Jürgen Christian Walter

Automation in Software Performance Engineering Based on a Declarative Specification of Concerns

Dissertation, Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik, 2019

Gutachter: Prof. Dr. Samuel Kounev, JMU Würzburg,
Prof. Dr. Ralf Reussner, KIT Karlsruhe

Tag der mündlichen Prüfung: 10. Dezember 2018

Abstract

Software performance is of particular relevance to software system design, operation, and evolution because it has a significant impact on key business indicators. During the life-cycle of a software system, its implementation, configuration, and deployment are subject to multiple changes that may affect the end-to-end performance characteristics. Consequently, performance analysts continually need to provide answers to and act based on performance-relevant concerns.

To ensure a desired level of performance, software performance engineering provides a plethora of methods, techniques, and tools for measuring, modeling, and evaluating performance properties of software systems. However, the answering of performance concerns is subject to a significant semantic gap between the level on which performance concerns are formulated and the technical level on which performance evaluations are actually conducted. Performance evaluation approaches come with different strengths and limitations concerning, for example, accuracy, time-to-result, or system overhead. For the involved stakeholders, it can be an elaborate process to reasonably select, parameterize and correctly apply performance evaluation approaches, and to filter and interpret the obtained results. An additional challenge is that available performance evaluation artifacts may change over time, which requires to switch between different measurement-based and model-based performance evaluation approaches during the system evolution. At model-based analysis, the effort involved in creating performance models can also outweigh their benefits.

To overcome the deficiencies and enable an automatic and holistic evaluation of performance throughout the software engineering life-cycle requires an approach that: (i) integrates multiple types of performance concerns and evaluation approaches, (ii) automates performance model creation, and (iii) automatically selects an evaluation methodology tailored to a specific scenario.

This thesis presents a *declarative* approach —called Declarative Performance Engineering (DPE)— to automate performance evaluation based on a human-readable specification of performance-related concerns. To this end, we separate the definition of performance concerns from their solution. The primary scientific contributions presented in this thesis are:

- *A declarative language to express performance-related concerns and a corresponding processing framework*: We provide a language to specify performance concerns independent of a concrete performance evaluation approach. Besides the specification of functional aspects, the language allows to include non-functional tradeoffs optionally. To answer these concerns, we provide a framework architecture and a corresponding reference implementation to process performance concerns automatically. It allows to integrate arbitrary performance evaluation approaches and is accompanied by reference implementations for model-based and measurement-based performance evaluation.

- *Automated creation of architectural performance models from execution traces*: The creation of performance models can be subject to significant efforts outweighing the benefits of model-based performance evaluation. We provide a model extraction framework that creates architectural performance models based on execution traces, provided by monitoring tools. The framework separates the derivation of generic information from model creation routines. To derive generic information, the framework combines state-of-the-art extraction and estimation techniques. We isolate object creation routines specified in a generic model builder interface based on concepts present in multiple performance-annotated architectural modeling formalisms. To create model extraction for a novel performance modeling formalism, developers only need to write object creation routines instead of creating model extraction software from scratch when reusing the generic framework.

- *Automated and extensible decision support for performance evaluation approaches*: We present a methodology and tooling for the automated selection of a performance evaluation approach tailored to the user concerns *and* application scenario. To this end, we propose to decouple the complexity of selecting a performance evaluation approach for a given scenario by providing solution approach capability models and a generic decision engine. The proposed capability meta-model enables to describe functional and non-functional capabilities of performance evaluation approaches and tools at different granularities. In contrast to existing tree-based decision support mechanisms, the decoupling approach allows to easily update characteristics of solution approaches as well as appending new rating criteria and thereby stay abreast of evolution in performance evaluation tooling and system technologies.

- *Time-to-result estimation for model-based performance prediction*: The time required to execute a model-based analysis plays an important role in different decision processes. For example, evaluation scenarios might require the prediction results to be available in a limited period of time such that the system can be adapted in time to ensure the desired quality of service. We propose a method to estimate the time-to-result for model-based performance prediction based on model characteristics and analysis parametrization. We learn a prediction model using performance-relevant features that we determined using statistical tests. We implement the approach and demonstrate its practicability by applying it to analyze a simulation-based multi-step performance evaluation approach for a representative architectural performance modeling formalism.

We validate each of the contributions based on representative case studies. The evaluation of automatic performance model extraction for two case study systems shows that the resulting models can accurately predict the performance behavior. Prediction accuracy errors are below 3% for resource utilization and mostly less than 20% for service response time. The separate evaluation of the reusability shows that the presented approach lowers the implementation efforts for automated model extraction tools by up to 91%. Based on two case studies applying measurement-based and model-based performance evaluation techniques, we demonstrate the suitability of the declarative performance engineering framework to answer multiple kinds of performance concerns customized to non-functional goals. Subsequently, we discuss reduced efforts in applying performance analyses using the integrated and automated declarative approach. Also, the evaluation of the declarative framework reviews benefits and savings integrating performance evaluation approaches into the declarative performance engineering framework. We demonstrate the applicability of the decision framework for performance evaluation approaches by applying it to depict existing decision trees. Then, we show how we can quickly adapt to the evolution of performance evaluation methods which is challenging for static tree-based decision support systems. At this, we show how to cope with the evolution of functional and non-functional capabilities of performance evaluation software and explain how to integrate new approaches. Finally, we evaluate the accuracy of the time-to-result estimation for a set of machine-learning algorithms and different training datasets. The predictions exhibit a mean percentage error below 20%, which can be further improved by including performance evaluations of the considered model into the training data.

The presented contributions represent a significant step towards an integrated performance engineering process that combines the strengths of model-

based and measurement-based performance evaluation. The proposed performance concern language in conjunction with the processing framework significantly reduces the complexity of applying performance evaluations for all stakeholders. Thereby it enables performance awareness throughout the software engineering life-cycle. The proposed performance concern language removes the semantic gap between the level on which performance concerns are formulated and the technical level on which performance evaluations are actually conducted by the user.

Zusammenfassung

Die Performanz von Software ist von herausgehobener Relevanz für das Design, den Betrieb und die Evolution von Softwaresystemen, da sie den Geschäftserfolg stark beeinflusst. Während des Softwarelebenszyklus ändern sich die Implementierung und die Art der Bereitstellung mehrfach, was jeweils das Ende-zu-Ende Verhalten bezüglich der Performanz beeinflussen kann. Folglich muss sich kontinuierlich mit Fragestellungen der Leistungsbewertung beschäftigt werden.

Um performantes Verhalten sicherzustellen gibt es im "Software Performance Engineering" bereits eine Vielzahl an Methoden, Techniken und Werkzeugen um Performanzeigenschaften von Softwaresystemen zu messen, zu modellieren und zu evaluieren. Jedoch unterliegt die Beantwortung von konkreten Fragestellungen einem Missverhältnis zwischen dem einfachen Formulieren von Fragestellungen und dem sehr technischen Level auf dem die Fragen beantwortet werden. Verfahren zur Bestimmung von Performanzmetriken haben unterschiedliche Stärken und Einschränkungen, u.a. bezüglich Genauigkeit, Lösungsgeschwindigkeit oder der erzeugten Last auf dem System. Für die beteiligten Personen ist es ein nicht-trivialer Prozess ein passendes Verfahren zur Performanzevaluation auszuwählen, es sinnvoll zu parametrisieren, auszuführen, sowie die Ergebnisse zu filtern und zu interpretieren. Eine zusätzliche Herausforderung ist, dass sich die Artefakte, um die Leistung eines Systemes zu evaluieren, im zeitlichen Verlauf ändern, was einen Wechsel zwischen messbasierten und modellbasierten Verfahren im Rahmen der Systemevolution nötig macht. Bei der modellbasierten Analyse kann zudem der Aufwand für die Erstellung von Performance-Modellen den Nutzen überwiegen.

Um die genannten Defizite zu überwinden und eine ganzheitliche, automatisierte Evaluierung der Leistung während des Software-Entwicklungszyklus zu erreichen ist ein Ansatz von Nöten, der: (i) unterschiedliche Arten von Performanzanliegen und Evaluationsmethoden integriert, (ii) die Erstellung von Performanzmodellen automatisiert und (iii) automatisch eine Methodik zur Evaluation zugeschnitten auf ein spezielles Analyseszenario auswählt.

Diese Arbeit präsentiert einen beschreibenden Ansatz, Declarative Performance Engineering (DPE) genannt, um die Evaluation von Performanzfragestellungen basierend auf einem menschenlesbaren Spezifikation zu automatisieren.

Zu diesem Zweck trennen wir die Spezifikation von Performanzanliegen von deren Beantwortung. Die wissenschaftlichen Hauptbeiträge dieser Arbeit sind:

- *Eine beschreibende Sprache um performanzrelevante Fragestellungen auszudrücken und ein Framework um diese zu beantworten*: Wir präsentieren eine Sprache, um Performanzanliegen unabhängig von der Evaluationsmethodik zu beschreiben. Neben der Spezifikation von funktionalen Aspekten können auch nicht-funktionale Abwägungsentscheidungen beschrieben werden. Um die spezifizierten Anliegen zu beantworten präsentieren wir eine Frameworkarchitektur und eine entsprechende Referenzimplementierung, um Anliegen automatisch zu beantworten. Das Framework bietet die Möglichkeit beliebige Evaluationsmethodiken zu integrieren und wird ergänzt durch Referenzimplementierungen zur messbasierten und modellbasierten Performanzevaluation.

- *Automatische Extraktion von architekturellen Performanzemodellen aus Messdaten zur Anwendungsperformanz*: Der signifikante Aufwand zur Erstellung von Performanzmodellen kann deren Vorteile überlagern. Wir schlagen einen Framework zur automatischen Erstellung vor, welches Modelle aus Messdaten extrahiert. Das präsentierte Framework trennt das Lernen von generischen Aspekten von Modellerstellungsroutinen. Um generische Aspekte zu lernen kombiniert unser Framework modernste Extraktions- und Schätztechniken. Wir isolieren Objekterstellungsroutinen, die in einer generischen Schnittstelle zur Modellerzeugung angegeben sind, basierend auf Konzepten die in mehreren Performanz-annotierten Architekturmodellen vorhanden sind. Um eine Modellextraktion für einen neuen Formalismus zu erstellen müssen Entwickler müssen nur die Erstellung von Objekterstellungsroutinen schreiben statt eine Modell-Extraktionssoftware von Grund auf neu zu schreiben.

- *Automatisierte und erweiterbare Entscheidungsunterstützung für Leistungsbewertungsansätze*: Wir präsentieren eine Methodik und Werkzeuge für die automatisierte Auswahl eines auf die Belange und Anwendungenszenarien der Benutzer zugeschnittenen Leistungsbewertungsansatzes. Zu diesem Zweck schlagen wir vor, die Komplexität der Auswahl eines Leistungsbewertungsansatzes für ein gegebenes Szenario zu entkoppeln. Dies geschieht durch Bereitstellung von Fähigkeitsmodellen für die Lösungsansätze und einen generische Entscheidungsmechanismus. Das vorgeschlagene Fähigkeits-Metamodell ermöglicht es, funktionale und nichtfunktionale Fähigkeiten von Leistungsbewertungsansätzen und Werkzeugen in verschiedenen Granularitäten zu modellieren. Im Ge-

gensatz zu bestehenden baumbasierten Entscheidungensmechanismen ermöglicht unser Ansatz die einfache Aktualisierung von Merkmalen von Lösungsansätzen sowie das Hinzufügen neuer Bewertungskriterien und kann dadurch einfach aktuell gehalten werden.

- *Eine Methode zur Schätzung der Analysezeit für die modellbasierte Leistungsvorhersage*: Die Zeit, die für die Durchführung einer modellbasierten Analyse benötigt wird, spielt in verschiedenen Entscheidungsprozessen eine wichtige Rolle. Beispielsweise können Auswertungsszenarien erfordern, dass die Vorhersageergebnisse in einem begrenzten Zeitraum zur Verfügung stehen, so dass das System rechtzeitig angepasst werden kann, um die Dienstgüte sicherzustellen. Wir schlagen eine Methode vor, um die Zeit bis zum Ergebnis für modellbasierte Leistungsvorhersage basierend auf Modelleigenschaften und Analyseparametrisierung zu schätzen. Wir lernen ein Vorhersagemodell anhand von leistungsrelevanten Merkmalen, die wir mittels statistischer Tests ermittelt haben. Wir implementieren den Ansatz und demonstrieren seine Praktikabilität, indem wir ihn auf einen mehrstufiger Leistungsbewertungsansatz anwenden.

Wir validieren jeden der Beiträge anhand repräsentativer Fallstudien. Die Evaluierung der Leistungsmodellextraktion für mehrere Fallstudiensysteme zeigt, dass die resultierenden Modelle das Leistungsverhalten genau vorhersagen können. Fehler bei der Vorhersagegenauigkeit liegen für die Ressourcennutzung unter 3% und meist weniger als 20% für die Service-Reaktionszeit. Die getrennte Bewertung der Wiederverwendbarkeit zeigt, dass der Implementierungsaufwand zur Erstellung von Modellextraktionswerkzeugen um bis zu 91% gesenkt werden kann.

