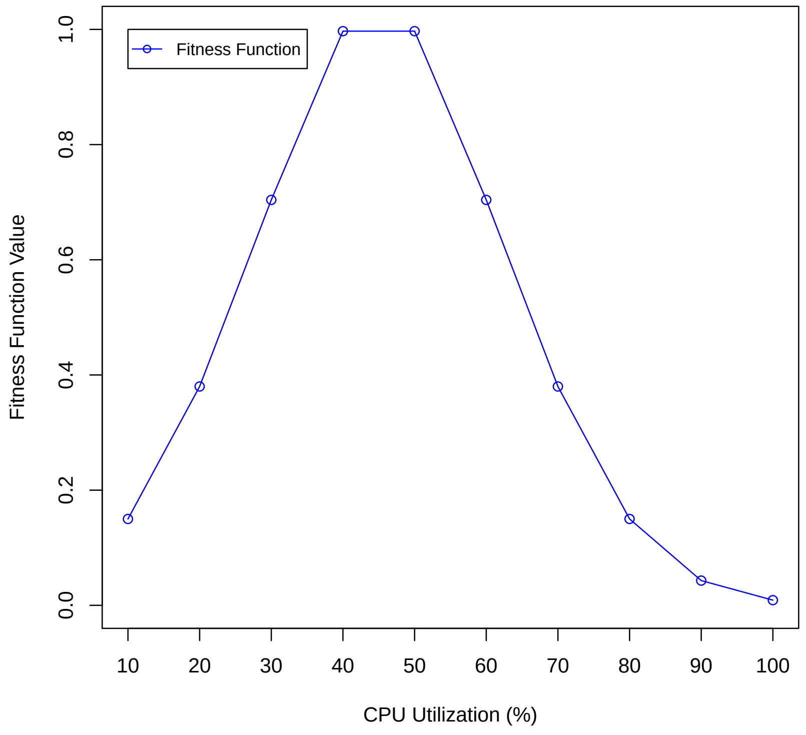


Figure 9.4: *Exemplary fitness function of the CPU utilization using the threshold range 40% to 50%.*

solutions instead of one global optimum. Several solutions can be found that fulfill these functions.

The Load Test Design Selector is based on the opt4j framework (version 3.1.4) for evolutionary computation (Lukasiewycz et al., 2011) using the NSGA-II genetic algorithm. To solve the MOP evolutionary algorithms are particularly suitable due to the fact that they are able to handle a set of possible Pareto-optimal solutions at the same time. A workload candidate is Pareto-optimal, if there exists no other candidate which would decrease some criterion without causing a simultaneous increase in at least one other criterion (assuming minimization) (Coello/Lamont/Van Veldhuisen, 2007). Further, another advantage of evolutionary algorithms is that they are capable of finding solutions in a very huge design space. Other meta-heuristics could be used as well.

The optimization process is illustrated in Figure 9.5. The optimization process takes the configuration and the initial candidate extracted in the previous step as input (see Section 9.3.3). A vector of possible workload design options is called genotype (Lukasiewycz et al., 2011) and represents the possible composition of the candidates. In the first iteration a set of candidates are randomly created by selecting concrete values for the decision variables of the genotypes. Each genotype is then decoded into concrete performance model instances. These performance model instances are called phenotypes and are concrete representatives of the genotype.

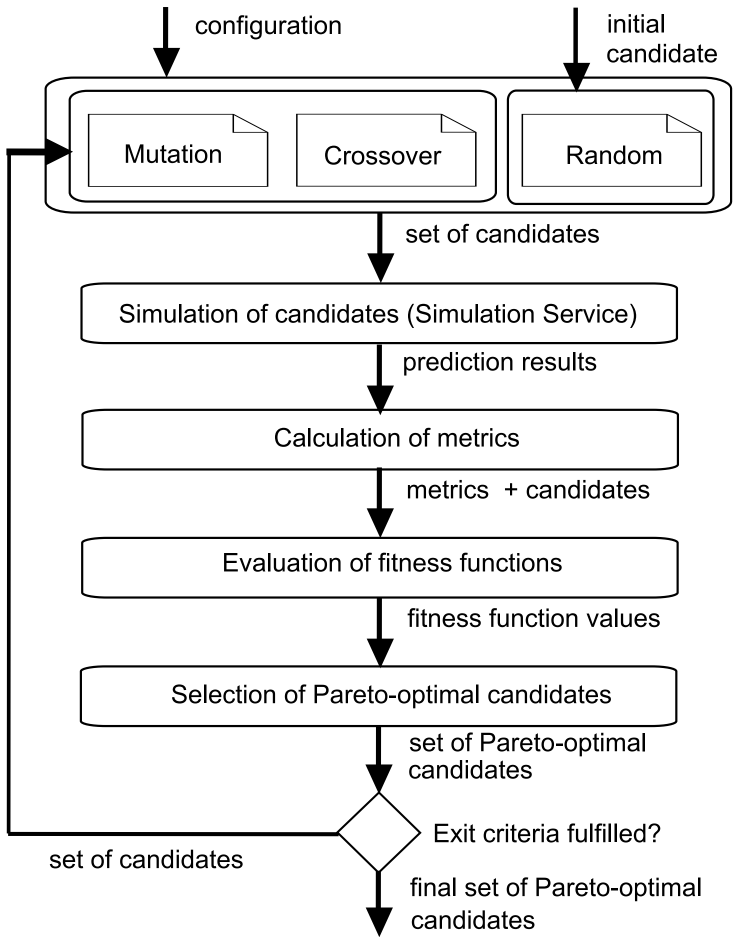


Figure 9.5: *Evolutionary Optimization Approach (based on Koziolek et al.* (2011)

Afterwards, each candidate is simulated in order to determine the prediction results. As the simulation of each candidate one after another would be very time consuming we execute the optimization process with the help of a distributed and scalable simulation cluster that can simulate multiple instances in parallel. This service, named SiaaS, simulates PCM instance based on a headless Eclipse instance (Willnecker/Vögele/Krcmar, 2016). Simulation jobs are triggered by a platform-independent REST interface and can be reused by other applications. This allows simulating a vast amount of model instances in parallel on cloud or on-premise installations.

In the next step based on the prediction results from the performance model the metrics that are used as input for the fitness functions are calculated (see Section 9.4.1.2). The configured fitness functions are evaluated and afterwards the candidates are checked for Pareto-optimality. Only the candidates which are Pareto-optimal are selected for the next generation.

When no exit criterion is reached these candidates are used for the next iteration. This time new candidates are created based on mutation and crossover (Coello/Lamont/Van Veldhuisen, 2007). Crossover mixes the genomes of the selected candidates and creates a new candidate. This is done for example by taking the Behavior Mix from one candidate and the Workload Intensity from another candidate. With mutation the values of the attributes of the candidates are modified, for example, by increasing the Workload

Intensity of one candidate. This way, the optimization iteratively searches the design space for better workload solutions based on the configured fitness functions.

When the configured stop criterion is reached the optimization will stop and the final set of Pareto-optimal candidates is presented to the user.

9.4.2.3 Results Presentation

Finally, the results of the optimization process will be presented to the test designer. The results are summarized as a list of the Pareto-optimal solutions and the predicted values of the configured performance objectives. Further, the resulting candidates will be presented using suitable visualizations like shown during the evaluation. The candidates which do not fulfill the threshold criteria can be filtered out. The test designer can identify the Pareto-optimal solutions and make trade-off decisions which test cases to choose based on given test goals. For instance, if resource coverage is more important than the maximization of the resource usage the selection of the Pareto-optimal solution where the coverage is higher can be preferred. Optionally, candidates of interest can also be examined in more detail and simulations with a higher simulation time can be conducted.

9.4.3 Automatic Generation of Load Test Scripts

The selected workload solution can automatically be feed back to the WESSBAS-DSL in terms of changing the Workload Intensity and the Behavior Mix specification. Based on the DSL the Test Plan Generator generates executable load test scripts for the common load testing tool Apache JMeter including the extension Markov4JMeter (van Hoorn et al., 2014) (see Section 9.3.4).

9.5 Evaluation

During evaluation, we apply our proposed extraction approach and tooling to the industry-standard benchmark SPECjEnterprise2010. This serves as an investigation of *(i.)* the practicality of the approach and tooling support (qualitative) and *(ii.)* the representativeness of the extracted load test designs (quantitative).

9.5.1 Research Questions and Evaluation Methodology

We particularly investigate the following research questions in order to evaluate our proposed approach:

- *RQ 1: How well is the applicability and the plausibility of the results of the multi-objective optimization approach?*

  To validate the applicability and the plausibility of the results of the approach two different configuration settings will be used. For each configuration a set of Pareto-optimal solutions will be derived from a performance model using the evolutionary optimization approach. These automatic derived solutions will then be visualized and discussed. Special care will be taken on the analysis of the plausibility of the results and on the time it takes to perform the evolutionary optimization.

- *RQ 2: How accurately do the performance and workload characteristics of the selected and simulated test design match the characteristics measured from the SUT?*

  To answer this research question for each run from (RQ 1) one Pareto-optimal solution will be selected and transformed into load test scripts (Section 9.3.4). The load test scripts will be executed and the measurement results of the real system will be compared with the predictions derived from the performance model.

An experiment using the SPECjEnterprise2010 Benchmark[30] will be executed. This Java EE benchmark represents an industry application of an automobile manufacturer whose main users are automobile dealers. First, a WESSBAS-DSL will be extracted (see Section 9.3.2) and a performance model of this benchmark will be created like proposed in Section 9.3.3.

As explained in Section 9.3.2 clustering algorithms are used for the identification of the Behavior Mix and the Behavior Models. With the configuration of the clustering the number of resulting cluster can be influenced. Depending on the number of derived clusters the complexity of the Behavior Models can be influenced. For example, when a high number of clusters is derived the number of possible test cases is increased but the complexity of the Behavior Models is reduced. Using X-means clustering the number of resulting clusters will be determined automatically by the algorithm. In this case the resulting Behavior Models can be quite complex. For example see purchase transaction of SPECjEnterprise2010 benchmark in Figure 9.7. As it could be the goal to reduce the complexity of the test cases simple Behavior Models are required. In this case K-means clustering with a high k value can be applied.

9.5.2 SPECjEnterprise2010 Deployment

The SPECjEnterprise2010 industry benchmark deployment is used for the evaluation of the proposed approach. SPECjEnterprise2010 is a Java EE application representing a business case combining customer relationship management (CRM), supply chain management (SCM), and manufacturing. It includes a workload specification and a dataset which is needed for the execution of load tests. The workload is generated by the Faban Harness and Benchmark Driver.[31] The benchmark consists of three different application

[30]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.