Wir zeigen die Eignung unseres Framworks zur deklarativen Leistungsbewertung basierend auf zwei Fallstudien die mess- und model-basierte Leistungsbewertungstechniken zur Beantwortung verschiedenster Performance-Anliegen zugeschnitten auf Nutzerbedürfnisse anwenden. Anschließend diskutieren wir die Einsparungen durch den integrierten und automatisierten Ansatz. Des weiteren untersuchen wir die Vorteile der Integration von weiteren Leistungsbewertungsansätzen in den deklarativen Ansatz. Wir demonstrieren die Anwendbarkeit unseres Entscheidungsframeworks für Leistungsbewertungsansätze, indem wir den Stand der Technik für Entscheidungsunterstützung abbilden. Anschließend zeigen wir die leichte Anpassbarkeit, was für baumbasierte Entscheidungsunterstützungssysteme eine signifikante Herausforderung darstellt. Hierbei zeigen wir wie man Änderungen funktionaler und nichtfunktionaler Fähigkeiten von Leistungsbewertungssoftware sowie neue Ansätze integriert. Abschließend bewerten wir die Genauigkeit der Zeit-zu-Ergebnis-Schätzung

für eine Reihe von maschinellen Lernalgorithmen und verschiedenen Trainingsdatensätzen. Unser Vorhersagen zeigen einen mittleren prozentualen Fehler von weniger als 20%, die weiter verbessert werden können durch Berücksichtigung von Leistungsbewertungen des betrachteten Modells in den Trainingsdaten.

Die vorgestellten Beiträge sind ein bedeutender Schritt hin zu einem integrierten Performance-Engineering-Prozess, der die Stärken von modellbasierter und messbasierter Leistungsbewertung kombiniert. Die vorgeschlagene Sprache um Performanzanliegen zu spezifizieren reduziert in Verbindung mit dem Beantwortungsframework die Komplexität der Anwendung von Leistungsbewertungen für alle Beteiligten deutlich und ermöglicht dadurch ein Leistungsbewusstsein im gesamten Softwarelebenszyklus. Damit entfernt die vorgeschlagene Sprache die Diskrepanz zwischen einem einfachen Fragen bezüglich der Leistung und der sehr technische Ebene auf der Leistungsbewertungen tatsächlich ausgeführt werden.

Acknowledgements

This thesis is the result of the great support of many people, of which I can only name a few. First of all I would like to thank my advisor Prof. Dr. Samuel Kounev for the opportunity to be part of his research group. Samuel, I have always been inspired by your passion and dedication to work. My thanks also go to Prof. Dr. Ralf Reussner, who took on the role of the second reviewer and offered me a job at KIT so that I could be close to my family

I had the pleasure of being a part of the Descartes research group in Karlsruhe and Würzburg. It was always a joy working with you: Dr. Simon Spinner, Simon Eismann, Johannes Grohmann, Lukas Iffländer, Dr. Nikolaus Huber, Dr. Fabian Brosig, Dr. Jóakim von Kistowski, Dr. Nikolas Herbst, Fritz Kleemann, and Susanne Stenglin. Keep on collaborating and supporting each other with discussions in the coffee corner, the meetings of the research group, or late in a bar where some ideas were born. I have enjoyed as well my time with the SDQ colleagues in Karlsruhe: Dr. Christian Stier, Dr. Robert Heinrich, Misha Strittmatter, and Sandro Koch, among others. Moreover, my research was supported by many committed and ambitious students. Thank you Simon Eismann, Nikolai Reed, Cristina Llamas Beltran, Katharina Dietz, Maximilian König, Andreas Knapp, Christoph Thiele, and Adrian Hildebrandt for supporting my research and tool development in various projects.

I would also like to thank the DFG and ABB for funding and supporting my ideas with research grants. I am thankful to Dr. Heiko Koziolek from ABB for his valuable feedback on the automated creation of architectural performance models. To my colleagues from the University of Stuttgart: Dr. André van Hoorn and Dr. Dušan Okanović, it was always great fun to discuss ideas on declarative performance engineering in our joint DFG DECLARE project.

The international exchange of ideas on conferences and research retreats was valuable to me. Especially, working together in the SPEC RG DevOps Performance Working Group has been a constant source of inspiration. Prof. Dr. Cor-Paul Bezemer, Vincenco Ferme, Dr. Felix Willnecker, and Dr. Andreas Brunnert, our joint work broadened my view on performance engineering.

Finally, I would like to thank my wife Barbara for supporting me and enduring my absences. Thank you for bearing my enthusiasm for research and for putting me back on track after setbacks. Thank you!

Publication List

Peer-Reviewed International Journal Articles

[WSK15b] Jürgen Walter, Simon Spinner, and Samuel Kounev. "Parallel Simulation of Queueing Petri Nets". In: *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems* 16.8 (Aug. 2015).

Peer-Reviewed International Conference Contributions

Full Research Papers

[Eis+19] Simon Eismann, Jürgen Walter, Jóakim von Kistowski, and Samuel Kounev. "Modeling of Parametric Dependencies for Performance Prediction of Component-based Software Systems at Run-time". In: *2018 IEEE International Conference on Software Architecture (ICSA 2018)*, Seattle, WA, USA, Apr. 2018, Acceptance Rate: 25,6%. Simon Eismann, Johannes Grohmann, Jürgen Walter, Jóakim von Kistowski, and Samuel Kounev. "Integrating Statistical Response Time Models in Architectural Performance Models". In: *2019 IEEE International Conference on Software Architecture (ICSA)*. Hamburg, Germany, Mar. 2019, Acceptance Rate: 21,9%

[Eis+18] Simon Eismann, Jürgen Walter, Jóakim von Kistowski, and Samuel Kounev. "Modeling of Parametric Dependencies for Performance Prediction of Component-based Software Systems at Run-time". In: *2018 IEEE International Conference on Software Architecture (ICSA 2018)*, Seattle, WA, USA, Apr. 2018, Acceptance Rate: 25,6%.

[WHK17] Jürgen Walter, André van Hoorn, and Samuel Kounev. "Automated and Adaptable Decision Support for Software Performance Engineering". In: *Proceedings of the 11th EAI International Conferenceon Performance Evaluation Methodologies and Tools (VALUETOOLS 2017)*. Venice, Italy, Dec. 2017

[Hub+15] Nikolaus Huber, Jürgen Walter, Manuel Bähr, and Samuel Kounev.

"Model-based Autonomic and Performance-aware System Adaptation in Heterogeneous Resource Environments: A Case Study". In: *Proceedings of the 2015 IEEE International Conference on Cloud and Autonomic Computing (ICCAC 2015)*. Cambridge, MA, USA: IEEE, Sept. 2015

[WSK15a] Jürgen Walter, Simon Spinner, and Samuel Kounev. "Parallel Simulation of Queueing Petri Nets". In: *Proceedings of the Eighth EAI International Conference on Simulation Tools and Techniques (SIMUTools 2015)*. Athens, Greece, Aug. 2015

Vision Papers

[Iff+18b] Lukas Iffländer, Jürgen Walter, Simon Eismann, and SamuelKounev. "The Vision of Self-aware Reordering of SecurityNetwork Function Chains". In: *Proceedings of the 9th ACM/SPEC International Conference on Performance Engineering (ICPE 2018)*. Berlin, Germany, Apr. 2018

[Wal+16a] Jürgen Walter, André van Hoorn, Heiko Koziolek, Dušan Okanović, and Samuel Kounev. "Asking "What?", Automating the "How?": The Vision of Declarative Performance Engineering". In: *Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering (ICPE 2016)*. Delft, the Netherlands, Mar. 2016.

Short, Tutorial, and Tool Papers

[Bez+19] Cor-Paul Bezemer and Simon Eismann and Vincenzo Ferme and Johannes Grohmann and Robert Heinrich and Pooyan Jamshidi and Weiyi Shang and André van Hoorn and Mónica Villavicencio and Jürgen Walter and Felix Willnecker. "How is Performance Addressed in DevOps?" In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering (ICPE 2019)*. Mumbai, India, Apr. 2019.

[Iff+18a] Lukas Iffländer, Stefan Geißler, Jürgen Walter, Lukas Beierlieb, and Samuel Kounev. "Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking". In: *Proceedings of the 9th Symposium on Software Performance 2018 (SSP 2018)*. Hildesheim, Germany, Nov. 2018

[Wal+18] Jürgen Walter, Simon Eismann, Johannes Grohmann, Dušan Okanović, and Samuel Kounev. "Tools for Declarative Performance Engineering". In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE 2018)*. Berlin, Germany: ACM, Apr. 2018.

[Wal+17a] Jürgen Walter, Simon Eismann, Nikolai Reed, and Samuel Kounev. "Providing Model-Extraction-as-a-Service for Architectural Performance Models". In: *Proceedings of the 2017 Symposium on Software Performance (SSP 2017)*. Karlsruhe, Germany, Nov. 2017.

[Dül+17] Thomas F. Düllmann, Robert Heinrich, André van Hoorn, Teerat Pitakrat, Jürgen Walter, and Felix Willnecker. "CASPA: A Platform for Comparability of Architecture-based Software Performance Engineering Approaches". (Demo Paper). In: *Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA 2017)*. Gothenburg, Sweden: IEEE, Apr. 2017.

[Wal+16b] Jürgen Walter, Maximilian König, Simon Eismann, and Samuel Kounev. "PAVO: A Framework for the Visualization of Performance Analyses Results". In: *Proceedings of the 2016 Symposium on Software Performance (SSP 2016)*. Kiel, Germany, Nov. 2016

[WEH15] Jürgen Walter, Simon Eismann, and Adrian Hildebrandt. "Automated Transformation of Descartes Modeling Language to Palladio Component Model". In: *Proceedings of the 2015 Symposium on Software Performance (SSP 2015)*. Munich, Germany, Nov. 2015.

Peer-Reviewed International Workshop Contributions

[Wal+17c] Jürgen Walter, Christian Stier, Heiko Koziolek, and Samuel Kounev. "An Expandable Extraction Framework for Architectural Performance Models". In: *Proceedings of the 3rd International Workshop on Quality-Aware DevOps (QUDOS 2017)*. l'Aquila, Italy: ACM, Apr. 2017

[WOK17] Jürgen Walter, Dušan Okanović, and Samuel Kounev. "Mapping of Service Level Objectives to Performance Queries". In: *Proceedings of the 2017 Workshop on Challenges in Performance Methods for Software Development (WOSP-C'17) co-located with 8th ACM/SPEC International Conference on Performance Engineering (ICPE 2017)*. l'Aquila, Italy: ACM, Apr. 2017

[SWK16] Simon Spinner, Jürgen Walter, and Samuel Kounev. "A Reference Architecture for Online Performance Model Extraction in Virtualized Environments". In: *Proceedings of the 2016 Workshop on Challenges in Performance Methods for Software Development (WOSP-C 2016) co-located with 7th ACM/SPEC Interna-*

tional Conference on Performance Engineering (ICPE 2016). Delft, the Netherlands, Mar. 2016

Peer-Reviewed Book Chapters

[SWK17] Klaus Schilling, Jürgen Walter, and Samuel Kounev. "Spacecraft Autonomous Reaction Capabilities, Control Approaches and Self-Aware Computing". In: *Self-Aware Computing Systems*. Ed. by Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu. Berlin Heidelberg, Germany: Springer Verlag, 2017.

[Wal+17b] Jürgen Walter, Antinisca Di Marco, Simon Spinner, Paola Inverardi, and Samuel Kounev. "Online Learning of Run-time Models for Performance and Resource Management in Data Centers". In: *Self-Aware Computing Systems*. Ed. by Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu. Berlin Heidelberg, Germany: Springer Verlag, 2017

Technical Reports and Manuals

[Gro+17] Johannes Grohmann, Simon Eismann, Jürgen Walter, and Samuel Kounev. Descartes Modeling Language - Quick Start Guide. University of Würzburg, Am Hubland, Informatikgebäude, 97074 Würzburg, Germany, Apr. 2017.

[Bru+15] Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Herbst, Pooyan Jamshidi, Reiner Jung, Jóakim von Kistowski, Anne Koziolek, Johannes Kroß, Simon Spinner, Christian Vögele, Jürgen Walter, and Alexander Wert. Performance-oriented DevOps: A Research Agenda. Tech. rep. SPEC-RG-2015-01. SPEC Research Group—DevOps PerformanceWorking Group, Standard Performance Evaluation Corporation (SPEC), Aug. 2015

Contents

Chapter 1

Introduction

1.1 Motivation

Modern software systems are expected to provide responsive services. Performance is one of the most crucial factors for successful software systems. Poor performance leads to a loss in revenue for companies relying on the availability of their systems and to penalties for service providers violating performance requirements. Performance is a software quality attribute that captures the degree to which a system satisfies its requirements concerning timing behavior and resource efficiency. It is of particular relevance to software system design, operation, and evolution because it has a significant impact on key business indicators and therefore affects all stakeholders. We understand stakeholders as individuals, teams, organizations, or classes thereof, having an interest in a system, as defined in [ISO11]. Consequently, during the life-cycle of a software system, performance analysts need to continually provide answers to and act on performance-relevant questions such as about response times, hardware utilization, bottlenecks, trends, or anomalies.

Evolution and change are inevitable for successful software systems as their functional and non-functional requirements, execution environments, and usage change over time [Bru+15]. Therefore, performance management should be an ongoing task to ensure performance requirements and resource efficiency [Bru+15]. Software and its deployment must be continually adapted in order not to become progressively less satisfactory [LR01]. Independent of the applied software development model, software passes several life-cycle stages, including design, testing, deployment, and operation. Each of these stages is connected to different performance evaluation artifacts.

During the past decades, various established methods, techniques, and tools for modeling and evaluating performance properties have been proposed—ranging from model-based prediction and optimization in early design, over measurement-based approaches in phases where implementations exist,

up to solutions that continuously evaluate and control system performance during operation.

In principle, techniques addressing the aforementioned performance-related questions and goals are available. However, there is no unifying engineering approach nor tooling to support holistic performance management [WFP07]. While there are manifold performance analysis approaches, several challenges hinder these techniques from being widely applied in practice including: (i) how to choose between the various available approaches, (ii) how to parameterize and apply the chosen approaches, and (iii) how to filter and interpret the obtained results.