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

domains; namely, *Orders domain* (CRM), *Manufacturing domain*, and *Supplier domain* (SCM). The *Orders domain* (CRM) provides a Web-based user interface representing a standard e-commerce application with product information and a shopping cart. It drives the demand to the *Manufacturing domain*, which simulates production across different manufacturing plants. The task of the *Supplier domain* (SCM) is to order new parts for the Manufacturing domain. In this work, we consider only the Orders domain, which represents a typical Web-based application providing e-commerce functionality to customers; in this case automobile dealer. Using this application, customers are able to purchase and sell cars, to manage their accounts and dealership inventory, and to browse the catalogue of cars. The Orders domain runs independently from the other two domains, as they are mainly intended to be used as (Web-)service by other applications. It represents the production system / SUT.

9.5.2.1 Hardware Infrastructure

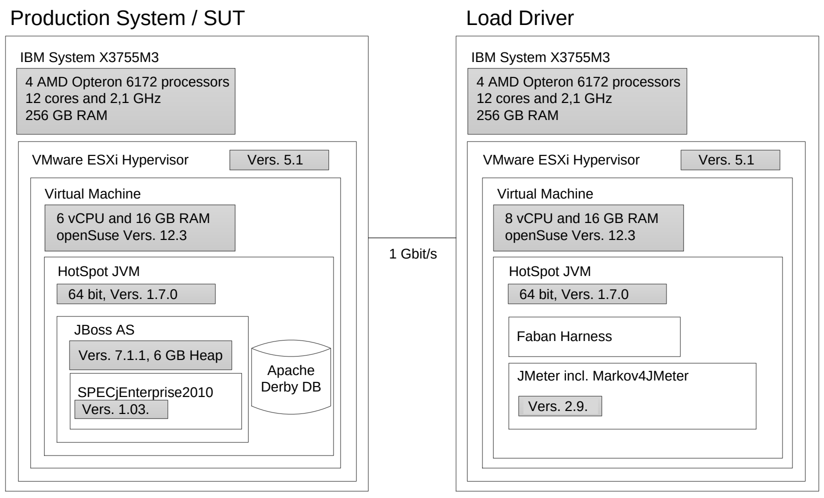


Figure 9.6: *Hardware and software infrastructure*

The SUT and the Dealer Driver are deployed on separate virtual machines (VM), linked by a 1 GBit/s network (see Figure 9.6). The SUT is deployed on an IBM System X3755M3 server with 6 virtual CPUs and 16 GB RAM. The Dealer Driver also runs on an IBM System X3755M3 server VM with 8 virtual CPUs and 16 GB RAM. The application server is JBoss 7.1.1. using the Java EE 6 full profile with 6 GB heap allocated. As persistence layer, an Apache Derby DB is used running in the same JVM as the JBoss application server. Both systems use openSUSE operating system in version 12.3 and are executed on a 64-bit OpenJDK 1.7.0 Server Java VM in version 1.7.0.

9.5.2.2 Workload Description

SPECjEnterprise2010 defines three different transaction types which are executed by automobile dealers: *Browse (B)*, *Manage (M)*, and *Purchase (P)*. Within the transaction type Browse, the benchmark driver navigates to the catalogue of available cars and browses the catalogue for a constant number of thirteen times. Manage describes a scenario during

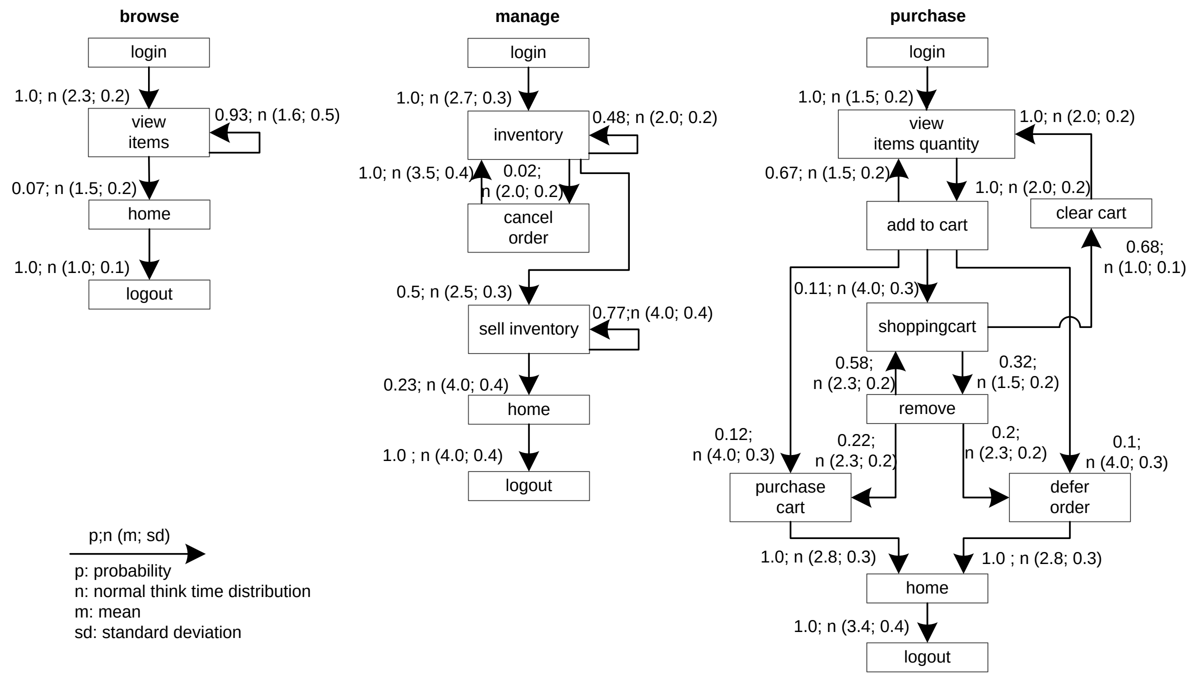


Figure 9.7: *SPECjEnterprise2010 transactions Browse, Manage, and Purchase as Behavior Models.*

which open orders are canceled and vehicles are sold. In the more complex transaction type Purchase, orders are placed and immediately purchased or deferred. The shopping cart is either cleared or items are removed one by one until only one item remains. Each of these transaction types corresponds to a sequence of HTTP requests. The workload in the Faban dealer driver is not defined in a probabilistic way and only a few of the HTTP requests are generated in a probabilistic way.

SPECjEnterprise2010 defines a total of 13 different HTTP request types, using a request parameter called *action*. We additionally split the request type called *View_Items* into two different request types as it executes two different use cases resulting in different resource demands; one request type is *View_Items* and the other is *View_Items_Quantity*. In the first use case, *View_Items* is called to browse the catalogue of available cars. In the second use case, only one specific item of the catalogue is selected.

Within the original dealer driver, no think times are defined between the execution of the HTTP actions, i.e., each HTTP action is executed directly after its previous request has been completed. Therefore, we added think times between these actions as Gaussian distribution with mean and standard deviation. The think times are randomly specified between mean values of one to four seconds. Figure 9.7 depicts the structure of the three transaction types as Behavior Models, obtained by applying our WESSBAS extraction approach including the transition probabilities and the specified think times.

In the original benchmark workload, automobile dealers log in to the system, execute *multiple instances* of the three transactions types, and log out. Each of the three transaction types is executed with a specified probability. The standard transaction mix is 50% Browse, 25% Manage, and 25% Purchase. We modified the dealer driver such that *each* transaction starts with a login and ends with a logout. In this way, each transaction corresponds to a unique session and the transaction mix corresponds to the Behavior Mix. As a result, the transaction types define the different navigational patterns.

9.5.2.3 Benchmark Execution and Monitoring

In a first step we execute a SPECjEnterprise2010 benchmark run in order to extract a WESSBAS-DSL instance and a performance model. We execute one benchmark run with the Faban Harness using the standard transaction mix and a load of 800 concurrent users, resulting in a moderate CPU utilization of the SUT of approximately 25%. The benchmark run is executed for twelve minutes after a three minute ramp-up phase and before a three minute ramp-down phase. We extract the system-specific parts of the performance model (as described in Section 9.3.3). This part of the performance model will be reused during the evaluation.

We collected standard HTTP access logs from the application server and transformed them to a session log using the Session Log Generator (see Figure 9.1). During the transformation, we only take complete sessions during steady state into account; meaning, sessions starting with a login request after the ramp-up phase and ending with a logout request before the ramp down phase. Thus, incomplete sessions are removed.

During the performance model creation we used k-means clustering with a high number of k (k = 50). This way we created a large number of test cases which are quite simple. Simple means that the test cases show little probabilistic behavior mostly representing a single path from the login request to the logout request. Using a high number of test cases we evaluate if our approach can also be used with a very large design space with many different test case combinations. However, due to the high number we are not able to visualize the resulting Behavior Models.

9.5.3 Results of Optimization Process

In order to answer RQ 1 (Section 9.5.1) we define two different exemplary configuration settings (see Table 9.4):

In the first configuration we are looking for workload candidates resulting in a throughput of 504 to 558 requests per second, having a low representativeness error and as fewer test cases as possible. The second configuration uses four performance objectives and derives workload candidates with a moderate CPU utilization between 20% and 25%, having a high test coverage, low representativeness error and also as less test cases as possible.

We configured opt4J to run until the coverage criteria C is below 53% (see Section 9.4.1.3). This means the optimization will stop when the Pareto front of the current iteration is at least 3% better than the Pareto front of the previous run. We used a parent population size of 50 and a offspring population size λ of 50 and a cross-over rate of 0.95%. For both configurations we enable to vary the workload design options Behavior Mix and Workload

Configuration 1

Performance Objective	Sign	Threshold	Value
Throughput	max	Y	504 - 558
Number of Test Cases	min	N	
Representativeness Error	min	N	

Configuration 2

Performance Objective	Sign	Threshold	Value
CPU Utilization - App Server	max	Y	20% - 25%
Coverage	max	N	
Number of Test Cases	min	N	
Representativeness Error	min	N	

Table 9.4: *Two configuration setting used for the evaluation of RQ 1*

Intensity. We used 16 parallel simulations for the simulation service and each simulation run simulates 300 time units.

As we can see in Figure 9.8 the coverage criteria C of configuration 1 was below 53% after 30 iterations. These iterations took approximately eight-and-a-half hours and resulted in 1500 simulations. Configuration 2 stopped after 20 iterations resulting in 1000 simulation runs and a duration of seven hours. We can also see that since iteration 16 the coverage criterion does not improve strongly anymore for both runs.

Configuration setting 1

Having executed the first configuration setting 86 candidates where considered to be Pareto-optimal. 14 of the 86 candidates are valid candidates, meaning candidates with a throughput exactly between 504 and 558 requests per second. As we use thresholds for the throughput criteria the objective function will be calculated like explained in Section 9.4.2.1. Thus, the fitness function (*throughput*‘) has the maximum value when the candidate is within the configured range. Figure 9.9 visualizes the *throughput*‘, representativeness error and number of test cases as 3-D Pareto front.

It can be seen that all Pareto-optimal candidates have a high value for the *throughput*‘ fitness function. This means the throughput of all candidates are in or very near to the configured throughput threshold. The candidate with the lowest *throughput*‘ value of 0.922 results in a throughput of 447 requests per seconds. This is only a difference of 11.3% to the required minimum throughput level of 504.

In Figure 9.10 the number of test cases against the representativeness error is shown. All candidates, the Pareto-optimal candidates and the valid Pareto-optimal candidates are illustrated. The Pareto-front on the lower left border shows that the lower the number of test cases the higher the representativeness error. This result is as expected as the

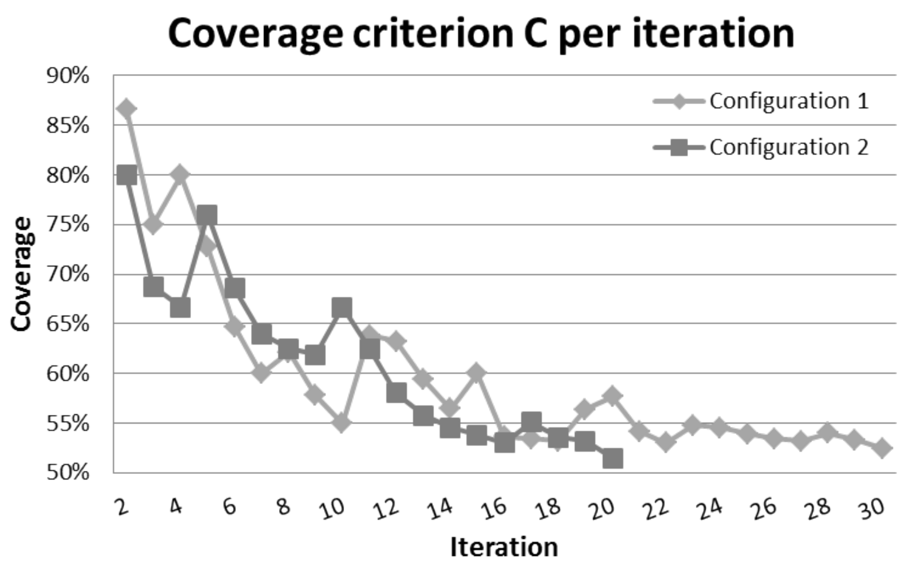


Figure 9.8: *Coverage criterion C of the Pareto front per iteration compared to the Pareto front of the previous iteration*

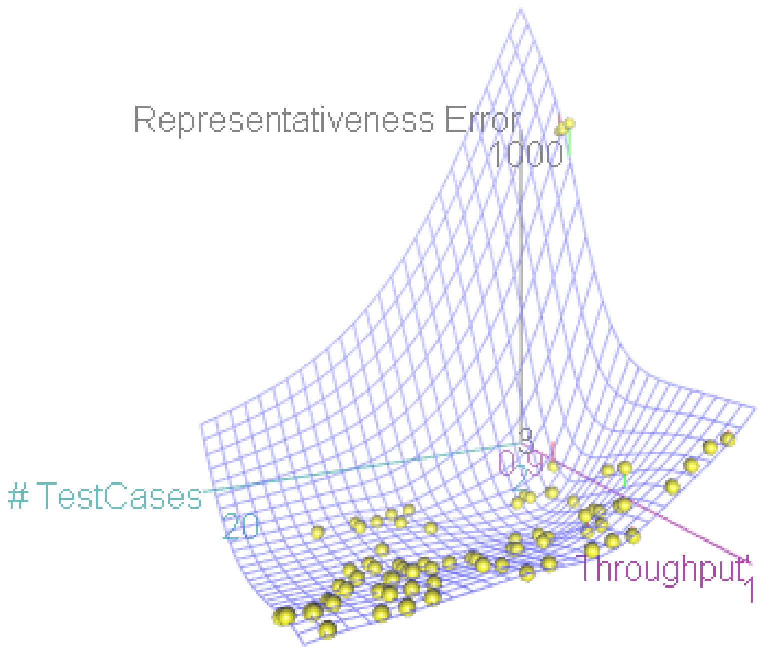


Figure 9.9: *3-D Pareto front throughput' vs. representativeness error vs. number of test cases*

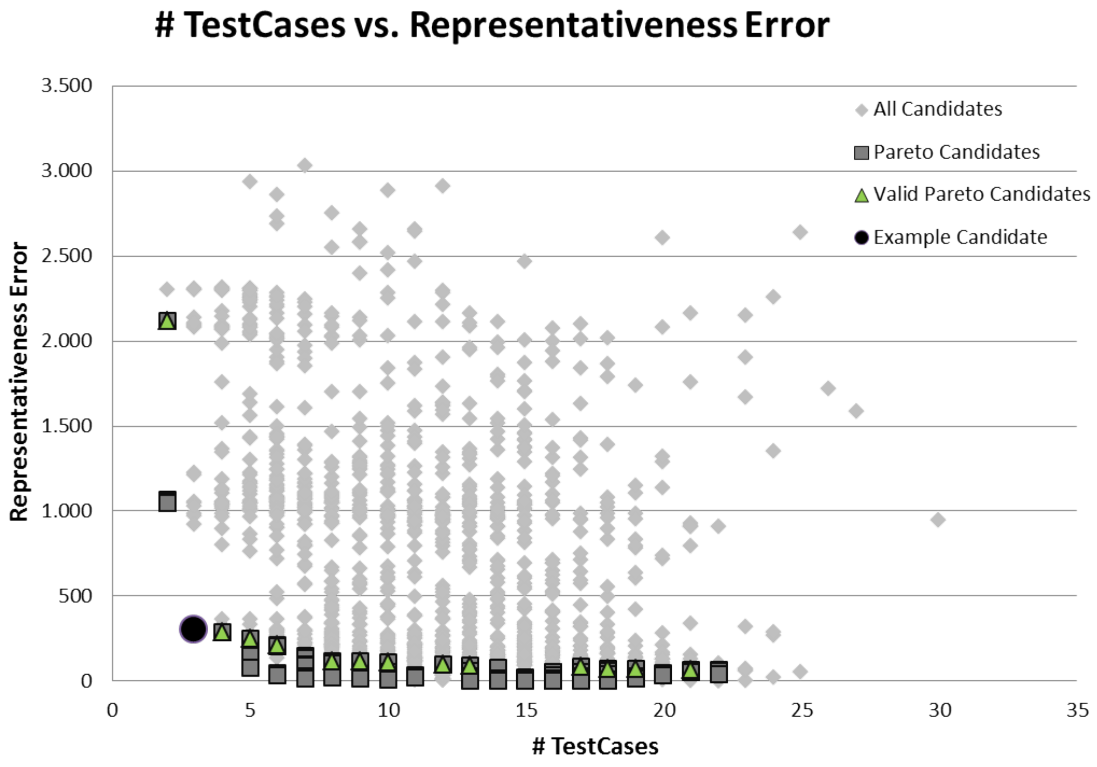


Figure 9.10: *Number of test cases vs. representativeness error*

lower the number of test cases it becomes more and more difficult to find representative Behavior Mixes.

An example of a valid candidate with a low number of test cases of 3 results in an representativeness error of 298 and a throughput of 504 requests per second. In case the representativeness error should be lower, than a higher number of test cases must be accepted by the performance engineer.

Configuration setting 2

87 of the candidates were considered as Pareto-optimal by the evolutionary algorithm. 17 of the 87 candidates are valid candidates in the CPU utilization range between 20% and 25%. Figure 9.11 shows the Pareto-front for coverage, representativeness error and number of test cases of all resulting candidates. We do not show the CPU utilization objective as this objective represents a constraint that should be fulfilled. This threshold criterion must be fulfilled, but it is not relevant how well it is fulfilled. Further, the CPU utilization has no impact on the other three objectives.

All candidate solutions as representativeness error over number of test cases are shows in Figure 9.12. The Pareto-optimal solutions and the valid Pareto-candidates are highlighted. Like in the first scenario it can be observed that the lower the number of selected test cases the higher the representativeness error.

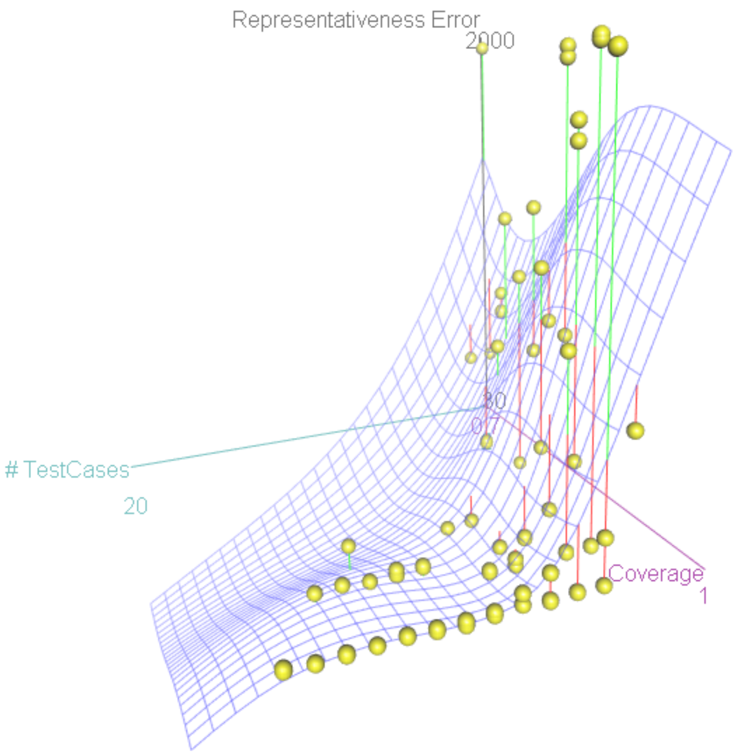


Figure 9.11: *3-D Pareto front coverage vs. representativeness error vs. number of test cases*