Currently, there is a significant abstraction gap between the level on which performance concerns are formulated and the level on which performance evaluations are actually conducted. For example, instead of requiring to intuitively specify constraints for performance analysis, like desired accuracy, time-to-result, or system overhead, it is often required to specify indirect parameters like experiment length or sampling rate. The configuration of performance measurements, as well as the creation and analysis of performance models, often rely on external expert knowledge and significant manual efforts. Required significant efforts prevent developers, operators, as well as other stakeholders from carrying out regular performance assessments.

Instead of being performed by all stakeholders, Software Performance Engineering (SPE) activities are often forwarded to a separate performance expert or a quality assurance team. While one could argue that the detour via experts and manual effort is tolerable in lengthy processes, modern agile processes, in which deployments take place weekly, daily, or even hourly do not allow for such efforts. This fast-paced development requires a unifying engineering approach *and* tooling to support holistic performance management that is automated and easy to use.

1.2 Problem Statement

In the following, we assume a software system life-cycle in which performance analysts need to continually provide answers to and act on performance-relevant concerns. We assume that the application implementation, deployment, and software usage patterns can and will change over time. The services provided by the application are subject to objectives on the performance and availability in terms of service-level metrics, such as end-to-end response time or throughput. Besides ensuring adequate quality of service, a second primary target is cost-efficient resource management that provisions only the required

resources. To enable resource management, we assume that resources can be provisioned and de-provisioned.

Performance concerns can be answered using various reactive measurement-based and proactive model-based analysis approaches and tools each coming with different advantages, disadvantages, and limitations. For performance evaluation within the software development life-cycle, we see the following major challenges:

- *Frequent system changes*: The system environment may frequently change due to changing requirements or varying workloads triggered by human users or external IT services. This volatility of the operation environment can imply updates of the application code or hardware infrastructure to ensure Service-level Agreements (SLAs) and resource efficiency. Faster release and deployment cycles and have become more and more widespread in practice. In consequence, this may result in a frequently recurring need for performance analysis. Evaluations include, on the one hand, the analysis of the actual system state, on the other hand, what-if analyses investigating potential future system states.

- *Different evaluation artifacts*: Performance can be evaluated using different evaluation artifacts. Before implementation, there can be performance-annotated design-time models allowing for rough estimations in early development stages. Later, after their implementation, software artifacts can be deployed to measure performance indices. Then, also parameterization of performance models can be based on measurement results. In addition to the models for offline analysis, models can be used to proactively reconfigure a system at run-time. The different fields of application are usually associated with different modeling formalisms. The various performance evaluation artifacts come with different strengths and limitations that need to be understood by a performance analyst in order to apply them correctly and efficiently.

- *Manifold performance evaluation approaches*: There are multiple different ways to measure software systems (e.g., using sampling or not, different software tools) and to analyze performance models (e.g., analytical or simulation-based). Each analysis strategy comes with different capabilities, strengths, and limitations. Besides conceptual differences, a disparity in the implementations can lead to significantly different capabilities of the tools. Additionally, the suitability of an analysis tool depends on the system implementation and modeling approach. Numerous performance evaluation approaches and the system itself have to be understood by

a performance analyst in order to be able to select a suitable analysis strategy.

- *Automation in performance model creation*: The creation of performance models is still associated with significant manual effort. Moreover, there are different performance modeling formalisms and corresponding analysis tools for performance evaluation at design-time and run-time.

- *Heterogeneous infrastructure*: System landscapes often contain a variety of different technologies that may evolve over time. The applied technology stack limits applicable performance evaluation tooling. Each technology requires a tailored analysis tooling implementation. It is not obvious when to use which performance evaluation tooling. Especially for measurement-based performance evaluation, available tools are mostly technology-bound.

To support SPE throughout the software life-cycle a holistic approach is needed that allows for: (i) end-to-end automation, (ii) flexibility of the applied solution approach, and (iii) optimization and processing tailored to the specific concerns. In particular, performance engineering requires means to specify performance concerns independent of the underlying system and performance evaluation approach. Further, end users require an automated response tailored to specific evaluation scenario conditions (available evaluation artifacts) and needs (acceptable system overhead, required time-to-result, desired accuracy). Further, the approach has to support various measurement-based and model-based performance evaluation strategies. As the creation and maintenance of performance models can be time-consuming and requires in-depth knowledge of the performance behavior of a system, we cannot expect performance analysts to provide them in advance. To allow for a seamless switch back and forth between model-based and measurement-based analysis, we need methods that extract models of the system automatically.

1.3 State-of-the-Art

Initial approaches to model and interpret performance requirements, queries, and goals have been proposed. However, these approaches are either limited to metrics *or* SLAs, but not both. Existing frameworks to investigate performance metrics lack the integration of algorithms on top of primary metrics to support more complex analyses. While there is a multitude of performance evaluation approaches, the resolution of performance concerns is usually associated with

a single performance evaluation strategy. Current approaches to performance and resource management strictly separate between measurement-based and model-based performance evaluation. Mostly, there is even a strict separation between design-time and run-time evaluation, each with separate tooling. The landscape of performance evaluation approaches is broad. Therefore, holistic performance management throughout the system life-cycle currently requires to adopt and integrate multiple separate toolchains. In general, performance testing and management is associated with significant manual efforts performed by a small group of experts [LB17; Heg+17]. Available tooling for performance evaluation is far from supporting end-to-end automation.

SPE provides a plethora of methods, techniques, and tools for measuring, modeling, and evaluating performance properties of software systems. These performance evaluation approaches come with different strengths and limitations concerning, for example, accuracy, time-to-result, or system overhead. To answer performance-related concerns requires to select a performance evaluation strategy. The chosen analysis strategy profoundly affects accuracy and analysis costs. However, there is little decision support on how to select a suitable analysis tooling tailored to a given performance analysis need. To acquire the required knowledge to select an appropriate solution strategy from multiple performance evaluation approaches can be challenging, as their functional (e.g., supported technologies) and non-functional (e.g., pricing, overhead, accuracy) capabilities may evolve independently over time. Besides the challenge to become an expert once, there is a continuous evolution of performance evaluation tools concerning their functional (e.g., new measurable technologies) and non-functional (e.g., pricing models or system overhead) properties. Also, the technologies applied to software system development evolve which affects the applicability of existing tools. Currently, there are only a few tree-based solutions automating decision support, as presented in [Bro+15] or [Bol+06]. However, none of them considers the evolution of SPE tools and approaches.

A second major problem for decision support is that we do not know about the time required for model-based analysis. While model-based performance prediction at run-time implies that there are usually upper limits on time-to-result, there is no established method to estimate the time-to-result for a given prediction scenario. Instead, a qualitative classification into slow or fast approaches depicts the state-of-the-art. The estimation of time-to-result is challenging as the computational overhead depends on model characteristics *and* analysis parametrization. Also, the response time behavior of alternative solvers may depend on *different* characteristics of the model and the analysis configuration. For example, some model-based analysis approaches are

insensitive to model complexity but sensitive to workload definition [Mül+16].

Measurement-based performance analysis became more and more popular in recent years driven by established companies like CA Technologies, IBM, HP/HPE, Compuware (DynaTrace), New Relic, Riverbed, and AppDynamics. Their yearly software revenue is over one billion U.S. dollar and expected to grow further [KC14; Cam15].

In contrast to measurement-based analysis, predictive performance modeling has a smaller commercial distribution. Model-based predictions allow for an exploration of alternative deployments, architectures, and configurations without the need to test them in a live system. However, it can be challenging to derive performance models. Therefore, model-based analysis does not have a broad industrial acceptance yet. There are only a few professional providers for automated performance model creation including small, innovative companies such as [Per18], [Sum18], [Vis18], or [RET18]. Mostly, model-based analysis methods and tools are largely untested in an industrial setting outside the academic research groups where they originated [Goo+12]. Low-level stochastic modeling formalisms, popular in research, are not straightforward for layman users making them hardly suitable for a broad industrial application [Goo+12].

1.4 Approach and Contributions

In this thesis, we develop a novel *declarative* approach for SPE, called Declarative Performance Engineering (DPE), to provide automated performance analyses throughout the software development life-cycle and thereby overcome the named limitations. DPE aims to separate the specification of performance concerns from their resolution. In particular, we specify performance concerns using a Domain Specific Language (DSL). Based on the specification, we automate the processing of concerns using measurement-based and model-based performance evaluation techniques. Building on a generic processing framework, we provide reusable analysis parts such as algorithms to process manifold concerns as well as automatic result filtering and graphical presentation. Moreover, we provide means to automatically generate models from measurements to seamlessly switch between measurement-based and model-based analysis. The model-based analysis of the presented approach uses descriptive architecture-level performance models supporting different mathematical solution algorithms for reasoning. These models allow for a representation of the system architecture and its major performance-influencing factors at different granularities. All efforts of this thesis target facilitating and automating perfor-

mance evaluation, and thereby enable a broader application of performance evaluations and a holistic performance management.

1.4.1 Contributions

A Declarative Framework to Answer Performance Related Concerns

There is a significant semantic gap between the level on which performance concerns are formulated and the technical level on which performance evaluations are conducted. For layman users and experts, it is not always straightforward to reasonably parameterize and correctly apply performance evaluation approaches, and to filter and interpret the obtained results. Therefore, our goal is to automate SPE based on a human-readable specification of performance-related concerns. To answer these concerns, we provide a framework that allows to plug in arbitrary performance evaluation approaches. We make the following contributions as part of our vision of DPE:

- We provide a DSL to specify performance concerns that includes several performance analysis concepts independent of a concrete performance evaluation approach. Besides the specification of functional aspects, our language allows appending optimization of non-functional processing attributes like accuracy, time-to-result, or system overhead.

- We provide a framework architecture and reference implementation to process performance concerns automatically. The framework allows to plug in arbitrary performance evaluation approaches. We implement exemplary adapters for model-based and measurement-based performance evaluation.

- We integrate generic performance concern processing algorithms into the framework, such as SLA generation and evaluation. Existing SLA evaluation infrastructure is usually limited to a single evaluation strategy which impedes continuous SLA evaluation in all stages of the software life-cycle. In contrast, our approach allows for holistic SLA management.

The vision of DPE has been published in [Wal+16a]. An initial version of the adapter to connect measurement-based analysis has been published in [Blo+16], the visualization in [Wal+16b], and parts of the SLA evaluation in [WOK17]. All implementations were realized as extensions of the Descartes Query Language (DQL) framework. The work on DPE was partially supported by the German Research Foundation (DFG) under grant No. KO 3445/15-1.

Automated Creation of Architectural Performance Models

Architectural performance models can be leveraged to explore performance properties of software systems at design-time and run-time. Many modeling formalisms have been proposed for different domains and applications. At design-time, architectural performance models support reasoning on effects of design decisions. At run-time, they enable automatic reconfigurations by reasoning on the effects of changing user behavior. Building models from scratch in an editor does not scale for medium and large-scale systems in an industrial context. Efforts for manual modeling are significant, but also tools for automated model extraction are complex and challenging to build. So far, model extraction approaches do not generalize over multiple formalisms; they normally support only a single specific formalism. The independent implementation of model extraction for multiple formalisms leads to significant initial costs and long-term maintenance efforts when maintaining each tool separately.

In this thesis, we present a model extraction framework to create architectural performance models using execution traces which generalizes over the targeted architectural modeling language.

- We provide the first publicly available architectural performance model extraction framework, called Performance Model eXtractor (PMX), which derives architectural performance models based on execution traces and realizes our proposed method for performance model extraction. Our framework integrates state-of-the-art extraction and estimation techniques with object creation routines that apply to many architectural performance models.

- We propose a generic model builder interface based on concepts present in multiple performance-annotated architectural description languages. Then, we separate the derivation of generic information from object creation routines specified in the builder interface. To create a model extraction for a new performance modeling formalism, developers only need to write object creation routines instead of creating model extraction software from scratch. The reuse of the framework lowers the effort to implement performance model extraction tools by up to 91% through a high level of reuse. Further, our approach allows for complementary use of different analysis toolchains accessible through builder implementations for different architectural performance modeling formalisms.

- We introduce and discuss providing Model-Extraction-as-a-Service (MEaaS) for architectural performance models. Existing open-source performance model extraction approaches imply significant initial efforts that might be challenging for layman users. To simplify usage, we provide the extraction of architectural performance models based on application execution traces as a web service. MEaaS solves the usability problem and lowers the initial effort of applying model-based analysis approaches. We provide web services for two architectural performance modeling formalisms and discuss future security and scalability challenges. Besides usability improvements, maintainability efforts can be reduced. Further, a central service provisioning may allow collecting data that can be used as a basis for further improving model extraction.

Alternative methodologies for performance model extraction have been described in [Wal+17b]. The PMX tool, as well as the idea of the generic builder interface, have been published in [Wal+17c]. We explained providing model extraction as a web service for two architectural performance modeling formalisms in [Wal+17a]. The tool PMX has been applied in cooperation with Forschungszentrum Informatik (FZI) research transfer institute in [Wal+17c] and also independently by researchers from FZI in [Rom17]. The development of PMX was partially supported by the DFG under the grant KO 3445/11-1 and in cooperation with ABB as part of an Academic Research Award ($80,000) awarded in the area of industrial software systems. The ABB Research Grant Program is a highly competitive international program.

Automated and Extensible Decision Support for Software Performance Engineering

The declarative approach presented in this thesis allows for seamless changeability of evaluation approaches based on transformations of monitoring data and models. While the performance evaluation approaches allow for interchangeability, the choice of an appropriate approach and tooling to solve a given performance concern still relies on expert knowledge that can quickly become obsolete. We present a methodology and tooling for the automatic selection of performance engineering approaches tailored to the user concerns *and* application scenario. We propose to decouple the complexity of selecting an SPE approach for a given scenario by using solution approach capability models coupled with a generic decision engine. Concerning decision support for SPE, we make the following contributions:

- We provide an automated decision framework and a corresponding capability meta-model to describe performance evaluation techniques. The framework recommends an evaluation approach from a set of registered performance evaluation techniques tailored to a specific evaluation scenario. The framework considers the user concern and the characteristics of the system under consideration. The proposed tool capability meta-model enables describing functional and non-functional capabilities of performance evaluation approaches and tools at different granularities. In contrast to existing tree-based decision support mechanisms, our approach explicitly considers to modify solution approaches and rating criteria and thereby stay abreast of performance evaluation tool and system evolution.

- We propose an approach to estimate the time-to-result for model-based performance prediction based on model characteristics and analysis parametrization. We determine performance-relevant features using statistical tests. We implement the approach and demonstrate its practicability by applying to a simulation-based multi-step analysis of an architectural performance model. This allows evaluating the suitability of a model solver to match time-to-result constraints for a given prediction scenario.

The decision framework and the corresponding capability meta-model have been published in [WHK17]. The work on decision support was partially supported by the German Research Foundation (DFG) under grant No. KO 3445/15-1.

1.4.2 Evaluation

This thesis consists of three main building blocks that we evaluate separately to allow for a focused evaluation of each block.

Automated Model Creation The evaluation of performance model extraction for several case study systems shows that the resulting models can accurately predict the performance behavior. Prediction accuracy errors are below 3% for resource utilization and mostly less than 10% for service response time. While, without the use of our framework, performance model extraction software has to be implemented from scratch, the separate evaluation of reusability shows that it lowers the implementation efforts for automated model extraction tools by up to 91%. Using our framework, model extraction for a specific

architectural performance modeling language requires only to implement the object creation routines of our builder interface. As an additional benefit, our approach simplifies the complementary use of multiple analysis toolchains for different formalisms.

Language and Framework The evaluation of our DPE-framework answers how our methodology can be applied to the software development life-cycle of a real-world application by means of two case study systems. It demonstrates that our declarative approach suits well to holistically investigate performance by applying different performance evaluation techniques in different stages of the software life-cycle. Our evaluation of the declarative performance analysis framework further underlines its extensiveness and novelty by a features comparison to related approaches.

Decision Support We demonstrate the applicability of our decision framework for performance engineering approaches by presenting how to depict existing decision support systems. At this, we consider qualitative and quantitative attributes. Subsequently, we demonstrate the ability to cope with the evolution within SPE. We show how to cope with the evolution of functional and non-functional capabilities of evaluation software and explain how to integrate new approaches. The evaluation discusses decision support for measurement-based and model-based analysis approaches separately as well as in combination.

Time-to-Result Estimation Considering a representative model-based analysis approach, we evaluate the time-to-result estimation accuracy for a selection of machine-learning algorithms and different training dataset sizes. Moreover, the evaluation considers time-to-result estimations for models not included in the training data and for models where alternative analysis configurations have been used for training. The results obtained in this study show that based on a small set of training models, accurate time-to-result estimation for model-based performance evaluation can be achieved. Our predictions show a mean deviation below one second and a mean percentage error below 20% percent for unseen performance models, which can be further improved by including performance evaluations of the considered model into the training data.

1.5 Application Scenarios and Assumptions

In this thesis, we propose the DPE approach to automate SPE during the software life-cycle. At this, we focus on performance awareness during the

software development life-cycle rather than on self-aware system adaptation at run-time. Even though the approaches presented in this thesis can be applied to achieve self-aware resource management, we focus on how to obtain the information required to act.

The DPE approach presented in this thesis does not assume a specific software development model. In general, all can benefit from an automatic answering of performance concerns. Further, we do not make assumptions on whether the system consists of existing modules or is built from scratch. We support development from scratch (referred to as greenfield development) as well as the development and deployment of new software systems in the presence of existing software systems (referred to as brownfield development) [HJ08].

In general, we do not restrict our DPE approach to a particular type of system or technology stack. However, we cannot evaluate all types of software systems. In particular, our evaluation investigates an RSS reader application provided by Netflix and the Petclinic portal application. The former is a microservice-based application; the second system represents a monolithic application. Moreover, it is out of scope for this thesis to technically integrate and evaluate the plethora of popular performance evaluation approaches. Our technical implementation is limited to the Kieker monitoring framework [HWH12] for measurement-based performance evaluation and the Descartes Modeling Language (DML) [Hub+17] for model-based performance evaluation. Especially, the fully automated performance model extraction is subject to several limitations discussed in the respective chapter. We assume applications that have a component-based software architecture for which we can control the underlying physical and virtual resource infrastructure. There can be particular challenges caused by specific technology stacks or organizational issues, for example, limited access and control in public clouds (no network throughput and latency guarantees, control only at provider level) or isolation of performance at different layers in virtualized or containerized environments which have not been investigated in this thesis.

1.6 Thesis Outline

The thesis is organized as follows:

- Chapter 2 introduces the theoretical and technical foundations on SPE required to understand the contributions of this thesis. First, we introduce measurement-based and model-based performance evaluation. Second, we present terminology and foundations on performance modeling for-

malisms. Finally, we briefly discuss existing performance prediction techniques.

- Chapter 3 surveys related work. First, we discuss the specification and interpretation of performance-related concerns. Second, we study the state-of-the-art on the evaluation, comparison, and selection of performance evaluation approaches and tooling. Then, we study the state-of-the-art on performance model creation. Finally, we discuss the integration of SPE in the development life-cycle.

- Chapter 4 introduces the idea of DPE that decouples the specification of performance-related concerns from their resolution. Following this idea, we propose a domain specific language to specify performance-related concerns and a framework to automatically answer them based on the underlying model-based and measurement-based performance evaluation techniques.

- Chapter 5 presents the automated extraction of architectural performance models. Preceding to the model extraction, we discuss shared concepts among architectural performance modeling formalisms. Then, we introduce a framework, called PMX, to derive performance models based on application performance execution traces. It abstracts from concrete architectural performance models and describes a generic model builder interface to generate performance models.

- Chapter 6 presents our methodology to provide decision support for performance evaluation approaches that explicitly considers characteristics of system under consideration and the specific concern to be solved. We propose a novel recommendation architecture that explicitly considers the evolution of performance evaluation tooling and later changes of rating criteria. Then, we propose our solution strategy capabilities meta-model and the corresponding solution strategy decision engine including generic algorithms for an automated choice. Finally, we present a methodology to estimate the time-to-result analysis costs for model-based analysis based on machine-learning algorithms.

- Chapter 7 contains the evaluation of our declarative approach to performance engineering. First, it evaluates the prediction accuracy of automatically extracted performance models, quantifies savings compared to manual creation, and assesses effort reduction when reusing generic model extraction parts. Secondly, we evaluate our decision framework by providing decision support systems for model-based, measurement-based,

as well as holistic performance analysis also comparing both among each other. At this, we discuss modeling capabilities as well as setup and maintenance efforts in comparison to state-of-the-art tree-based decision support. Third, we evaluate efforts and accuracy for time-to-result estimation investigating different training datasets and machine-learning algorithms. Finally, we evaluate effort reduction and reuse of the performance concern language and processing framework.

- Chapter 8 concludes this thesis by summarizing its contributions and benefits, and provides an overview of areas for future work.

Part I

Foundations and Related Work

Chapter 2

Foundations of Software Performance Engineering

This thesis applies and extends the state-of-the-art of Software Performance Engineering (SPE). Therefore, this chapter provides foundations on SPE which is a sub-discipline of software engineering. It addresses the challenges related to application performance by means of an engineering discipline. [Smi81], who invented the term, states that it is a systematic quantitative approach to the cost-effective development of software systems to meet performance requirements. SPE encompasses the techniques applied during a systems life-cycle to ensure the non-functional requirements for performance (such as throughput, latency, or memory usage) will be met. [WFP07] expands the definition by stating that SPE represents the entire collection of software engineering activities and related analyses used throughout the software development cycle, to met performance requirements.

SPE can be investigated at different abstraction levels —structuring the following sections: In the following, we provide a brief introduction to measurement-based performance evaluation in Section 2.1 and model-based performance evaluation in Section 2.2. Next, Section 2.3 introduces performance modeling formalisms. Orthogonal to formalisms, Section 2.4 introduces analysis approaches for performance models. The formalization of performance requirements, analysis method selection, application of approaches, and life-cycle integration will be discussed as related work in Chapter 3.

2.1 Measurement-based Performance Evaluation

Measurement-based performance evaluation techniques obtain values for performance measures of interest—e.g., response times, throughput, and resource utilization—by collecting, processing, and analyzing run-time data from a system under execution. Typical measurement-based applications for software system development and operation are debugging, profiling, logging, and

monitoring. Debugging and profiling usually happen at development time in development environments where a high degree of perturbation is acceptable. For example, profiler tools typically impose a significant performance overhead but provide very detailed information on the run-time behavior. Logging and monitoring are used during operation in the production environment, which limits the accepted perturbation to a level that does not violate the system's Service-level Agreements (SLAs). Administrators usually accept a certain level of perturbation to gain valuable information about the run-time behavior and system health [Hoo14]. Infrastructures for performance measurement, ranging from proof-of-concept research approaches to established commercial tools for production use, have been proposed throughout the past decades. Classic approaches, focusing on C/C++, include UNIX's *prof* [Bel79] and *gprof* [GKM82] tools, for analyzing program profiles, as well as ATOM [SE94] and Pin [Luk+05] for static and dynamic program instrumentation [Hoo14]. The Application Response Management (ARM) [Joh98] defines an Application Programming Interface (API) for monitoring performance information about business transactions, e.g., response times. Under the term Application Performance Management (APM) [Men02a], various tools for continuously monitoring heterogeneous enterprise application system (EAS) landscapes are available. Gartner regularly analyzes the market of APM tools and publishes a report including the so-called "Magic Quadrant for Application Performance Monitoring" [KC14; Cam15]. The Gartner report includes a set of commercial tools, e.g., by companies like CA Technologies, IBM, HP/HPE, Compuware (DynaTrace), New Relic, and AppDynamics. These tools provide a rich set of features and support monitoring of system infrastructures comprising different technologies. Kieker [HWH12] is an open-source monitoring and analysis framework which already includes selected APM features, e.g., concerning discovery and visualization of distributed architectures. Compared to commercial tools, Kieker's strength is its flexibility and extensibility [Hoo14] which makes it a judicious choice for the research software developed during the course of this thesis.

For monitoring tools, system overhead plays a critical role. System monitoring approaches, like Magpie [BIN03; Bar+04] or X-Trace [Fon+07], are minimally invasive and target only network and operating system parameters. They come with the advantage of low system overhead but are not able to provide a view of internal application behavior. To reduce the performance impact of application monitoring, also sampling can be used to avoid to collect and store detailed execution traces [ES14; Sig+10].

For APM tools, the monitoring overhead complies with the number of transactions times the number of measurement points divided by implementation

overhead in seconds — depending on the actual monitoring tool implementation. Measurement-based approaches are usually not assessed or compared according to accuracy, as there is no ground truth like for model-based analysis.

2.2 Model-based Performance Evaluation

Model-based performance evaluation techniques obtain values for performance measures of interest by analyzing performance models. Model-based predictions allow for an exploration of alternative deployments, architectures, and configurations without the need to test them in a live system. Hence, model-based predictions can be considered to be faster and more convenient compared to measurements. However, model-based predictions are only valuable when providing a certain level of accuracy. Moreover, the creation of performance models can be associated with significant efforts outweighing their benefits.

While the adoption of model-based performance evaluation in the industry is still in its infancy, research outlines a plethora of applications [Bru+15]. Performance models enable the integration of performance in development and operation by means of an engineering discipline that proactively evaluates future system states. "Model-based performance evaluation is, for instance, useful for (i) exchanging and comparing performance metrics during the whole system life-cycle, (ii) optimizing system design and deployment for a given production environment, and (iii) early performance estimation during system development. " [Bru+15] Due to convenience and analysis run-time (neglecting efforts for model creation), performance models suit for design space exploration [Koz11; Mar+10]. Moreover, it has been discussed to apply models at run-time for automated performance and resource management [BHK14; Hub+15].

Performance models can be created automatically either based on monitoring information from running applications [BK17; BVK13; BKK09; BHK11; Wal+17c] or from a variety of different design specifications [PW03; PS02; PW02; CM00]. Exploitable design specifications summarized in [Bru+15] contain the Unified Modeling Language (UML) including sequence-, activity-, and collaboration diagrams [PW03], execution graphs [PS02], Use Case Maps (UCMs) [PS02; PW02], Specification and Description Language (SDL), and Message Sequence Chart (MSC) which depict design documents popular in the field of telecommunication [Ker01], or object-oriented specifications of systems like class, interaction, or state transition diagrams based on object-modeling techniques [CM00].

Independent of performance evaluation approach, system investigation usually assumes that the system is subject to incoming requests. While measurements may monitor production load without explicitly specifying it, model-based analysis has to specify load patterns explicitly. Workloads can be classified as open-, closed-, or hybrid [SWH06a]. For closed workloads, the inter-arrival time of requests directly correlates with their departure time as new requests are sent only after a previous one completes. This correlation is inexistent for open workloads where a client continuously sends requests irrespectively of how long each request takes to complete. The impact of workload models on performance has been discussed emphasizing the importance of choosing the appropriate workload model [SWH06b; SWH06a]. For example, choosing a workload generator based on a closed model can significantly underestimate response times and underestimate the benefits of scheduling when evaluating capacity for an open system [SWH06a].