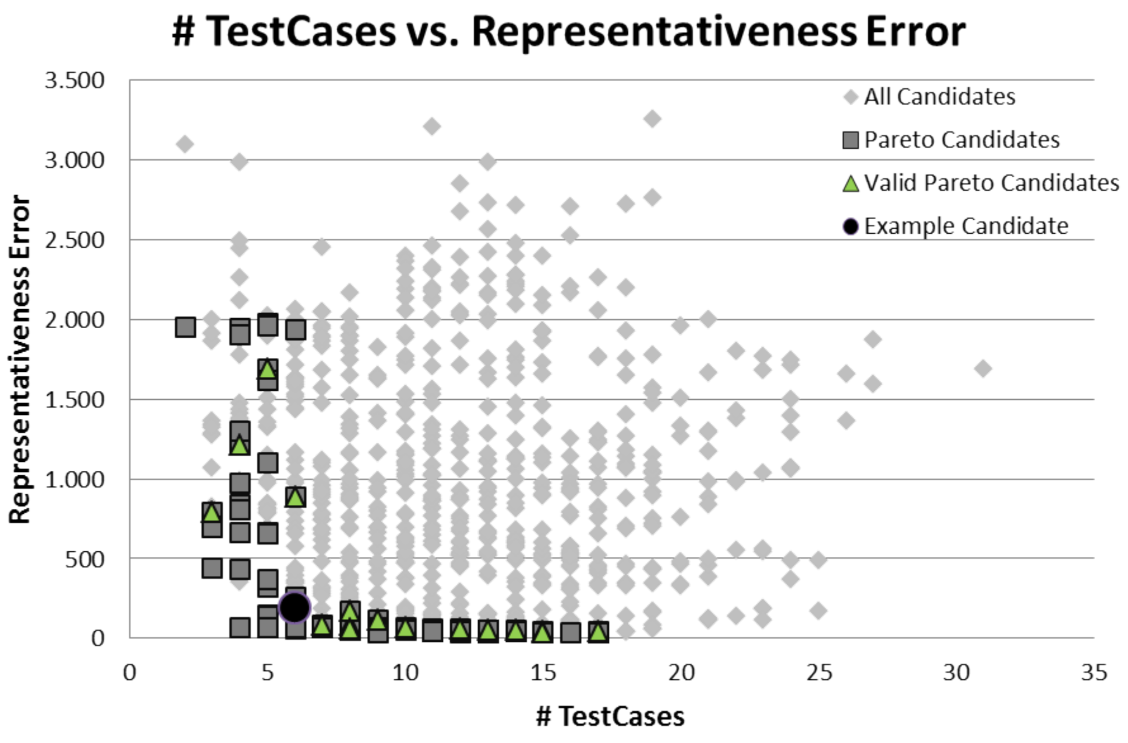


Figure 9.12: *Number of test cases vs representativeness error*

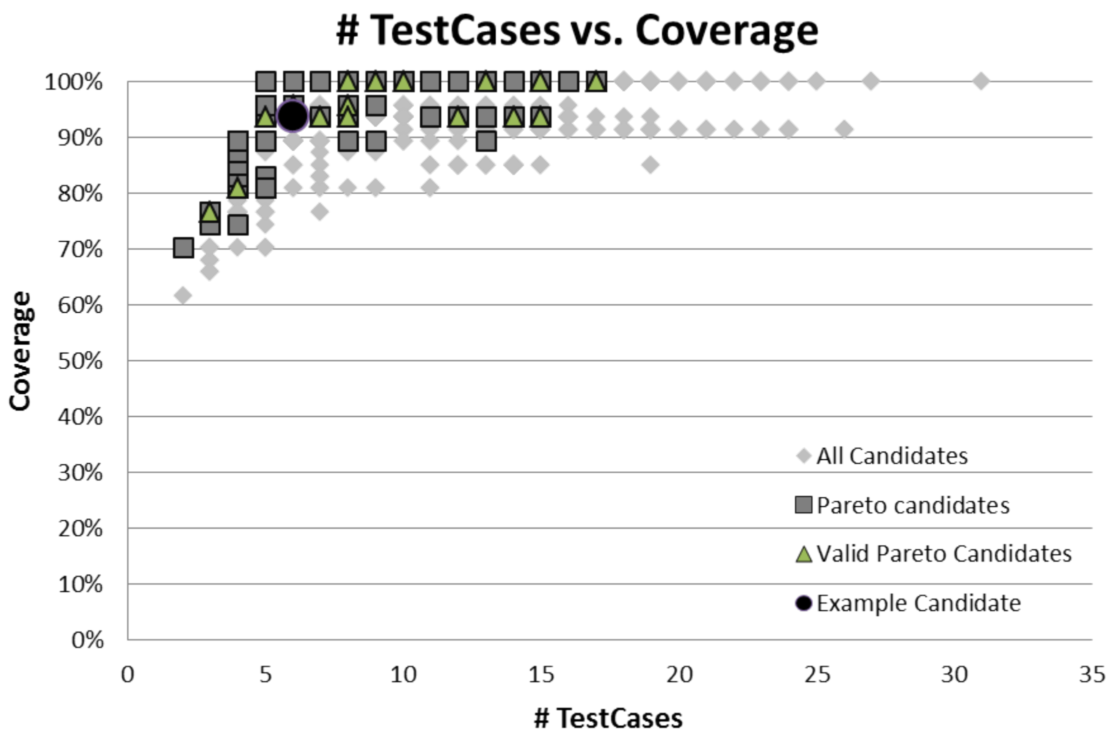


Figure 9.13: *Number of test cases vs. coverage*

In Figure 9.13 the number of test cases over the coverage is illustrated. It can be observed for the Pareto-optimal candidates that the lower the number of test cases the lower the coverage. Candidates who are not at the border are superior to others in the other two quality criteria representativeness error and CPU utilization.

The performance engineer can know decide which resulting candidate fits best to his objectives. For example, let's assume the performance engineer want to have a candidate with a high representativeness then the resulting candidate with 6 test cases is the best solution. In this case the representativeness error of this candidate is 182 SSE and the RDSEFF coverage is 94%. When the number of test cases should be lower than a candidate with a higher representativeness error and a lower coverage must be selected.

Concluding, the answer to RQ 1 is that the results of the multi-objective optimization process for the two configuration settings are plausible. The lower the number of test cases is the higher is the representativeness error and the lower the system coverage. Furthermore, the approach is able to find solution within the predefined thresholds. The applicability of the approach is also good as only little manual effort is need. However, the time it takes to conduct the analysis must be further optimized.

9.5.4 Prediction Accuracy of Selected Test Design Candidates

In this section for each configuration setting of the previous section a Pareto-optimal candidate is selected and the load test scripts for this candidate are generated as explained in

Section 9.4.3. The exemplary mentioned candidates of the previous section are taken into account. Afterwards, we execute the load test scripts and compare the predicted metrics of the Pareto-optimal candidates with the resulting measurements to answer RQ 2. Again, the benchmark run is executed for twelve minutes after a three minute ramp-up phase and before a three minute ramp-down phase. The results can be found in Table 9.5. We evaluate the prediction metrics required for the evaluation of the performance objectives (see Table 9.2). Further evaluations of resulting workload and performance characteristics of the extracted performance models and load test scripts can be found in our previous work (see Vögele et al. (2016); Brunnert/Vögele/Krcmar (2013)).

	Workload Candidate 1 - Configuration Setting 1			Workload Candidate 2 - Configuration Setting 2		
Performance Objective	Simulation	Measured	Prediction Error	Simulation	Measured	Prediction Error
CPU Utilization - App Server [%]	30.04%	31.1 %;34.03% (usr;1-idle)	3.41%; 11.71%	23.69%	25.61%; 28.36% (usr;1-idle)	7.51%; 16.45%
Response Time - Default Usage Scenario [s]	0.0036	0.0048	25.23%	0.0035	0.0037	5.41%
Throughput - Default Usage Scenario [req/s]	504	486	3.7%	464	459	1.09%
Representativeness Error [SSE]	298	309	3.56%	182	178	1.69%
Coverage [%]	82.97%	82.97%	0%	94%	94%	0%
Number of Test Cases [#]	3	3	0%	6	6	0%

Table 9.5: *Comparison of prediction metrics from resulting workload candidates with measurement results*

		Workload Candidate 1				Workload Candidate 2			
Request	MRP	MRP	SSE	SRP	SSE	MRP	SSE	SRP	SSE
home	8.0%	6.9%	1.2	6.9%	1.2	6.9%	1.2	6.8%	1.4
remove	0.4%	5.4%	25.1	5.4%	25.1	0.1%	0.1	0.1%	0.1
deferorder	0.9%	0.0%	0.8	0.0%	0.8	3.4%	6.1	3.5%	6.6
View_Items	28.3%	40.3%	145.5	40.0%	138.8	22.3%	36.1	23.0%	27.6
View_Items_quantity	8.4%	6.8%	2.5	6.8%	2.3	15.3%	47.8	15.4%	49.9
clearcart	0.8%	0.0%	0.6	0.0%	0.6	1.3%	0.3	1.3%	0.3
cancelorder	0.1%	0.0%	0.0	0.0%	0.0	0.0%	0.0	0.0%	0.0
shoppingcart	1.2%	5.4%	17.9	5.4%	17.8	1.4%	0.0	1.4%	0.0
Add_to_Cart	8.4%	6.8%	2.5	6.8%	2.4	15.3%	47.8	15.4%	49.8
logout	8.0%	6.9%	1.2	6.9%	1.2	6.9%	1.2	6.8%	1.4
inventory	8.3%	5.3%	9.2	5.3%	9.1	3.5%	23.0	3.4%	24.1
login	8.2%	6.9%	1.7	6.9%	1.7	8.1%	0.0	8.0%	0.0
sellinventory	18.0%	8.0%	100.9	8.2%	97.0	14.3%	13.8	13.5%	20.3
purchasecart	1.1%	1.3%	0.1	1.4%	0.1	1.3%	0.0	1.3%	0.0
Sum	100.0%	100.0%	309.2	100.0%	298.2	100.0%	177.6	100.0%	181.7

Table 9.6: *Representativeness error of workload candidates*

Workload Candidate 1

We measured the CPU utilization every 10 seconds using the Linux command line tool System Activity Reporter (SAR)[32]. The CPU utilization is split into overall CPU utilization (1-idle) and user CPU utilization.

This candidate shows a moderate prediction error for the overall CPU utilization of 11.71% and 3.41% for the user CPU utilization. The difference can be explained by the fact that the used performance model generator (Brunnert/Vögele/Krcmar, 2013) do not take the system utilization into account. Therefore the predicted overall utilization is lower

[32]http://linux.die.net/man/1/sar

than the total utilization. The prediction error of the average response times is with 25.23% below the 30% acceptable error propagated by (Menascé, 2002). The throughput is predicted with a very high accuracy (prediction error 3.7%) and the coverage is exactly the same compared to the measurements. The representativeness error of the simulated candidate and the measurement compared to the originally measured request distribution are both quite low. The simulated error is 298 SSE and the measured 309 SSE. This comes from the fact, that the simulated distribution (SRP) of the system level requests is almost identical to the measured distribution (MRP) (see Table 9.6).

Workload Candidate 2

The prediction error for the second workload candidate for the overall CPU utilization is 16.45%. The prediction error of the user CPU utilization is with 7.51% very low. The prediction error of the throughput defined as system level requests per second is with 1.09% also very low. The prediction error for the average response times is 5.41%. The workload metric representativeness error has almost the same value as the simulation (see Table 9.6).

Overall, the prediction accuracy of both workload candidates for the performance and workload characteristics is very high (RQ 2). Only the prediction errors of the response times are quite high but nevertheless below the 30% acceptable error.

9.6 Assumptions and Limitations

During our experiments a performance model generator is used (Brunnert/Vögele/Krcmar, 2013) to create the system-specific parts of the performance model in an automatic way. We were able to use this generator as it is designed for generating performance models for Java EE applications. Furthermore, the prediction accuracy of the generated model has previously been evaluated. This type of generator is not available for all session-based systems and performance models. Alternatively, the system-specific can also be modeled manually.

The usage model can also be modeled manually without the Wessbas approach in case no system implementation or no session records are available. In this case the usage model must be transformed in a way that it can be used with our approach. This means for example, the first branch represents the Behavior Mix followed by the Behavior Models. We will also provide tool support for this transformation. However, using manually modeled usage models no executable test case scripts can be generated.

The configuration of contra dictionary objectives and thresholds is currently possible. For instance, when a configuration is defined searching for candidates with a high CPU utilization threshold and very low response times. In this case no valid candidates can be found. This is difficult to prevented, as prediction values are not available at the time the configuration is specified. To prevent this, a workaround would be to allow only the configuration of thresholds for one objective.

The duration of the optimization process took around seven hours during our evaluation. Depending on the configured stop criteria and the duration of the simulation the duration can also be higher. As we use the scalable simulation service we can reduce the runtime by using additional simulation worker in parallel. As this increases the required hardware resources we will evaluate the usage of faster simulation engines as well. Also analytical solvers like the LQN (Koziolek/Reussner, 2008) solver would reduce the runtime.

The used metric coverage is very simple to use and interpret. Further, it can be directly calculated based on the prediction results. However, structural coverage metrics like statement, branch and condition coverage metrics, which are common in functional testing, could increase the accuracy of this approach. To enable this, also structural metrics, like which statements, branches and actions are called must be added to the PCM results.

9.7 Threats to Validity

No global optimization criterion exists to which this approach can be compared. Thus, it cannot be guaranteed that the real Pareto-optimal solutions are found. This comes also from the fact that we use a metaheuristic Coello/Lamont/Van Veldhuisen (2007). However, we presented a structured approach that optimizes effectively based on optimization goals.

Further, this approach is only applicable when the underlying performance model represents the most important user interactions. Therefore we extract the workload specification from the production system. In case the system is not yet in production the performance model must be created manually.

We enable to specify thresholds for performance objectives and proved that valid results can be derived. However, it cannot be guaranteed that all Pareto-optimal candidates fulfill the threshold criteria. This can be seen at the evaluation results. Thus, the number of resulting valid candidates can be considerably lower than the absolute number of Pareto-candidates. As we present all resulting candidates to the performance engineer, in some cases manual adjustment of the workload candidates is required. For example, when the CPU utilization for one candidate is too low the performance engineer can increase the Workload Intensity as long as the threshold is reached. However, this could be very time consuming.

Ideally, the proposed research questions should be answered using logs of a real-world system to obtain production workloads with corresponding performance measurements and a test environment for load testing. Some non-synthetic logs of real-world system are publicly available and have been used by researchers. However, we do not have performance measurements of these systems as well. Thus, we cannot use these publicly available logs only to evaluate our approach. Using synthetic logs imposes a threat to external validity and performance measurements would also be not available. As a result, a lab experiments under controlled conditions is the best option for us. Therefore, we select an industry standard benchmark that includes a representative workload profile. As future work we are planning to evaluate also against other applications like the SPECjEnterpriseNEXT benchmark which represents a distributed system setup.

9.8 Conclusions and Future Work

The selection of test case designs enabling the given performance objectives is an important task when planning load tests. However, it is still a big challenge to design the load tests before load test execution. In response to this challenge we propose an approach that extracts, evaluates and generates load test designs in an automatic way based on given multiple performance objectives. We use a performance model to assess different workload designs that match the objectives using a multi-objective genetic algorithm. The resulting Pareto-optimal candidates are presented to the performance engineer who can chose the candidate fitting to the trade-off decision. The evaluation with the industry-standard benchmark SPECjEnterprise2010 demonstrated the practicality and plausibility of the results of the proposed approach. Further, we demonstrate the high prediction accuracy of the simulated workload candidates.

As future work we will apply the approach to select load test cases in distributed applications with many servers and components. Furthermore, we extend the approach that it can be used with faster simulation engines like EventSim (Merkle/Henss, 2011) or analytical solvers like LQN to reduce the simulation time. Both the impact of different user input parameter will be investigated in the future and the integration with feedback-based learning approaches. Moreover, we investigate to add the resource types HDD and memory to the optimization process.

Part C

Chapter 10

Discussion

In this chapter the results of the publications are further discussed. First, the results are summarized and then limitations, contributions and future research are described.

10.1 Summary of Findings

An overview of the findings for each embedded publication is given in this section. The key results of each paper are also listed in Table 10.1.

In **P1** the goal was to support the load testing process by automatic extraction of representative and probabilistic workload specifications from system logs. To enable that these workload specifications are tool and system independent, a DSL is introduced. Instance of this DSL can be automatically extracted from system logs. During the extraction process groups of users having similar usage patterns are identified with clustering techniques. Based on these instances workload specifications of common load testing tools, in our case Apache JMeter (including the Markov4JMeter Plugin), are automatically generated. We evaluated the approach using the industry benchmark SPECjEnterprise2010. We proved that the workload characteristics of the extracted workload match the characteristics of the measured workload with high accuracy. Furthermore, the clustering technique was able to identify the transaction types of the SPECjEnterprise2010 benchmark with 100% accuracy. However, the load test scripts are not yet executable on real applications as the required protocol information are not extracted automatically. Therefore, we evaluated the extracted workload characteristics using a mock-up Web application.

The authors of **P2** worked in several industrial projects during this research. In this paper the authors were involved in a large-scale development project in which a SOA was developed. The main task within this project was to ensure that predefined performance requirements are met when the application is going into production. As the existing applications that should be integrated within this project are not designed for this integration the performance evaluation using load testing is a difficult task. Especially, when the services are under control of different teams and the services need to be scaled before the tests start. The contribution of this paper is the usage of performance models to predict service call frequencies for each service. The performance models are derived from existing

UML activity diagrams. Additionally, these models are used to derive suitable load test scenarios based on the most frequently used usage paths.

The goal of this dissertation is to derive load test designs with performance models. As a result, in **P3** we evaluated if the already extracted workload specification of **P1** can be transformed into workload specifications of model-based performance evaluation tools as well. Therefore, we transformed instances of the Wessbas-DSL into workload specification of performance models and evaluated this transformation using the SPEC-jEnterprise2010 industry benchmark. Again, the workload characteristics of the extracted workload match the characteristics of the measured workload with high accuracy. Thus, we could prove in this paper that the Wessbas-DSL can be used as an intermediate language for the extraction of workload specifications of load testing approaches and model-based performance evaluation. However, we encountered limitations in modeling complex workload specification only with the PCM usage model. Thus, as a workaround we violated the separation of concerns within PCM and modeled parts of the workload in the repository model using RDSEFFs.

P4 includes contents from and extends the content of **P1** and **P3**. This paper contains the following major improvements and extensions to the work reported in the previous papers.

- First, it considerable extends the Wessbas approach and tooling support (modeling language, extraction, generation). The relevant extensions include:
  - Automatic learning of guards and actions to ensure that valid sequences of user requests can be generated. Moreover, the guards and actions are automatically transformed to JMeter Test Plans and to the Palladio Component Model.
  - The extraction and integration of protocol information required to generate executable load tests.
  - The extraction and integration of input data that is required to execute the load tests.
- Second, a comprehensive evaluation against the industry-standard benchmark SPEC-jEnterprise2010 was conducted.
  - We have refined the research questions introduced in **P1** (impact of clustering algorithms and impact on accuracy of workload characteristics) and added three additional research questions (impact on the accuracy of performance characteristics, accuracy under changing workload settings, impact of guards and actions).
  - In van Hoorn et al. (2014) the workload characteristics of the extracted workloads are compared against a mock-up Web application. This was sufficient to analyze the workload characteristics. However, performance characteristics could not be evaluated. In this paper, we evaluate more workload characteristics (e.g., session durations, think times) and performance characteristics (e.g., response times, CPU utilizations, and heap usage) against the real SPEC-jEnterprise2010 application. Based on the extensions developed as part of

this paper, we were able to generate and execute the load test scripts for the SPECjEnterprise2010 application.

- Third, a more comprehensive description of the concepts from the previous work along with an extended related work section.

P5 has the goal to overcome the limitations of the modeling capability of the PCM usage model. The limitations include reusing modeled usage scenarios and the modeling of arbitrary usage flows. Due to these limitations the workload specification of **P2** cannot be modeled only in the PCM usage model. Thus, we extend the PCM usage model in way that complex usage pattern can be modeled only with the PCM usage model. This has the advantage that the clear separation of concerns within PCM can be prevented. As a result, the models that represent the system-specific parts are not modified when instances of the Wessbas-DSL are transformed into workload specifications of PCM.

As the extraction of representative workload specifications for load testing and model-based performance evaluation is enabled with the output of the papers **P1** to **P5** in **P6** we demonstrate how to select load test designs from performance models that fulfill given performance objectives. This approach, named Load Test Design Selector, is integrated with the Wessbas approach. As it is not always the goal to derive representative workloads, in this paper the performance engineer is enabled to derive specifications following (possibly multiple) different goals. Therefore, we define a set of performance objectives derived from literature which are common for load test design selection. These objectives comprise the commons goals for resource utilizations, response times, and throughput. We also include new objectives that are not considered in other approaches comprising the number of test cases, system coverage, and representativeness. For each goal it can be specified if it should be maximized/minimized or if a predefined threshold should be reached. As an example, the goal of a workload might be to stress the CPU to a specified value, using a minimal set of test cases and representing the real workload as good as possible. The performance objectives and combination of these objectives are specified by a performance engineer and can be manifold and depend on the system, the development status and the goals of the software project. Based on these performance objectives the load test designs are extracted.

We defined an multi-objective optimization process based on an EA that derives load test designs fulfilling the given objectives. The load test design in form of the Workload Intensity and the Workload Mix of the usage model are constantly adapted in the reproduction phase of the EA to generate new individuals that are evaluated based on the prediction results. The simulation framework introduced in **P13** is used to derive the prediction results (Willnecker/Vögele/Krcmar, 2016). This process iterates several times until a predefined exit criteria is reached. This process searches for Pareto-optimal candidates. A candidate is Pareto-optimal, if there exists no other candidate which would decrease some criterion without causing a simultaneous increase in at least one other criterion (assuming minimization). These resulting candidates are then presented to the performance engineer who can chose between the resulting load test designs. After the selection of a load test design it can be transformed into executable load test scripts.

No.	Key Results
P1	• Tool and system independent DSL for the specification of workloads • Extraction of instances of these DSL in an automatic way • Identification of groups of users having similar usage pattern • Transformation of these DSL instances into load test scripts • Evaluation of workload characteristics of extracted workloads
P2	Using performance models to support load testing of a large SOA environment by • prediction of service call frequencies • selection of usage scenarios from performance models
P3	• Transformation of DSL instances introduced in P1 into workload specifications of architecture-level performance models • Demonstration that the DSL can be used as intermediate language for the extraction of workload specifications of load testing approaches and model-based performance evaluation.
P4	• Enable to (close to) fully automatically extract and generate executable load tests and model-based performance predictions • A comprehensive evaluation against the industry-standard benchmark SPECjEnterprise2010 in which workload and performance characteristics are evaluated • A more comprehensive description of the concepts from the previous work along with an extended related work section
P5	• Extension of PCM usage model meta-model to enable the modeling of complex usage pattern • By transforming WESSBAS-DSL instances into workload specification of PCM the clear separation of concerns within PCM can be retained
P6	• Approach for multi-objective optimization of load test designs based on performance models • Integration with the WESSBAS approach to automatically derive performance models and executable load tests from the chosen test design • Comprehensive evaluation against the industry-standard benchmark SPECjEnterprise2010

Table 10.1: *Key results of embedded publications*

Again, using the industry benchmark SPECjEnterprise2010 we proved the practicality and plausibility of the resulting load test design candidates. We defined two real world scenarios and derived load test design candidates for each scenario. Additionally, we demonstrate the high prediction accuracy of the simulated workload candidates by comparison with real measurements.