Performance modeling formalisms support various modeling means which may lead to a discrepancy of modeling precision when using different formalisms. Based on the modeled ground truth, a variation of model solving enables to trade off accuracy versus the time required to derive results. As model-based performance evaluation depicts a core topic of this thesis, we provide more details in the following two sections. Although related work often considers modeling formalisms and solution strategies in combination (e.g., [Koz10; Bro14; Bal+04], we strongly recommend to consider model description (cf. Section 2.3) and model solving. (cf. Section 2.4) separately.

2.3 Performance Models

There are a variety of concepts to model the performance of software systems. To predict performance metrics, various types of models may be used to represent systems in a manner that suits its purpose. Different notations for performance modeling have been proposed focusing on specific applications and types of analysis.

The presence of various system characteristics and analysis needs explains the variety of modeling notations. In particular, the universal model characteristics (i.e., mapping, reduction, and pragmatism) postulated in [Sta73] explain the variety of formalisms but do not suit for classification of performance modeling formalisms in the context of this thesis. [Kou+17] distinguish three main model characteristics refining the purpose of a model:

- *Descriptive* models describe the actual internal state of a system at a given point in time.

- *Prescriptive* models represent planned states of a system in the future. The system needs to adapt to reach such a state.
- *Predictive* models enable the prediction of the impact of changes in the system or its environment on the system behavior. Predictive models are required for a system to become *proactive*, i.e., to react to changes in the environment in advance before violating high-level goals.

These characteristics suit to categorize performance modeling formalisms. In the following, we distinguish stochastic-, black-box-, and architectural performance models. Stochastic and black-box performance models are purely predictive. Architectural performance models are predictive and descriptive, and can also be enhanced to be prescriptive. In this thesis, we apply architectural performance models to reflect software systems within our declarative approach and transform them into stochastic performance models to solve them. Furthermore, we learn black-box models for time-to-result estimation of model-based performance evaluation.

2.3.1 Black-box Performance Models

Black-box approaches for performance prediction apply machine-learning algorithms to infer mathematical models of the performance behavior by interpolation and extrapolation of measurements. However, the behavior behind the measurements is not captured. Black-box performance models abstract the system at a very high level without taking its structure into account. "Predictions for service composition changes and system reconfiguration scenarios cannot be obtained" [Bro14] as black-box models do not capture the system architecture.

Compared to black-box machine-learning approaches, a key advantage of white-box models is that they provide insights and causality to solve performance problems. They allow for compositionality and explicitly capture behavior for several system variations within the modeling formalism, for example, the effects of changing the number of CPU cores, CPU frequency, or workload behavior. Mostly, black-box and white-box models are applied separately. As an exception, [Did+15] combine analytical modeling and machine-learning to get more robust performance predictions.

Existing black-box prediction approaches either derive the training data during operation without controlling the independent parameters or systematically vary the independent parameters to representatively capture a large input space (e.g., [Wes+12; Thü+14]). In this direction, studies investigate the trade-off between the amount of training data and the accuracy of the inferred

function. [Guo+13] propose a variability-aware approach to performance prediction via statistical learning. [Zha+15] provide theoretical guarantees for prediction accuracy which enable to minimize the number of required samples based on a user-specified confidence level. Furthermore, black-box approaches differ in their assumptions about the form of the functional dependency, for example, whether there is a linear dependency or not.

Summarizing, the presented black-box approaches apply supervised learning to obtain a function from inputs to outputs, so that the function can return outputs for previously unseen inputs. For additional details on statistical learning, please refer to fundamental literature on machine-learning such as [HTF09].

2.3.2 Architectural Performance Models

Compared to black-box machine-learning approaches and classical stochastic performance models, architectural performance models also preserve architectural information alongside the performance information.

Architectural performance models have been applied for various purposes, in different domains, and at different development stages in online and offline scenarios. In the past, a number of meta-models for building architecture-level performance models of software systems have been proposed. Prominent meta-models include Descartes Modeling Language (DML) [Kou+16], Palladio Component Model (PCM) [BKR09], CACTOS [16], SAMM [08], UML Profile for Schedulability, Performance and Time (UML-SPT) profile [Obj05] and its successor the UML Marte [MAD09], ACME [SG98], CSM (Petriu and Woodside, 2007), KLAPER [GMS07], and ROBOCOP [Bon+04].

In the context of this thesis, we mainly apply DML and also refer to PCM for automated model extraction. Therefore, we quickly introduce both formalisms. The PCM mainly targets design time analysis, while DML also targets run-time analysis. Therefore, both differ in their corresponding performance evaluation toolchain. However, large parts of both meta-models are conceptually similar. Differences include: PCM lacks a component behavior description at different granularities. DML provides advanced concepts to describe parametric dependencies.

Descartes Modeling Language DML provides descriptive, architecture-level performance models for performance and resource management. In contrast to other architecture-level performance models, it supports empirical as well as explicit descriptions of model variables and parameter dependencies. DML

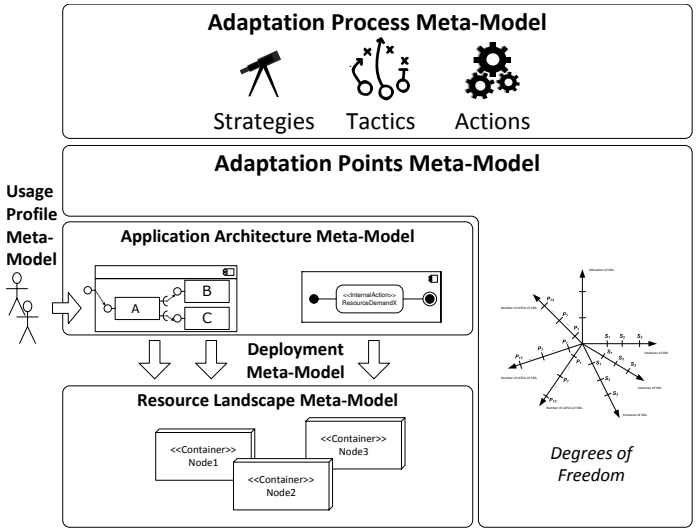


Figure 2.1: DML meta-model overview (from [Spi17]).

provides an architectural performance modeling language designed for run-time performance and resource management. This architectural performance modeling language has been applied for various purposes, in different domains, and at different development stages in online and offline scenarios. The DML is an architectural performance modeling language for quality-of-service and resource management of dynamic IT systems and infrastructures. DML has a modular structure depicted in Figure 2.1. It is provided as a set of meta-models for modeling the usage profile, resource landscape, and application architecture. The application architecture description divides into meta-models for repository and deployment. The component specification defines the relation between service input parameters and the performance effect of a service call. The deployment specification maps component instances to available resources, e.g., servers. The usage profile defines a set of usage scenarios that describe the investigated user behaviors. The explicit specification of usage profiles allows evaluating whether a system adheres to SLAs for a set of given user behaviors.

A DML instance (see example in Figure 2.2) contains a *repository* of *basic* and *composite components*. Each component has *interface providing roles* and *interface requiring roles*. Roles are associated with an *interface* that declares a set of *operations*. Each operation of an interface providing role corresponds to a service of a component that can be called by other components. The interface requiring roles specify the services that a component depends on. A basic component must specify a service behavior for each provided service (i.e., for

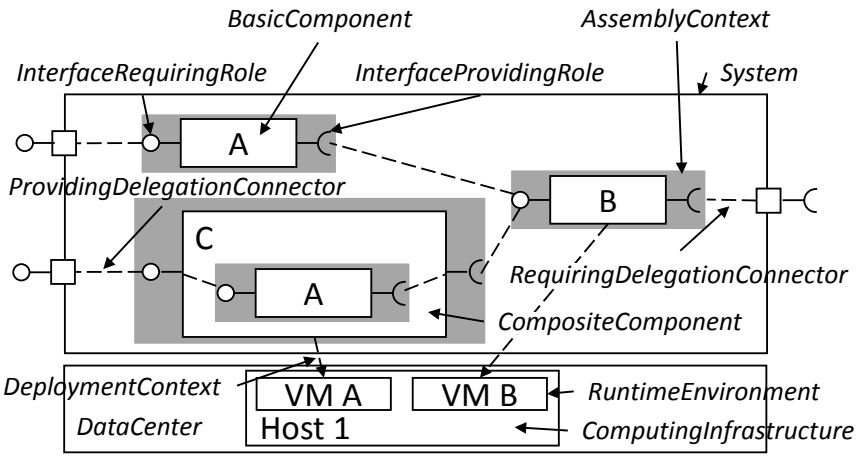


Figure 2.2: DML notation example (from [Wal+18]).

each interface providing role and operation). The service behavior specifies the performance-relevant control flow of the component, such as resource accesses, external calls to other services, loops, forks. Composite components bundle a set of components which can be deployed together.

Components are composed to a *system* using *assembly contexts*, *assembly connectors*, and *delegation connectors*. Each assembly context represents a component instance within a system or a composite component. A component may be instantiated multiple times in a system at different positions in the control flow (e.g., component `A` in Figure 2.2). Assembly connectors represent the control flow between components. Delegation connectors can be used to expose providing or requiring roles to enclosing composite component or system. The *resource landscape* describes the physical and logical resources in a *data center*. The primary entities are *containers* which can be a *computing infrastructure* (i.e., physical server) or a *run-time environment* (e.g., a VM or a middleware service). Each container contains a description of its resources, such as CPU, hard disks, or network links. *Deployment contexts* map an assembly context to a container. A *usage profile* contains a set of *usage scenarios* describing the incoming workload to a system (open/close workload). A usage scenario defines the sequence of *system user calls* to interfaces provided by the system.

The various submodels of DML reference elements from each other. They are interconnected in many parts. For instance, the specification of deployment creates pointers to the resource landscape and a component description. Linking between elements is common for architectural languages as it allows to

reuse elements in several contexts. For example, a single component may be deployed on multiple servers, and a single server may host several components.

Degrees of Freedom and Adaptations The Declarative Performance Engineering (DPE) approach presented in this thesis aims at automatically providing and solving an as large as possible set of performance analyses. Models prescriptively capturing a set of possible system states enable a variety of additional performance analyses. In contrast to the majority of stochastic models, architectural models can be extended to be prescriptive. In order to be prescriptive, a space of alternative configurations has to be modeled. The transcription of alternative configurations can be explicit, by modeling Degree of Freedom (DoF) or implicitly by defining adaptation operations. DoFs affecting performance can be defined based on software, deployment, meta-model particularities or explicitly defined to a system [Koz11]. Software-related degrees modify the application layer of the system. Deployment-related adaptations change the mapping of the application to hardware, the configuration of hardware, or further options in the software stack. Adaptation operations can be specified at the meta-model level or added to a specific system model [Hub+12].

2.3.3 Stochastic Performance Models

Stochastic performance models depict a popular approach to capture temporal system behavior. A number of different performance modeling formalisms has been proposed. We provide an overview of popular modeling formalisms and briefly introduce foundations to the ones discussed and applied throughout this thesis. In the course of this thesis, we refer to them when evaluating our method for decision support. Moreover, we apply Queueing Petri Nets (QPNs) to solve DML models. We start with an introduction to Queueing Networks (QNs). Then, we explain Layered Queueing Networks (LQNs) combining Stochastic Process Algebras (SPAs) and QNs. We do not go into detail of SPAs since we do not refer to them later. Then, we briefly describe formalisms based on Petri Nets (PNs). Finally, we introduce QPNs as a merge of QNs and Coloured Generalized Stochastic Petri Nets (CGSPNs).

Queueing Networks (QNs) QNs provide a powerful method for modeling contention for processing resources, i.e., hardware contention and scheduling strategies. QNs represent a system by a set of interconnected queues. A queue consists of a waiting line and a server. Incoming requests have to stay at the waiting line if another request occupies the queue. Otherwise, the queue server

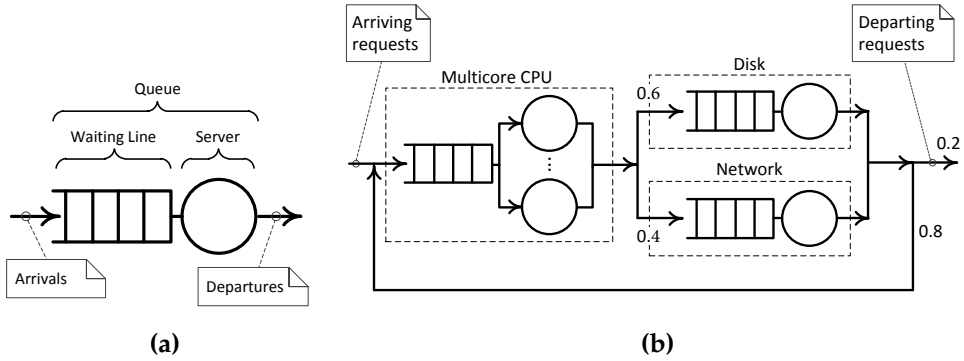


Figure 2.3: Graphical notation of a) queue and (b) queueing network.

can process arriving requests immediately. A request leaves the queue after the server has completely processed it. Figure 2.3(a) shows a single server queue. Multi-server queues process incoming requests at a server that is currently free. Incoming requests are only put into the waiting line if all servers are in use. Queues can be formalized using *Kendall's notation* [Ken53] which we omit as we do not refer to this formalization later.

Hereafter, we introduce common terms for performance engineering originating from queueing theory that can also be applied to other performance models. Incoming requests may arrive at arbitrary points in time. The interval between successive request arrivals is called *inter-arrival time*. Its reciprocal is the *arrival rate* which is the number of requests per time unit. The *service time* describes the time a server is busy to process a request. The time a request spends waiting in the waiting line is referred to as *waiting time*. The *response time* of a request is the sum of its waiting time and service time. It is the total amount of time the request spends at the queue. On finishing a request and a non-empty waiting line, another request from the waiting line is scheduled according to the queues *scheduling strategy*. The scheduling strategy determines the order in which requests are processed. Typical scheduling strategies when modeling IT systems include First-Come-First-Serve (FCFS), Processor-Sharing (PS), and Infinite-Server (IS). FCFS processes requests in incoming order which reflects the behavior of I/O devices. PS schedules all requests according to an idealized round-robin scheduling with an infinitesimal time slice commonly used to model CPUs behavior. IS schedules all requests immediately which is typically applied to model constant delays such average network delays.

To illustrate, Figure 2.3b shows an example of a QN with three queues. The multi-server queue models a multi-core CPU, the single server queues represent

a disk and a network, respectively. All incoming requests arrive at the CPU. Afterward, they are routed either to the disk or the network according to labeled probabilities. Coming from the CPU, a request is routed to the disk queue with a probability of 60 percent. With a probability of 40 percent, a request is routed to the queue representing the network. After being processed at the disk or the network queue, the request either leaves the QN with a probability of 20 percent or is immediately routed back to the CPU queue with the inverse probability of 80 percent. As illustrated by example, requests may visit a queue multiple times while circulating through a QN. The *resource demand* of a request at a queue describes the cumulated amount of service times for all visits of the queue.

QNs provide a powerful method to model contention due to hardware and scheduling strategies. However, QNs do not provide means to represent blocking behavior, synchronization of processes, simultaneous resource possession, or asynchronous processing of software processes [MAD94; KB03]. For additional details on QNs, we refer to [Jai91; Bol+06; Tri02].

Layered Queueing Networks (LQNs) LQNs expand QNs to picture simultaneous resource possession. While QN may only consider contention for a single resource, LQNs can model contention also for multiple resources. At this, LQNs apply the concept of layered performance models grouping the servers in layer sub-models [Fra+09]. When the server calls another server and waits for the return, it is an example of layered queueing. Such nestings may recur in any depth.

LQNs consist of a set of *processors* which themselves contain a set of *tasks*. Processors consume time and represent physical resources. They may only receive service requests from comprised tasks. Tasks model interaction and may represent different kinds of objects, e.g., load generating clients, software entities, non-processor devices such as disks, critical sections, and resources such as buffers [Fra+09]. Tasks can be modeled in a layered hierarchy. *Entries* model the services provided by a task. They may specify a resource demand or refer to *activities* structured in a control flow graph. Such graphs support modeling of sequences, branches, loops, and forks parameterized by branching probabilities and loop iteration counts. For additional details on LQNs, we refer to [Fra+09].

Petri Net based Formalisms PNs —introduced by [Pet62]— suit for the analysis of concurrent systems. In contrast to QNs, PNs are designed to model synchronization behavior of concurrent requests via shared places. The PN formalism

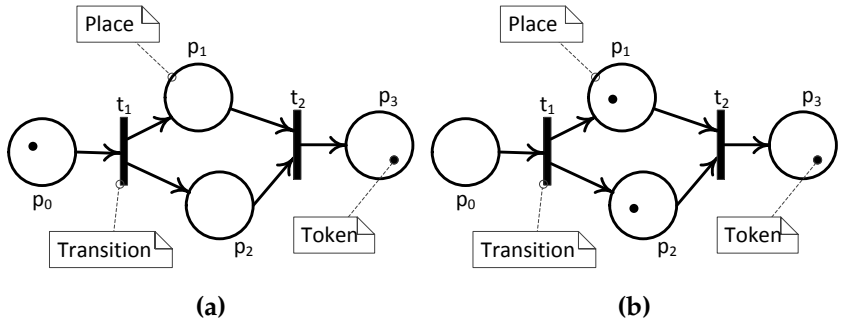


Figure 2.4: An ordinary Petri net (a) before firing and (b) after firing transition t_1 (from [Bro14]).

includes no timing behavior but serves as a basis for various extensions. An ordinary PN is a bipartite directed graph with two node types: *places* and *transitions*. Places represent system states by containing a number of *tokens*. Tokens may represent resource availability, jobs to perform, the flow of control or synchronization conditions. An initial marking of a PN describes for each place the number of *tokens* the place contains. A transition is enabled to *fire*, if each input place, contains the required number of tokens. According to a predefined mapping, the *firing* of a transition destroys the tokens in its input places and creates tokens in its output places. An input place has a directed connection *from* the place *to* the transition. Output places have a directed connection *from* the transition *to* the place. Figure 2.4 illustrates an ordinary PN with four places and two transitions, before and after the firing of a transition. The places are illustrated as circles. Transitions are visualized as bars. The directed edges connecting a place and a transition are depicted as arcs.

Stochastic Petri Nets (SPNs) [BK02] add an exponentially distributed firing delay to each transition. The delay specifies the waiting time before a transition fires after being enabled. Generalized Stochastic Petri Nets (GSPNs) provide immediate transitions and timed transitions. Immediate transitions fire in zero time. If at a point in time more than one immediate transition is enabled, the transition to fire next is chosen based on configured firing weights. Timed transitions fire after an exponentially distributed delay as in SPNs. Coloured Petri Nets (CPNs) [Jen81] introduce a convenient way to distinguish between different resources in one net. Transition firing rules can be defined according to token colors represent different resources. To this end, a color function assigns different firing modes to one transition. The greater modeling convenience comes along with no loss of PN properties as CPNs are a backward-compatible

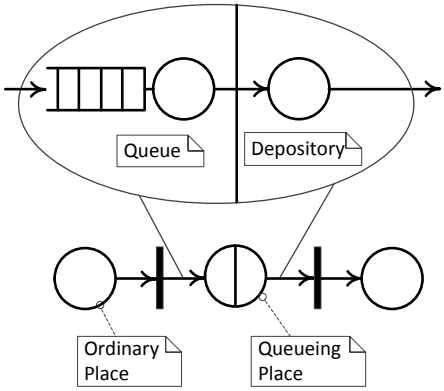


Figure 2.5: Graphical notation for a queueing place within a queueing Petri net (from [Kou05]).

extension to the original PNs. CGSPNs combine GSPNs and CPNs resulting in a modeling formalism incorporating the strengths of both formalisms. For formal definitions of the previously mentioned extensions to PNs, we refer to [BK02].

Queueing Petri Nets (QPNs) While modeling formalisms based on PNs provide powerful means to model blocking and synchronization behavior; it is challenging to model waiting lines for processing resources with respect to common scheduling strategies such as FCFS. On the other hand, QNs cannot accurately reflect software aspects. [Bau93] introduces QPNs as a merger of QNs and CGSPNs. QPNs combine the advantages of QNs and CGSPNs, and thus eliminate their respective disadvantages. The combination enables to enrich places of a CGSPN by queues as described for QNs. As a result, resources that are processed using specific scheduling strategies as well as simultaneous resource possession, synchronization and blocking can be modeled.

Figure 2.5 depicts a QPN with two ordinary places and a *queueing place*. The *queueing place* is drawn as a circle with a horizontal line. In contrast to an ordinary place, a queueing place consists of two parts, a queue, and a depository. The depository holds tokens that have completed their service at the queue. Tokens entering a queueing place are passed the queue's waiting line, tokens leaving a queueing place are taken from its depository. For additional details as well as for a formalization of QPNs see [Bau93].

2.4 Model Solution Techniques

Independent of performance modeling formalism, generally a spectrum of solution strategies to derive performance indices is available. Alternative techniques to analyze performance models include analytical techniques providing exact results, numerical approximation techniques, simulation, or operational laws. Such solution strategies differ in restrictions on applicability, functional (e.g., supported performance metrics, aggregated and non-aggregated) and non-functional (e.g., accuracy, scalability, and time-to-result) properties.

There are several popular tools to solve performance models. Techniques to solve QNs are supported by, e.g., the SPEED tool [SW97], or the Java Modelling Tools (JMT) [BCS09]. Both tools support both simulative solutions as well as analytical solution techniques such as Mean Value Analysis (MVA). ORIS [Buc+10] provides simulation and analytical solvers for SPN and GSPN. Fluid-based approximation for QNs in random environments is provided, e.g., by LINE solver [PC13; PC17]. Popular tools for QPN analysis are HiQPN and SimQPN. The HiQPN provides an analytical solution of QPNs based on Markov chain analysis and thus suffers from the state space explosion problem. SimQPN is a tool supporting the analysis of QPNs by optimized discrete-event simulation which is not subject to state space explosion [KB06; SKM12]. LQNs can be solved using the lqsim simulator [Fra+11] as well as an analytical solver based on approximate MVA named LQNS [Fra+11].

2.4.1 Analytical Approaches

Analytical techniques provide exact analysis results for upper bounds and steady-state metrics. However, their applicability is often limited to subclasses of models constraining model structure, workload definition, scheduling strategies, or applied stochastic distributions of model parameters. In general, efficient mathematical solutions often rely on strong assumptions [Jai91]. Common assumptions include exponentially distributed inter-arrival times and service time distributions. QNs with weaker assumptions are usually mathematically intractable. Thus, they need to be simulated to derive performance predictions. In the following, we provide a brief overview and refer the interested reader to [Bol+06; Jai91] for additional details.

Operational laws

A large number of performance analyses can be solved by using basic relationships that do not require any assumptions about the distribution of service

times or inter-arrival times. They are commonly referred to as *operational laws*. Common laws include utilization law, forces flow law, Little's law, general response time law, interactive response time law, and bottleneck analysis [Jai91]. One operational law we refer to later is the *utilization law* [MAD94]. Given a network containing a workload description, we can calculate the utilization $U_{i,r}$ of a resource i due to the requests of workload class s using the relationship

$$U_{i,s} = X_{i,s} D_{i,s}$$

where $S_{i,s}$ is the average resource demand and $X_{i,s}$ is the throughput of requests of service s at resource i.

Product Form Solution

A *product-form* solution is an efficient way of determining metrics of a system with distinct components. It requires that the metric for the collection of components can be written as a product of the metric across the different components

$$\mathrm{P}(x_1, x_2, x_3, \ldots, x_n) = B \prod_{i=1}^{n} \mathrm{P}(x_i)$$

where B is a constant. Solutions of this form are of interest as they are computationally inexpensive to evaluate for large values of n. The *product-form* characteristic requires that the network has to satisfy the conditions of job flow balance, one-step behavior, and device homogeneity. The BCMP theorem (named after Baskett, Chandy, Muntz, Palacios) defines an expanded set of conditions for which networks with multiple workload classes can be efficiently solved using a product-form solution [MAD94; Bol+06; Tri02].

Mean Value Analysis

MVA is a recursive algorithm to compute expected queue lengths, waiting time at queues and throughput in equilibrium for closed workloads. The algorithm starts with an empty network and increments the number of customers in each iteration until the required number of customers has been reached. For a large number of jobs, particularly if the performance for smaller subsets is not required, it would be preferable to avoid this recursion. Hence, several approximate analysis techniques have been developed. [SSB93] avoids the recursion estimating the queue lengths for a subset of jobs and computing the response times and throughputs. The derived values can be used to recompute the queue lengths. [SSB93] assume that as the number of jobs increases, the queue length

at each device increases proportionately. Note that MVA is applicable only if the network is a product form network.

Markovian Solutions and Fluid Approximations

Network models can be understood as Markov chains. Markov chains satisfy the Markov property also referred to as memorylessness. A stochastic process is memoryless if the conditional probability distribution of future states of the process depends only upon the present state and not on the sequence of preceding events. Classical Markovian analysis results, after preceding matrix operations, in solving a system of equations. The solution of equation systems may suffer from the state space explosion problem.

Fluid or mean-field methods are approximate analytical techniques which have proven effective in tackling the state-space explosion problem. These methods are particularly suitable for large populations of small local state spaces since they require the analysis of a problem which is insensitive to the population sizes but depends only on the size of the local state spaces.

2.4.2 Simulation

Simulation can be classified to be either discrete or continuous. The discrete-event simulation requires to describe the system behavior over time as a finite sequence of events. In contrast, continuous simulation can be applied if the system cannot be split into discrete events.

For the analysis of software systems, discrete-event simulation is the method of choice [Jai91]. Compared to analytical solutions, the concept of simulation has no limitations concerning required input models. However, limitations may stem from restrictions of particular simulation engine implementations, such as restrictions to open workloads for an efficient parallelization [WSK15a]. Simulation can be used to investigate steady-state behavior —like supported by the previously introduced performance evaluation techniques— as well as transient behavior analysis not supported by analytical approaches.

Steady-state Analysis

Steady-state analysis answers the question of how systems behave in the long run. From a mathematical viewpoint, we simulate a discrete-time stochastic process X_i , $i = 1, 2, \ldots$ and are interested in the steady-state behavior. For example, we can determine steady-state mean $\lim_{i \to \infty} E(X_i)$. One should keep in

mind that steady-state analysis is only reasonable if a steady-state exists. For instance, the average of a rising population would not be useful to determine.

A challenge for steady-state simulation is to reasonably parametrize simulation length to be sufficient to provide the desired accuracy. Commonly, discrete event simulation applies two termination criteria in parallel: maximum virtual time and sufficient accuracy. Moreover, a challenge of steady-state analysis is initial bias detection. The goal is to find a minimal truncation point of sufficient size to discard initialization bias. Approaches to detecting an initialization bias include graphical, heuristics, statistical, test-based, and hybrid methods [HRD10]. Graphical methods involve the visual inspection of time-series of the output data. Heuristic approaches apply simple rules with few underlying assumptions. Statistical methods build upon the principles of statistics for determining the warm-up period. Initialization bias tests identify whether there is initialization bias in the data. Strictly speaking, these are not methods for identifying the warm-up period, but they can be combined with warm-up methods for this purpose.

Transient Behavior Analysis

In contrast to analytical solutions, a simulation may also be used to investigate transient behavior when dynamically changing system states. Simulation enables the analysis of varying workloads as well as an incorporation of self-adaptive behavior into a simulation.