10.2 Limitations

There are several limitations that must be considered. First, the used data structure in the Wessbas approach to represent Behavior Models can be very inefficient when these models comprise a large number of states. The size of the transition matrices grow exponentially as the number of possible transition is n^2. Especially, large matrices increase the complexity for the clustering with the Weka framework. This fact could possibly make it unfeasible to use the Wessbas approach with a large number of states.

The usage of performance models to select workload specifications is dependent on a performance model that models the system-specific parts of a system with high prediction accuracy. These models cannot be extracted from all types of application systems in a automatic way. During this thesis we used an already evaluated extraction approach for Java EE applications. For other types of systems it is possible that no automatic extraction approach (see Section 2.2.4) is available and manual modeling is required.

The main limitations of the multi-objective optimization is that no global optimization criterion exists to which this approach can be compared. As a result, it cannot be guaranteed that the best Pareto-optimal solutions are found. Moreover, it takes same time to conduct the optimization and it is quite resource intensive. The runtime of the evolutionary optimization can be reduced when faster simulation techniques (e.g. EventSim Merkle/Henss (2011)) or analytical solvers (e.g. LQN Solver Koziolek/Reussner (2008)) are applied.

During this thesis we primarily used the industry benchmark SPECjEnterprise2010 to evaluate the developed prototypes. As it is difficult for researchers to get access to real world application systems we used this industry benchmark. As we mainly used this benchmark, the generalizability of our findings is limited. In the future it would be of great benefit if the proposed approach is evaluated using real world systems.

10.3 Contribution to Research

In literature most approaches use performance models in combination with multi-objective optimization techniques to derive optimal system architectures. In this thesis we are the first approach that uses these models to derive executable load test designs following multiple performance objectives. Research in this area is mainly focusing on extracting either representative workloads or on extracting workloads that are able to detect performance problems under load. This dissertation addresses this gap by proposing an automatic approach that derives load test design from performance models following multiple performance goals using a multi-objective optimization approach. Additionally, we formulate further performance goals that could additionally be taken into account like component coverage and number of test cases.

With the modeling of the workload specification using a DSL we are not only enabling to extract system and tool agnostic models we are also able to integrate the workload modeling for measurement and model-based approaches. This DSL is extracted from runtime data and can be used as intermediate language to generate executable load tests and workload specifications for model-based performance predictions. To the best of our knowledge Wessbas this is the first approach that enables the process from runtime data to executable load tests and model-based performance prediction.

In the field of software performance research two main approaches exists that model user behavior. One way is the usage of CBMGs (van Hoorn/Rohr/Hasselbring, 2008; Menascé, 2002) extracted from HTTP server logs, which are based on Markov Chains. The other is the extraction and specification of EFSMs. In our work we combine both approaches. As a result, probabilistic user behavior modeling is enabled (cf. CBMGs) while ensuring that valid sequences of user requests are generated (EFSMs) with the usage of guards and actions. As the probabilities of the extracted workload specifications are changed when guards and actions are introduced, conditional probabilities must be calculated for each transition. We proved that we can combine both approaches and extract workload specifications which are almost identical to the measured workload specifications.

10.4 Contribution to Practice

The combination of measurement and model-based prediction is of great benefit for the usage in practice. The flexible specification of performance goals and the selection of load test designs enable faster selection of load tests, as the number of possible load test design options is too high to be evaluated manually. Furthermore, the assessment of the impact of workload designs on the system is only possible to evaluate by conducting multiple load tests. The approach is also applicable for all session-based systems and requires no detailed knowledge about workload extraction.

This dissertation considerably improves the applicability of model-based performance evaluations in practice by providing tool support for the automatic extraction of large parts of these models. This way the required effort to specify workloads that fulfill the given performance objectives is significantly reduced. The ability to automatically generate load tests and workload specifications of performance models makes it possible to better apply existing model-based performance evaluation techniques in practice. Workload specifications and usage pattern can be continuously extracted from the production environment and used for load tests during development. This is especially useful when release cycles of the systems should be reduced and not much time to specify the load tests is available. As a result, the Wessbas approach can be integrated into DevOps processes.

10.5 Future Research

There are several possible directions for future research. First of all, the integration of the approach in DevOps processes can further be evaluated. During continuous integration the capabilities of model-based performance prediction can be used to identify weaknesses in the system architecture for specific usage pattern. This way the robustness of the system architecture can be continuously assessed and further improved.

Another important direction of future research is to consider the impact of different input variables on the selection of the test cases. Input variables can have a high impact on the response times of specific user requests. For example, the response time can be considerably lower when a customer with a very seldom name is searched in a application system than a customer with a very popular name. Even though these parametric dependencies can be specified in PCM but yet cannot be extracted in an automatic way.

Furthermore, the evaluation of the approach for modern distributed applications is required. The industry benchmark SPECjEnterprise2010 is more a classical three tier web application which will be gradually substituted by multi-tier applications. Furthermore, modern web technologies must be evaluated which do not provide the required information in the request URL to derive protocol information for the execution of load tests. In this case it is not possible to generate executable load tests based on standard http access logs.

The integration with feedback-based learning approaches is also of interest. During load test execution the measured and predicted system behavior can be compared. When there are differences the models can be adjusted and new test cases can be selected automatically (Barna/Litoiu/Ghanbari, 2011a).

The identification of test coverage criteria is also a very interesting direction of future research. Only few coverage criteria for load test selection are currently available. In this dissertation we introduced and used a criteria for the determination of representative workloads and a quite simple coverage criterion based on the percentage of methods called within a test. The introduction of new test coverage criteria used for load test design selection remains important. Moreover, the relation of classical coverage criterion used during functional testing (e.g. code coverage, structural metrics) and performance characteristic should also be examined in more detail.

References

Abbors, F.; **Ahmad, T.**; **Truşcan, D.**; **Porres, I. (2012):** MBPeT: A Model-Based Performance Testing Tool. In Proceedings of the 4th International Conference on Advances in System Testing and Validation Lifecycle (VALID)., 1–8.

Alavi, M. (1984): An Assessment of the Prototyping Approach to Information Systems Development. *Communications of the ACM*, vol. 27 no. 6, 556–563, ISSN 0001–0782.

Aleti, A.; **Buhnova, B.**; **Grunske, L.**; **Koziolek, A.**; **Meedeniya, I. (2013):** Software Architecture Optimization Methods: A Systematic Literature Review. *Software Engineering, IEEE Transactions on*, vol. 39 no. 5, 658–683, ISSN 0098–5589.

Ardagna, D.; **Casale, G.**; **Ciavotta, M.**; **Pérez, J. F.**; **Wang, W. (2014):** Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, vol. 5 no. 1, 11, ISSN 1869–0238.

Arlitt, M.; **Jin, T. (2000):** A workload characterization study of the 1998 World Cup Web site. *Network, IEEE*, vol. 14 no. 3, 30–37, ISSN 0890–8044.

Arlitt, M. F.; **Krishnamurthy, D.**; **Rolia, J. (2001):** Characterizing the Scalability of a Large Web-based Shopping System. *ACM Transactions on Internet Technology*, vol. 1 no. 1, 44–69, ISSN 1533–5399.

Avritzer, A.; **Weyuker, E. R. (1995):** The automatic generation of load test suites and the assessment of the resulting software. *IEEE Transactions on Software Engineering*, vol. 21 no. 9, 705–716, ISSN 0098–5589.

Avritzer, A.; **Kondek, J.**; **Liu, D.**; **Weyuker, E. J. (2002):** Software performance testing based on workload characterization. In Proceedings of the 3rd International Workshop on Software and Performance (WOSP)., 17–24.

Balsamo, S.; **Marco, A. D.**; **Inverardi, P.**; **Simeoni, M. (2004):** Model-based performance prediction in software development: A survey. vol. 30 no. 5, 295–310, ISSN 0098–5589.

Barber, S. (2004): Creating Effective Load Models for Performance Testing with Incomplete Empirical Data. In Proceedings of the 6th International Workshop on Web Site Evolution (WSE). IEEE, 51–59.

Barford, P.; **Crovella, M. (1998):** Generating Representative Web Workloads for Network and Server Performance Evaluation. *SIGMETRICS Perform. Eval. Rev.* vol. 26 no. 1, 151–160, ISSN 0163–5999.

Barna, C.; **Litoiu, M.**; **Ghanbari, H. (2011a):** Autonomic load-testing framework. In Proceedings of the 8th International Conference on Autonomic Computing (ICAC). ACM, 91–100.

Barna, C.; **Litoiu, M.**; **Ghanbari, H. (2011b):** Model-based Performance Testing (NIER Track). In Proceedings of the 33rd International Conference on Software Engineering. ACM, New York, NY, USA, ICSE '11, ISBN 978–1–4503–0445–0, 872–875.

Bayan, M. S.; **Cangussu, J. W. (2006):** Automatic Stress and Load Testing for Embedded Systems. In 30th Annual International Computer Software and Applications Conference (COMPSAC'06). IEEE, ISSN 0730–3157, 229–233.

Bayan, M.; **Cangussu, J. a. W. (2008):** Automatic Feedback, Control-based, Stress and Load Testing. In Proceedings of the 2008 ACM Symposium on Applied Computing. ACM, New York, NY, USA, SAC '08, ISBN 978–1–59593–753–7, 661–666.

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.

Berkhin, P.; Kogan, J.; **Nicholas, C.**; **Teboulle, M. eds. (2006):** A Survey of Clustering Data Mining Techniques. Springer Berlin Heidelberg, Berlin, Heidelberg, 25–71, ISBN 978–3–540–28349–2.

Beschastnikh, I.; **Brun, Y.**; **Schneider, S.**; **Sloan, M.**; **Ernst, M. D. (2011):** Leveraging Existing Instrumentation to Automatically Infer Invariant-constrained Models. In Proceedings of the 19th SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. ACM, 267–277.

Brebner, P. C. (2016): Automatic Performance Modelling from Application Performance Management (APM) Data: An Experience Report. In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering. ACM, New York, NY, USA, ICPE '16, ISBN 978–1–4503–4080–9, 55–61.

Briand, L. C.; **Labiche, Y.**; **Shousha, M. (2005):** Stress Testing Real-time Systems with Genetic Algorithms. In Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation. ACM, New York, NY, USA, GECCO '05, ISBN 1–59593–010–8, 1021–1028.