Previous performance evaluation approaches assume a constant load pattern over time, simulation enables to investigate the temporal progression of workload intensity. An example of simulation of dynamic input is a trace-driven simulation. Trace-driven simulation derives the model input from a sequence of observations made on a real system [MR09; Zhu+05]. Replays of production load can be an alternative to explicit modeling of job arrivals. Simulation can also be used to workload models containing seasonal, trend, burst, and noise components [Kis+15] or Markovian arrival processes where arrivals depend on previous arrivals [CZS08].

Systems reactions to their dynamic environment that can be simulated include, for example, restructuring of components and connectors, exchanging components or services, or altering hardware infrastructure [BLB13]. Applications can be the investigation of software or hardware failures on the system behavior as well as recovery times for self-healing systems. Also, network attacks and self-aware protection mechanisms can be subject to dynamics [Iff+18b] demanding transient analysis.

2.4.3 Multi-step Solution Strategies

Model transformations, simulation or analytical solutions, and bridging code can be put together to form a multi-step solution strategy. Architectural performance models can be either simulated directly or automatically translated into analytical models and, then be processed by respective solvers. Each transformation into a stochastic performance modeling formalism enables to apply its solution methods. However, the support is in a limited way if the transformation cannot transmit all concepts provided by the source model.

Various transformations from architectural performance modeling languages to purely predictive stochastic performance modeling languages have been developed [DM06] to derive performance metrics for software architectures. Performance interchange formats, e.g., S-PMIF [Smi+05], KLAPER [GMS07], and Core Scenario Model (CSM) [PW07], aim to bridge the gap between analysis and architecture-level models. Model transformations are a core component of model-driven performance prediction approaches in SPE [DM06; CDI11]. Particularly, this concerns model-to-model transformations from design-level models—including performance-relevant completions [WPS02]—into purely predictive performance models. Popular transformations convert from UML Profile for Schedulability, Performance, and Time (SPT) and UCMs to LQN [PS02; PW02], from PCM to LQN [KR08], from PCM to QPN [MKK11], and from UCMs to PCM [Vog+13]. Less widespread, there are also transformations between stochastic formalisms, e.g., QPN to LQN [Mül+16].

While architectural models allow for different analysis approaches, this work focuses on DML's commonly used solution approach [Bro+15; Hub+15]. The solution of DML models applies a multi-step approach. It includes a model-to-model transformation followed by simulation and result filtering. The Descartes Query Language (DQL) framework [GBK14] triggers a transformation to QPNs [MKK11], followed by a simulation using the batch-means method implemented by the SimQPN simulation engine [Kou+16], and finalized by filtering of simulation results.

Chapter 3

Related Work

Woodside et al. discuss in [WFP07] the state-of-the-art and future trends within Software Performance Engineering (SPE). They emphasize the huge gap between measurement-based and model-based analysis and proclaim a need for them to converge, in order to cover the entire software development cycle. Even though the work has been published more than a decade ago, it still captures the main issues preventing the automation of SPE life-cycle and is, therefore, a strong motivation for this thesis. In the following, we present four areas which are closely related to the Declarative Performance Engineering (DPE) approach presented in this thesis. Section 3.1 surveys the state-of-the-art for the specification and interpretation of performance analysis and performance requirements. In Section 3.2, we review approaches for the assessment and selection of performance evaluation approaches and tools. As a prerequisite for model-based analysis, Section 3.3 discusses performance model creation, and automated model building approaches. Section 3.4 is about the software life-cycle integration of SPE.

3.1 Modeling and Interpretation of Performance Related Concerns

This section presents work that is related to Chapter 4 where we present a language to specify performance concerns and a corresponding framework to automatically answer them. We subdivide related work into modeling (Section 3.1.1) and deduction (Section 3.1.2) of performance concerns.

3.1.1 Modeling of Performance Related Concerns

Performance indices of software systems depict a basic information need. To specify such needs, several modeling languages (respectively meta-models) have been proposed. They enable to express relationships between measures of interest, referencing elements from other modeling languages describing the

actual software. Examples include the Software Measurement Modeling Language (SMML) ([Mor+09]) and the Structured Metrics Meta-Model (SMM) by the Object Management Group (OMG) [Obj12]. These approaches usually focus on static software properties (e.g., architecture or code metrics) and perform only structural queries on the referenced models instead of analyzing the models quantitatively, e.g., by model-based performance prediction. The *Unified Modeling Language (UML) Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QFTP)* [Obj08] enables to express the quality of service requirements and properties, including performance. However, as for the previously mentioned approaches, no connection to model-based or measurement-based evaluation infrastructures is provided. Continuous performance evaluation frameworks, like [Ben18], often include own Domain Specific Languages (DSLs) to specify metrics to be measured.

Particularly in domains where performance is business-critical, service providers and consumers often negotiate Service-level Agreements (SLAs). SLAs constitute contractual specifications of Quality of Service (QoS) requirements and penalties that are due in case these QoS requirements are violated. The QoS requirements as part of the SLAs are also denoted as Service-level Objectives (SLOs). An example of a performance SLO is that *for a defined service, 95 % of all response times observed within any 30-minute time-window must not exceed 500 milliseconds.* For the specification of SLAs, (machine-processable) languages, which can be considered modeling languages in the context of model-driven software engineering (MDSE), and corresponding frameworks have been developed in academic and industrial contexts, e.g., Web Service Level Agreement (WSLA) [IBM03; KL03], Web Service Offerings Language (WSOL) [TPP02; Tos04], WS-Agreement [Ope11], a language for specifying SLAs (SLA$\star$) [KTK10], and a language for specifying SLAs (SLAng) [Ske07; SRE10]). Most of these languages focus on SLAs in web service scenarios. WSLA includes a mapping of SLAs to a corresponding monitoring service that assesses the conformity of run-time performance properties based on the specified SLAs.

There are also approaches extending the UML to specify QoS properties of the software at modeling phase such as QML [FK98; Zsc10]. For cloud environments where landscape does not consist of dependable server and network resources but fluctuates, [KTK10], as well as [KL12], propose language constructs like fuzziness to cope with uncertainty.

Most works on SLA formats focus on defining a machine-readable format to be processed automatically (e.g., [KTK10; KL03; TA06; SRE10]). However, XML specifications increase the manual effort required for their maintenance and

calls for a format that is both machine-readable and human-readable [Par01; Vad+15]. Therefore, [Lud+15] propose a DSL for describing SLAs which is also human readable.

Based on SLAs, there are manifold performance engineering activities. Examples extending plain SLA evaluation include multiple system settings analysis, system configuration optimization and automated resource management [Hub+15], SLA generation and parametrization of thresholds [SHD16], or the determination of upper bounds for usage behavior [BV17]. For these activities, there is no unifying formalization yet, as they are all provided by separate tooling. Respective tools commonly focus on a single concern which does not require to formulate the concern explicitly. This also applies to many in-depth analyses such as bottlenecks analysis [And+13; Jai91]. [Sin+13] propose a framework to answer what-if concerns using a language called What-If Query Language (WIFQL). At this, WIFQL specifies the measures of interest and system variations.

Another popular interest is to optimize resource allocation. Optimization problems can be formalized using the Quality of Service Modelling Language (QML) [FK98]. Based on extensions, also optimization problem definitions may also include quality of service constraints [KNR11]. Required degrees of freedom to parametrize optimization problem definitions can be modeled per application [Hub+12] or derived from the meta-model [Koz11].

3.1.2 Interpretation of Performance Related Concerns

The result of performance concern modeling is a model describing an information desire which needs to be resolved by a performance analysis. Multiple software solutions exist. For example, MAMBA implements the model-based measurement of the SMM for the Kieker monitoring framework [Fre+12].

Despite the interpretation of performance concerns, such as deriving indices and SLA evaluation, could be performed using various measurement-based and model-based performance evaluation approaches, existing analysis frameworks mainly focus on a single evaluation software. Some meta-models even lack a connected performance evaluation technique. Exceptions, include [GBK14; Lud+15]. [GBK14] sketch analysis of performance indices using different model-based analysis approaches. To tackle the problem of heterogeneity in cloud systems, [Lud+15] propose to use a service that would connect to proprietary monitoring subsystems of the underlying cloud systems using REST interface. Despite changeability of evaluation being required for continuous performance management, the evaluation is either measurement *or* model-based.

The most common approach for the evaluation of SLA execution is to monitor the service during the production phase and check for any violation of SLA [KTK10; KL03; SRE10; TA06]. Although it would be possible, monitoring is usually performed by the provider, using its infrastructure, rather than by some third party [Kri+13].

Most commercial cloud providers provide autoscaling mechanisms, to support elasticity [HKR13]. They all allow for addition and removal of Virtual Machines (VMs), if the load is increasing or decreasing, respectively, maintaining the required service levels while minimizing their own cost of operations. One of the first, called EC2, was offered by Amazon([LaM08]), but soon other providers followed [But13]. However, monitoring solutions are proprietary and tied to a particular platform, as well as almost always under provider's control [Mot+14]. For example, the WSLA [KL03] framework also provides monitoring capabilities to accompany the SLA specification, but the prototype implementation is available only for proprietary IBM environments.

Reactive approaches dominate SLA evaluation. However, there are also proactive model-based approaches [Ard+14; BBM17; Hub+17; Kla+11] [Ard+14] survey modeling techniques for QoS in cloud computing. [Kla+11] integrate QoS prediction support in a large-scale SLA management framework and a service mash-up platform. [BBM17] apply analytical Queueing Network (QN) solvers to evaluate SLAs. [Hub+17] propose an approach that builds a model based on the run-time data, evaluates different possibilities for adaptation based on heuristics and then executes a resource-efficient configuration still satisfying goals.

In general, solving optimization problems results in a demanding process requiring analyses of several system states. Optimization problems can be efficiently solved using genetic algorithms [Koz11]. Processing of optimization problems can be more efficient when constrained by quality of service requirements [KNR11].

3.1.3 Discussion

In summary, initial description approaches and corresponding techniques for assurance or derivation of performance capabilities have been proposed. In general, the formal modeling of performance concerns happens by specifying a DSL or by the definition of a meta-model. Based on such specifications, analysis frameworks may provide end-to-end automation to derive an evaluation. However, current approaches suffer from several limitations. Existing works focus on a single performance concern and a single performance evaluation mechanism. None of the approaches provides an integrated evaluation

using model-based *and* measurement-based analysis. Moreover, none of the approaches allows for solving multiple concerns for SLAs. To the best of our knowledge, no SLA approach aims to cover both the design- and production-time evaluation of SLAs, using an SLA definition that is both human- and machine-readable, while being general and applicable to any kind of system. The disperse landscape of modeling and evaluation frameworks requires to apply separate tooling at each stage of the software development life-cycle.

3.2 Evaluation, Comparison, and Selection of Performance Evaluation Approaches

A plethora of performance evaluation techniques and tools have been proposed (cf. Chapter 2). They significantly differ in various dimensions. Therefore, a very relevant question is when to apply which methodology. The performance evaluation in each life-cycle stage has different artifacts, capabilities, and optimization goals. Moreover, software systems (and respective model-based representations) are built on different technologies restricting applicable evaluation approaches. Compared to a classical lookup system, decision support for performance engineering introduces additional challenges: First, the analysis costs highly depend on execution settings and performance concerns to derive. Second, the applicability of analysis approaches is often tailored to a specific analysis setting and limitations concerning other settings are not documented. Existing work on decision support within SPE can be grouped into decision support for model-based (Section 3.2.1) and measurement-based analysis (Section 3.2.2). Each group can be further divided into quantitative and qualitative assessment of capabilities.

3.2.1 Decision Support for Model-based Analysis

Performance predictions can be performed based on different model descriptions like regression models, queueing networks, control-theoretical models, and descriptive architectural performance models [Spi+17b].

Qualitative comparisons of model-based solution approaches have been performed, e.g., in [Bal+04; Ban+00; Bol+06; Bro+15; Koz10]. Existing work compares concrete toolchains including methods to derive models [Bal+04; Bro+15; Koz10] as well as comparisons of methodologies abstracting from concrete implementations [Ban+00; Bol+06]. Related surveys either focus on a single analysis type (like [Ban+00] focusing on discrete event simulation) or a single formalism (like [Bol+06] focusing on QN or [Bro+15] focusing on Palladio

Component Model (PCM)). [Bal+04] provide a categorization of different model-based solution approaches according to their supported model, application domain, and life cycle phase. [Koz10] surveys model-based analysis approaches for component-based systems focusing on model expressiveness, tool support, and maturity. [Goo+12] mix capabilities of formalisms and solution approaches and discuss the usability from an industrial perspective.

Many efficient model solution techniques can be applied only for constrained model subclasses. The applicability of solution approaches for QNs has been discussed for example by [Bol+06].

Quantitative analysis of solution approaches has been performed, e.g., in [Bro+15; KR08; Mül+16; PC17]. All have in common that they compare combinations of model-to-model transformations and solvers. [Bro+15] perform a quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. [Mül+16] quantify analysis time in relation to model parameters of a benchmark model. [RKT15] compare different solution approaches for network performance evaluation. Summarizing, none of the above works discusses automating the choice of analysis approaches, except for [Bol+06; Bro+15] providing automatable decision trees. While there are evolution and advancement in SPE, none of the works discusses extensibility.