Brosig, F.; **Huber, N.**; **Kounev, S. (2011):** Automated extraction of architecture-level performance models of distributed component-based systems. In Proceedings of the 26th International Conference on Automated Software Engineering (ASE). IEEE/ACM, ISSN 1938–4300, 183–192.

Brosig, F.; **Kounev, S.**; **Krogmann, K. (2009):** Automated Extraction of Palladio Component Models from Running Enterprise Java Applications. In Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, VALUETOOLS '09, ISBN 978–963–9799–70–7, 1–10.

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 Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days., 30–38.

Brunnert, A.; **Hoorn, A. van**; **Willnecker, F.**; **Danciu, A.**; **Hasselbring, W.**; **Heger, C.**; **Herbst, N.**; **Jamshidi, P.**; **Jung, R.**; **Kistowski, J. von**; **Koziolek, A.**; **Kroß, J.**; **Spinner, S.**; **Vögele, C.**; **Walter, J.**; **Wert, A. (2015):** Performance-oriented DevOps: A Research Agenda. SPEC Research Group — DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC) (SPEC-RG-2015-01) – technical report.

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 (VALUETOOLS)., 165–172.

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 Proceedings of the 10th European Workshop on Performance Engineering (EPEW). Springer, ISBN 978–3–642–40724–6, 74–88.

Bulej, L.; **Kalibera, T.**; **Tůma, P. (2005):** Repeated Results Analysis for Middleware Regression Benchmarking. *Performance Evaluation*, vol. 60 no. 1, 345–358.

Calinescu, R.; **Grunske, L.**; **Kwiatkowska, M.**; **Mirandola, R.**; **Tamburrelli, G. (2011):** Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering*, vol. 37 no. 3, 387–409, ISSN 0098–5589.

Calzarossa, M. C.; **Massari, L.**; **Tessera, D. (2016):** Workload Characterization: A Survey Revisited. *ACM Comput. Surv.* vol. 48 no. 3, 48:1–48:43, ISSN 0360–0300.

Canfora, G.; **Di Penta, M.**; **Esposito, R.**; **Villani, M. L. (2005):** An Approach for QoS-aware Service Composition Based on Genetic Algorithms. In Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation. ACM, New York, NY, USA, GECCO '05, ISBN 1–59593–010–8, 1069–1075.

Ciancone, A.; **Filieri, A.**; **Drago, M.**; **Mirandola, R.**; **Grassi, V. (2011):** Klaper-Suite: An Integrated Model-Driven Environment for Reliability and Performance Analysis of Component-Based Systems. In Objects, Models, Components, Patterns Springer, ISBN 978–3–642–21951–1, 99–114.

Coello, C. A. C.; **Lamont, G. B.**; **Van Veldhuisen, D. A. (2007):** Evolutionary algorithms for solving multi-objective problems. vol. 5, Springer, ISBN 0387367977.

Costa, L. T.; **Czekster, R. M.**; **Oliveira, F. M. de**; **Rodrigues, E. d. M.**; **Silveira, M. B. da**; **Zorzo, A. F. (2012):** Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE)., 112–117.

Danciu, A.; **Kross, J.**; **Brunnert, A.**; **Willnecker, F.**; **Vögele, C.**; **Kapadia, A.**; **Krcmar, H. (2015):** 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.

Denning, P. J. (1997): A New Social Contract for Research. *Communications of the ACM*, vol. 40 no. 2, 132–134, ISSN 0001–0782.

Di Penta, M.; **Canfora, G.**; **Esposito, G.**; **Mazza, V.**; **Bruno, M. (2007):** Search-based Testing of Service Level Agreements. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. ACM, New York, NY, USA, GECCO '07, ISBN 978–1–59593–697–4, 1090–1097.

Draheim, D.; **Grundy, J.**; **Hosking, J.**; **Lutteroth, C.**; **Weber, G. (2006):** Realistic load testing of Web applications. In Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR). IEEE, ISBN 1534–5351, 59–70.

Dynatrace (2016): Dynatrace. `http://www.dynatrace.com`, Last accessed December 11, 2016.

Ehrgott, M. (2006): Multicriteria optimization. Springer Science & Business Media, ISBN 978–3540213987.

Feitelson, D. G. (2002): Workload modeling for performance evaluation. *Performance Evaluation of Complex Systems: Techniques and Tools*, vol. 2459, 114–141.

Feitelson, D. G. (2015): Workload modeling for computer systems performance evaluation. Cambridge University Press, ISBN 978–1107078239.

Filieri, A.; **Grunske, L.**; **Leva, A. (2015):** Lightweight Adaptive Filtering for Efficient Learning and Updating of Probabilistic Models. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1, IEEE, 200–211.

Foo, K. C.; **Jiang, Z. M.**; **Adams, B.**; **Hassan, A. E.**; **Zou, Y.**; **Flora, P. (2010):** Mining Performance Regression Testing Repositories for Automated Performance Analysis. In 2010 10th International Conference on Quality Software. IEEE, ISSN 1550–6002, 32–41.

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.

Gambi, A.; **Filieri, A.**; **Dustdar, S. (2013):** Iterative Test Suites Refinement for Elastic Computing Systems. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, New York, NY, USA, ESEC/FSE 2013, ISBN 978–1–4503–2237–9, 635–638.

Goševa-Popstojanova, K.; **Singh, A. D.**; **Mazimdar, S.**; **Li, F. (2006):** Empirical Characterization of Session-Based Workload and Reliability for Web Servers. *Springer Empirical Software Engineering*, vol. 11 no. 1, 71–117, ISSN 1382–3256.

Grechanik, M.; **Chen, F.**; **Xie, Q. (2012):** Automatically finding performance problems with feedback-directed learning software testing. In Proceedings of the 34th International Conference on Software Engineering (ICSE 2012). , ISBN 0270–5257, 156–166.

Grinshpan, L. (2012): Solving enterprise applications performance puzzles: queuing models to the rescue. Wiley-IEEE Press, ISBN 978–1–1180–615–72.

Hall, M.; **Frank, E.**; **Holmes, G.**; **Pfahringer, B.**; **Reutemann, P.**; **Witten, I. H. (2009):** The WEKA data mining software: An update. *ACM SIGKDD Explor. Newsletter*, vol. 11 no. 1, 10–18.

Heinrich, R.; **Merkle, P.**; **Henss, J.**; **Paech, B. (2015):** Integrating business process simulation and information system simulation for performance prediction. *Software & Systems Modeling*,, 1–21, ISSN 1619–1374.

Heinrich, R.; **Schmieders, E.**; **Jung, R.**; **Rostami, K.**; **Metzger, A.**; **Hasselbring, W.**; **Reussner, R.**; **Pohl, K. (2014):** Integrating Run-Time Observations and Design Component Models for Cloud System Analysis. In 9th International Workshop on Models@run.time. CEUR Vol-1270, 41–46.

Herbst, N. R.; **Huber, N.**; **Kounev, S.**; **Amrehn, E. (2013):** Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. In Proceedings of the International Conference on Performance Engineering (ICPE). ACM, 187–198.

Hevner, A. R.; **March, S. T.**; **Park, J.**; **Ram, S. (2004):** Design Science in Information Systems Research. *MIS Quarterly* vol. 28 no. 1.

Hoorn, A. van; **Rohr, M.**; **Hasselbring, W. (2008):** Generating Probabilistic and Intensity-Varying Workload for Web-Based Software Systems. In Proceedings of the SPEC International Performance Evaluation Workshop 2008. vol. 5119, Springer, ISBN 978–3–540–69813–5, 124–143.

Hoorn, A. van; **Vögele, C.**; **Schulz, E.**; **Hasselbring, W.**; **Krcmar, H. (2014):** Automatic Extraction of Probabilistic Workload Specifications for Load Testing Session-Based Application Systems. In Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS). ACM, ISBN 978–1–63190–057–0, 139–146.

Hoorn, A. van; **Waller, J.**; **Hasselbring, W. (2012):** Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In Proceedings of the International Conference on Performance Engineering (ICPE). ACM, ISBN 978–1–4503–1202–8, 247–248.

Israr, T. A.; **Lau, D. H.**; **Franks, G.**; **Woodside, M. (2005):** Automatic Generation of Layered Queuing Software Performance Models from Commonly Available Traces. In Proceedings of the 5th International Workshop on Software and Performance. ACM, New York, NY, USA, WOSP '05, ISBN 1–59593–087–6, 147–158.

Jain, R. (1991): The Art of Computer Systems Performance Analysis. John Wiley & Sons, New York, NY, USA, ISBN 978–0–471–50336–1.

Jiang, Z. M.; **Hassan, A. E. (2015):** A Survey on Load Testing of Large-Scale Software Systems. *IEEE Transactions on Software Engineering*, vol. 41 no. 11, 1091–1118, ISSN 0098–5589.

Junzan, Z.; **Bo, Z.**; **Shanping, L. (2014):** LTF: A Model-Based Load Testing Framework for Web Applications. In Proceedings of the 14th International Conference on Quality Software (QSIC). IEEE, ISBN 1550–6002, 154–163.

Kalaji, A. S.; **Hierons, R. M.**; **Swift, S. (2009):** Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM). In 2009 International Conference on Software Testing Verification and Validation. IEEE, ISSN 2159–4848, 230–239.

Kim, G.-H.; **Kim, Y.-G.**; **Shin, S.-K.; Kim, K. J.**; **Chung, K.-Y. eds. (2013):** Software Performance Test Automation by Using the Virtualization. Springer Netherlands, Dordrecht, 1191–1199, ISBN 978–94–007–5860–5.

Kistowski, J. v.; **Herbst, N. R.**; **Kounev, S. (2014):** Modeling Variations in Load Intensity over Time. In Proceedings of the 3th International Workshop on Large Scale Testing (LT). ACM, ISBN 978–1–4503–2762–6, 1–4.

Kolmogorov, A. N. (1950): Foundations of the Theory of Probability. Chelsea Publishing Co..

Kounev, S.; **Brosig, F.**; **Huber, N. (2014):** The Descartes Modeling Language. Department of Computer Science, University of Wuerzburg – technical report.

Koziolek, A.; **Koziolek, H.**; **Reussner, R. (2011):** PerOpteryx: Automated Application of Tactics in Multi-objective Software Architecture Optimization. In Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS. ACM, ISBN 978–1–4503–0724–6, 33–42.

Koziolek, H. (2010): Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, vol. 67 no. 8, 634–658.

Koziolek, H.; **Reussner, R.; Kounev, S.**; **Gorton, I.**; **Sachs, K. eds. (2008):** A Model Transformation from the Palladio Component Model to Layered Queueing Networks. Springer Berlin Heidelberg, Berlin, Heidelberg, 58–78, ISBN 978–3–540–69814–2.

Krcmar, H. (2015): Informationsmanagement. Springer, Berlin/Heidelberg, Germany, 6. Auflage, ISBN 978–3–662–45863–1.

Krishnamurthy, D.; **Rolia, J. A.**; **Majumdar, S. (2006):** A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems. *IEEE Transactions on Software Engineering*, vol. 32 no. 11, 868–882.

Labs, H.-P. (1998): Worldcup98 web logs. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`, Last accessed December 16, 2016.

Lehrig, S.; **Becker, M. (2014):** Approaching the cloud: Using palladio for scalability, elasticity, and efficiency analyses. In Proceedings of the Symposium on Software Performance., 26–28.

Levy, Y.; **Ellis, T. J. (2006):** A systems approach to conduct an effective literature review in support of information systems research. *Informing Science: International Journal of an Emerging Transdiscipline*, vol. 9 no. 1, 181–212.

Litoiu, M.; **Barna, C. (2013):** A performance evaluation framework for Web applications. *Journal of Software: Evolution and Process*, vol. 25 no. 8, 871–890, ISSN 2047–7481.

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 (COMPSAC). Beijing, China, ISSN 0730–3157, 327–334.

Lukasiewycz, M.; **Glaß, M.**; **Reimann, F.**; **Teich, J. (2011):** Opt4J - A Modular Framework for Meta-heuristic Optimization. In Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011). Dublin, Ireland, 1723–1730.

Lutteroth, C.; **Weber, G. (2008):** Modeling a Realistic Workload for Performance Testing. In Enterprise Distributed Object Computing Conference. IEEE, ISBN 1541–7719, 149–158.

March, S. T.; **Smith, G. F. (1995):** Design and natural science research on information technology. *Decision Support Systems*, vol. 15 no. 4, 251 – 266, ISSN 0167–9236.

Meier, J.; **Farre, C.**; **Bansode, P.**; **Barber, S.**; **Rea, D. (2007):** Performance testing guidance for web applications: patterns & practices. Microsoft Press, 288, ISBN 9780735625709.

Menascé, D. A. (2002): Load testing of Web sites. *IEEE Internet Computing*, vol. 6 no. 4, 70–74, ISSN 1089–7801.

Menascé, D. A. (2008): Computing missing service demand parameters for performance models. In Proceedings of the CMG Conference 2008., 241–248.

Menascé, D. A.; **Almeida, V. A. F.**; **Fonseca, R.**; **Mendes, M. A. (1999):** A Methodology for Workload Characterization of E-Commerce Sites. In Proceedings of the 1st Conference on Electronic Commerce (EC). ACM, ISBN 1–58113–176–3, 119–128.

Menascé, D. A.; **Almeida, V.**; **Riedi, R.**; **Ribeiro, F.**; **Fonseca, R.**; **Wagner Meira, J. (2000):** In search of invariants for e-business workloads. In Proceedings of the 2nd Conference on Electronic commerce (EC). ACM, 56–65.

Merkle, P.; **Henss, J. (2011):** EventSim–an event-driven Palladio software architecture simulator. *Karlsruhe Reports in Informatics (Palladio Days 2011 Proceedings)*, vol. 32, 15–22, ISSN 2190–4782.

Mizan, A.; **Franks, G. (2012):** Automated Performance Model Construction through Event Log Analysis. In 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation. IEEE, ISSN 2159–4848, 636–641.

Mosberger, D.; **Jin, T. (1998):** httperf-a tool for measuring web server performance. *SIGMETRICS Performance Evaluation Review*, vol. 26 no. 3, 31–37, ISSN 0163–5999.

Object Management Group, Inc. (2005): UML Profile for Schedulability, Performance, and Time (SPT), Version 1.1. `http://www.omg.org/spec/SPTP/1.1/`.

Object Management Group, Inc. (2013): Modeling and Analysis of Real-time Embedded Systems (MARTE)..

Ohmann, T.; **Herzberg, M.**; **Fiss, S.**; **Halbert, A.**; **Palyart, M.**; **Beschastnikh, I.**; **Brun, Y. (2014):** Behavioral Resource-aware Model Inference. In Proceedings of the 29th International Conference on Automated Software Engineering (ASE). ACM, ISBN 978–1–4503–3013–8, 19–30.

Peffers, K.; **Tuunanen, T.**; **Rothenberger, M. A.**; **Chatterjee, S. (2007):** A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, vol. 24 no. 3, 45–77.

Pelleg, D.; **Moore, A. W. et al. (2000):** X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In Proceedings of the 17th International Conference on Machine Learning (ICML)., 727–734.

Rodrigues, E. M.; **Saad, R. S.**; **Oliveira, F. M.**; **Costa, L. T.**; **Bernardino, M.**; **Zorzo, A. F. (2014):** Evaluating capture and replay and model-based performance testing tools: an empirical comparison. In Proceedings of the 8th International Symposium on Empirical Software Engineering and Measurement. ACM, 1–8.

Rolia, J. A.; **Sevcik, K. C. (1995):** The Method of Layers. *IEEE Transactions on Software Engineering*, vol. 21 no. 8, 689–700, ISSN 0098–5589.

Ruffo, G.; **Schifanella, R.**; **Sereno, M.**; **Politi, R. (2004):** WALTy: a user behavior tailored tool for evaluating Web application performance. In Proceedings of the 3th International Symposium on Network Computing and Applications. IEEE, 77–86.

Schulz, E. (2014): Integrating Performance Tests in a Generative Software Development Platform., Master's Thesis, Kiel University, Germany.

Schulz, E.; **Goerigk, W.**; **Hasselbring, W.**; **Hoorn, A. van**; **Knoche, H. (2014):** Model-Driven Load and Performance Test Engineering in DynaMod. In Proceedings of the Workshop on Model-based and Model-driven Software Modernization., 10–11.

Segall, I.; **Tzoref-Brill, R. (2015):** Feedback-driven Combinatorial Test Design and Execution. In Proceedings of the 8th ACM International Systems and Storage Conference. ACM, New York, NY, USA, SYSTOR '15, ISBN 978–1–4503–3607–9, 12:1–12:6.

Shams, M.; **Krishnamurthy, D.**; **Far, B. (2006):** A model-based approach for testing the performance of web applications. In Proceedings of the 3rd International Workshop on software quality assurance. ACM, 54–61.

Silveira, M. B. da; **Rodrigues, E. d. M.**; **Zorzo, A. F.**; **Costa, L. T.**; **Vieira, H. V.**; **Oliveira, F. M. de (2011):** Generation of Scripts for Performance Testing Based on UML Models. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE)., 258–263.

Simon, H. A. (1996): The sciences of the artificial. MIT press.

Sjoeberg, D. I. K.; **Hannay, J. E.**; **Hansen, O.**; **Kampenes, V. B.**; **Karahasanovic, A.**; **Liborg, N. K.**; **Rekdal, A. C. (2005):** A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, vol. 31 no. 9, 733–753, ISSN 0098–5589.

Smith, C. U.; Bernardo, M.; **Hillston, J. eds. (2007):** Introduction to Software Performance Engineering: Origins and Outstanding Problems. Springer Berlin Heidelberg, Berlin, Heidelberg, 395–428, ISBN 978–3–540–72522–0.

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 (ICPE '14). ACM, 227–228.

Steinberg, D.; **Budinsky, F.**; **Paternostro, M.**; **Merks, E. (2009):** EMF: Eclipse Modeling Framework. 2. edition. Addison-Wesley.

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 3th International Workshop on Large Scale Testing. ACM, ISBN 978–1–4503–2762–6, 5–6.

Vögele, C.; **Heinrich, R.**; **Heilein, R.**; **Hoorn, A. v.**; **Krcmar, H. (2015):** Modeling Complex User Behavior with the Palladio Component Model. In Symposium on Software Performance (SSP 2015). GI - Softwaretechnik-Trends.

Vögele, C.; **Hoorn, A. van**; **Krcmar, H. (2015):** Automatic Extraction of Session-Based Workload Specifications for Architecture-Level Performance Models. In Proceedings of the 4th International Workshop on Large-Scale Testing (LT). ACM, 5–8.

Vögele, C.; **Hoorn, A. van**; **Schulz, E.**; **Hasselbring, W.**; **Krcmar, H. (2016):** WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems. *Software & Systems Modeling*,, 1–35, ISSN 1619–1374.

Vögele, C.; **Hoorn, A. van**; **Schulz, E.**; **Hasselbring, W.**; **Krcmar, H. (2016):** WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems. Available: http://dx.doi.org/10.5281/zenodo.54859.

Vögele, C.; **Krcmar, H. (2016):** Multi-Objective Optimization of Load Test Designs. Available: https://doi.org/10.5281/zenodo.61911.

Walkinshaw, N.; **Taylor, R.**; **Derrick, J. (2013):** Inferring Extended Finite State Machine models from software executions. In Proceedings of the 20th Working Conference on Reverse Engineering. IEEE, 301–310.

Wang, W.; **Huang, X.**; **Qin, X.**; **Zhang, W.**; **Wei, J.**; **Zhong, H. (2012):** Application-Level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications. In Proceedings of the 2012 IEEE 6th International Conference on Cloud Computing (CLOUD 2012)., 439–446.

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 02767783.

WFMC Terminology (1999): Glossary (WFMC-TC-1011)..

Willnecker, F.; **Brunnert, A.**; **Gottesheim, W.**; **Krcmar, H. (2015):** 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.; **Krcmar, H. (2016):** Optimization of Deployment Topologies for Distributed Enterprise Applications. In Proceedings of the 12th International ACM Sigsoft Conference on the Quality of Software Architectures (QoSA). IEEE, 106–115.

Willnecker, F.; **Vögele, C.**; **Krcmar, H. (2016):** SiaaS: Simulation as a Service. In Proceedings of the Symposium on Software Performance. GI - Softwaretechnik-Trends.

Wohlin, C.; **Runeson, P.**; **Höst, M.**; **Ohlsson, M. C.**; **Regnell, B.**; **Wesslén, A. (2012):** Experimentation in software engineering. Springer Science & Business Media, ISBN 978–3642290435.

Woodside, M.; **Franks, G.**; **Petriu, D. C. (2007):** The Future of Software Performance Engineering. In Proceeding of the Future of Software Engineering. IEEE, 171–187.

Woodside, M.; **Petriu, D. C.**; **Petriu, D. B.**; **Shen, H.**; **Israr, T.**; **Merseguer, J. (2005):** Performance by Unified Model Analysis (PUMA). In Proceedings of the 5th International Workshop on software and performance (WOSP). ACM, ISBN 1–59593–087–6, 1–12.

Zhang, J.; **Cheung, S. C. (2002):** Automated test case generation for the stress testing of multimedia systems. *Software: Practice and Experience*, vol. 32 no. 15, 1411–1435, ISSN 1097–024X.

Zhang, P.; **Elbaum, S.**; **Dwyer, M. B. (2011):** Automatic Generation of Load Tests. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, Washington, DC, USA, ASE '11, ISBN 978–1–4577–1638–6, 43–52.

Zhao, L.; **Tian, J. (2003):** Testing the suitability of Markov chains as Web usage models. In Proceedings of the 27th Annual International Computer Software and Applications Conference. IEEE, ISBN 0730–3157, 356–361.

Zitzler, E.; **Thiele, L.**; **Laumanns, M.**; **Fonseca, C. M.**; **Fonseca, V. G. da (2003):** Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, vol. 7 no. 2, 117–132, ISSN 1089–778X.