3.2.2 Decision Support for Measurement-based Analysis

Qualitative comparisons of measurement-based approaches have been performed in [KC14; Oka+16; Wat17]. Market researchers of Gartner evaluate popular Application Performance Management (APM) tool vendors on a yearly basis [KC14; Cam15]. [KC14] define three dimensions of APM functionality: application topology discovery and visualization, application component deep dive, and user-defined transaction profiling. [Oka+16] compare application performance monitoring tools concerning derivable information, like call parameters or error stack traces. According to [Wat17], key aspects to consider include programming language support, cloud support, Software-as-a-Service (SaaS) versus on-premise, pricing, and ease-of-use. Quantitative analyses focus on system overhead [ES14; Sig+10; Wal14] and costs of ownership [App10; Wat17]. Cost of ownership types for APM include: (i) software licenses, (ii) hardware and system software required to operate, (iii) deployment and instrumentation costs, and (iv) ongoing maintenance costs [App10]. At this, pricing models of vendors impact the choice of monitoring infrastructures.

Monitoring may cause a significant impact on the overall performance of software systems. The MooBench [Wal14] micro-benchmark has been developed to measure and quantify the overhead of monitoring frameworks. Monitoring

of system metrics usually induces less performance overhead than application monitoring [Wal14].To reduce the performance impact of application monitoring, sampling can be used to avoid to collect and store detailed execution traces [ES14; Sig+10]. In contrast to model-based analysis, quantitative comparison for measurement-based analysis is less common due to proprietary tools and disclosure agreements.

3.2.3 Discussion

Performance evaluation techniques include various approaches like measurement, simulation, analytical solutions, model transformations to stochastic formalisms, algebraic solutions, or fluid approximations. Each evaluation approach has its specific analysis goals and capabilities. SPE tools and approaches have been compared with respect to different dimensions. However, related work suffers from several limitations. Most work is intended for an offline recommendation. Often the recommendation is based on the life-cycle stage instead of considering characteristics of system *and* concern which affect applicability and analysis costs. None of the works discusses how to estimate application specific analysis costs. The rare approaches that provide automated decision support do not discuss its evolution. There is a repetitive evolution of SPE approaches and tooling which makes most of the referenced work quickly outdated. While there are advancements in SPE, none of the works providing automated decision support considers evolution and extensibility for additional approaches or additional comparison attributes. Furthermore, related work strictly separates model-based *and* measurement-based analysis approaches. None of the existing works dares a comparison of both. The lack of comparison can be explained by a lack of a fully automated approach integrating both types of analyses so far.

3.3 Performance Model Creation

This section presents work related to Chapter 5 which proposes methods to create architecture-level performance models from execution traces automatically. Substantial parts of this section build upon a summary we published in [Wal+17b]. A number of meta-models for building performance models of software systems have been proposed (cf. Section 2.3). In the following, we abstract from concrete formalisms and discuss gathering of information required to build architectural performance models based on generic aspects. We

categorize the learning aspects into the determination of static system structure (Section 3.3.1) and dynamic behavioral aspects (Section 3.3.2).

3.3.1 Static Model Structure

The learning of model structure involves extracting information about the software components of the system as well as information about resource landscape in which the system is deployed. Software systems that are assembled from existing prepackaged components may be represented by the same components in a performance model [WW04].

Apart from building performance models of systems assembled from prepackaged components, previous research on architecture extraction targeted system reengineering scenarios. Examples of reverse engineering tools and approaches in this area are for example FOCUS [DM01], ROMANTIC [HSE10], Archimetrix [Det12] or SoMoX [Bec+10]. These approaches are either clustering-based, pattern-based, or a combination of both. Components are identified from a reengineering perspective which does not necessarily correspond to deployable structures of the current implementation.

Software systems may correspond to different architectural styles and complexities. Therefore, the choice of an appropriate model granularity is essential when generalizing model extraction. The complexity of a component decomposition can be reduced by merging sub-services. However, there is currently no automatable rule to derive a suitable granularity.

To allow automated learning of the system structure, the system and its components should provide information about their boundaries. If no predefined component boundaries are provided, component identification requires manual effort. In general, the following guidelines can be applied in case of component-based systems implemented using an object-oriented programming language: i) classes that implement component interfaces form components, ii) all classes that inherit from a base class belong to the same component, iii) component A uses component B $\rightarrow$ A is a composite component containing B.

Compared to component extraction, the automated identification of hardware and software resources in a system environment is already supported by industrial software tools. For instance, Hyperic [Hyp18] and Zenoss [Zen18] provide such functionalities. An open issue is to define standardized interfaces for resource extraction. Such would significantly improve the integration of different toolchains supporting interoperability. Deployment information can be extracted using event logs containing identifiers for software components and resources. The extraction process creates one deployment component for each pair of a software component and resource identifier.

3.3.2 Dynamic Behavior

This includes the parametrization of the model with resource demands, component behavior, and inter-component interactions.

Resource Demands

In performance models, component resource demands are key parameters required for quantitative analysis. A resource demand describes the amount of a hardware resource needed to process one unit of work (e.g., a user request, a system operation, or an internal action). The granularity of resource demands depends on the abstraction level of the control flow in a performance model. Resource demands may depend on the values of input parameters. This dependency can be either captured by specifying the stochastic distributions of resource demands or by explicitly modeling parametric dependencies.

The estimation of resource demands is challenging as it requires the integration of application performance monitoring solutions with resource usage monitors of the operating system in order to obtain resource demand values. Operating system monitors often provide only aggregate resource usage statistics on a per-process level. However, many applications (e.g., web and application servers) serve different types of requests with one or more processes.

Profiling tools [GKM82; Hal92], typically used during development to track down performance issues, provide information on call paths and execution times of individual functions. These profiling tools rely on either fine-grained code instrumentation or statistical sampling. However, these tools typically incur high measurement overheads, severely limiting their applicability in production environments, and leading to inaccurate or biased results. In order to avoid distorted measurements due to overheads, [KKR08; KKR09] propose a two-step approach. In the first step, dynamic program analysis is used to determine the number and types of bytecode instructions executed by a function. In a second step, the individual bytecode instructions are benchmarked to determine their computational overhead. However, this approach is not applicable during operation and fails to capture interactions between individual bytecode instructions. APM tools enable fine-grained monitoring of the control flow of an application, including timings of individual operations. These tools are optimized also to be applicable to production systems.

Modern operating systems provide facilities to track the consumed CPU time of individual threads. This information is, for example, also exposed by the Java Runtime Environment and can be used to measure the CPU resource consumption by individual requests as demonstrated for Java in [BVK13] and at the

operating system level in [Bar+04]. Monitoring resource demands per request requires application instrumentation to track which threads are involved in the processing of a request. Such tracking can be challenging in heterogeneous environments using different middleware systems, database systems, and application frameworks. The accuracy of such an approach heavily depends on the accuracy of the CPU time accounting by the operating system and the extent to which request processing can be captured through instrumentation.

Over the years, a number of approaches to estimate the resource demands using statistical methods have been proposed. These approaches are typically based on a combination of aggregate resource usage statistics (e.g., CPU utilization) and coarse-grained application statistics (e.g., end-to-end application response times or throughput). These approaches do not depend on fine-grained instrumentation of the application and are therefore widely applicable to different types of systems and applications incurring only insignificant overheads. Different approaches from queueing theory and statistical methods have been proposed, e.g., response time approximation least-squares regression, robust regression techniques, cluster-wise regression, Kalman Filter, adaptive filtering, Bayesian estimation, optimization techniques, Support Vector Machines, Independent Component Analysis, Maximum Likelihood Estimation, and Gibbs Sampling [Spi+15]. These approaches differ in their required input measurement data, their underlying modeling assumptions, their output metrics, their robustness to anomalies in the input data, and their computational overhead. A detailed comparison including quantitative evaluation is provided in [Spi+14].

Inter-component interactions and control flow

The extraction of information about the interactions between components differs for design time and run-time. At design time, models can be created based on designer expertise and design documents as proposed, e.g., by [SW01], [MG00], [PW02], and [CDI11]. Commencing at a runnable state, monitoring logs can be generated to derive an *effective architecture* including only the executed system elements [IWF07]. The automated extraction of control flow based on monitoring logs has the advantage to track only the behavior of the actual product as executed. Furthermore, run-time monitoring data enables to extract branching probabilities for different call paths [BVK13]. The approaches for extraction of information about inter-component interactions by [Hri+99], [BLL06], and [IWF07] use monitoring information based on probes injected at the beginning of each response and propagated through the system.

Inter-component interactions and control flow may depend on input parameters. Reflection of parametric dependencies within a performance model happens by explicit modeling of input parameters and description of resource demands as a function of input parameters. [Kro12] describes how to parametrize parametric dependencies based on static code and dynamic analysis. While there are techniques to parametrize parametric behavior automatically, there is no mechanism to detect such dependencies automatically other than testing all possibilities. Application specific model parametrization scripts have been applied in various case studies (e.g., [Hub+15; LB16]).

Workload Characterization

The system specification does not depend on the workload description which allows considering this aspect independently of the system. The following workload extraction methods rely on different assumptions and description models.

[Men+99] introduce a Customer Behavior Model Graph (CBMG) to describe customer groups with similar navigational patterns. They automatically derive these models from session logs by employing a K-means clustering algorithm. [Vög+18] present the WESSBAS language for describing workload models. In comparison to [Men+99], they do not presuppose a number of clusters and use X-means clustering to generate these models from session logs. [VHK15] propose an automatic transformation from the WESSBAS to PCM usage profile models. A drawback of the cluster-based approaches [Men+99; Vög+18] is that a small change in input data may lead to fundamental changes in the clustering. Further, the clusters do not necessarily correspond to what humans would model as customer classes.

While previous approaches assume a constant load pattern over time, [Kis+15] introduce a language to describe the temporal progression of workload intensity. They also propose an algorithm to extract these models from a time-series of arrival rates. The extraction method applies statistical methods to split the time-series into seasonal, trend, burst, and noise components. Job arrivals may depend on previous arrivals. To reflect such dependencies, [CZS08] fit workload traces into Markovian arrival processes automatically. Nevertheless, accurate modeling of workload behavior can be challenging. Therefore, [Zhu+05; MR09] propose to apply replays of production load as an alternative to explicit modeling of job arrivals.

3.3.3 Discussion

While monitoring tools are increasingly employed also in production systems, performance models are not widely used in the industry [Spi17; BDK15; Koz10]. A commonly cited reason for the low adoption is that the effort required to create performance models which often outweighs their benefits [Kou05; BVK13; BDK15].

Existing modeling studies often apply application specific model creation scripts connected to substantial manual efforts accounting for weeks up to months for experienced performance engineers [Hub+10; SW01; LB16]. To increase adoption of model-based analysis in industrial contexts requires an automated extraction based on measurement data that is independent of the investigated application. Several researchers, (e.g., [WFP07; BDK15]) highlight the need to automatically learn models of a system and its environment.

There are open challenges not addressed by state-of-the-art model extraction. While models could be derived based on various inputs, existing solutions lack flexibility for different data inputs. While there are several performance modeling formalisms, there is no generalization and reuse of model extraction software. Existing software supports only the extraction of a single modeling formalism. This single-formalism approach requires the reimplementation and maintenance of tooling for each architectural performance modeling formalism. While some model extraction approaches have been published, their sources or executables are either not available or lack documentation how to set up. Aggravating this situation, some of the existing performance model extraction tools [WK16; BK17; Bre16] are not open source.

3.4 Integration of Performance Evaluation in the Software Engineering Life-Cycle

To ensure timeliness and efficiency of software systems, performance engineering has to be integrated into software engineering processes. This section reviews state-of-the-art and related work for the integration of performance into the software life-cycle. This relates to all main contributions of this thesis (see Chapters 4, 5, and 6). Despite a growing awareness of the importance of software quality, the "fix-it-later" approach postponing performance evaluation to late development stages is still widespread [HHF13]. Discussing the integration of performance into software engineering, Section 3.4.1 presents approaches for continuous performance evaluation and Section 3.4.2 is about performance engineering at different software development stages.

3.4.1 Continuous Performance Evaluation

Continuous performance engineering repeats performance tests on a regular basis in order to detect performance regression. At this, purely measurement-based evaluation approaches dominate which are usually integrated into a Continuous Integration (CI) pipeline [LB17; FP17].

While researchers discuss continuous performance testing, a study by [Ste+17] recognizes that less than 1 % of their analyzed projects actually apply any performance testing framework. Based on a developer survey, they explain this by a lack of automation in performance testing. Moreover, the authors state that test execution time is a major issue. A study presented in [LB17] shows that performance tests are only a minor part of the test suite, which is rarely updated, and usually maintained by a small group of core project developers. Due to required time and efforts for testing, often only a subset of changes can be investigated in industrial scale projects. [Hua+14] propose to find a regression testing target by performing risk analysis. [Ahm+16] notice that tools for mining software repositories proposed in research are not readily available to practitioners. Therefore, they study the effectiveness of APM tools for detecting performance regressions. The authors consider them to be good candidates for detecting performance regressions due to their availability for practitioners and their out-of-the-box regression detection capability.

Besides detection of regressions, identifying the root cause for a performance regressions is challenging [HHF13]. Compared to functional bugs, fixing of performance problems is more challenging as it requires more expertise and more code to change [ZAH11; ZAH12]. Detecting performance problems during operation can be too late to solve them efficiently [HHF13]. However, it is infeasible to foresee and test all possible future infrastructures and usage of software. Consequently, some problems might be detected in production. Therefore, canary deployment [HF10] can be applied for risk minimization cushioning a lack of previous performance engineering activities.

Especially for complex systems, like software product lines, there can be a huge configuration space which is infeasible to test thoroughly. Therefore, many works concern with how to infer the configuration space based on a small subset of measurement points [WKH11; Thü+14].

Existing research mostly highlights challenges for continuous measurement-based performance evaluation. The idea to continually apply model-based performance evaluation in combination with measurements has been envisioned (e.g., by [WFP07; Kro+16]). However, to the best of our knowledge, this combination has not yet been instantiated in research or industry. A likely reason is that the efforts required to set up continuous model-based performance

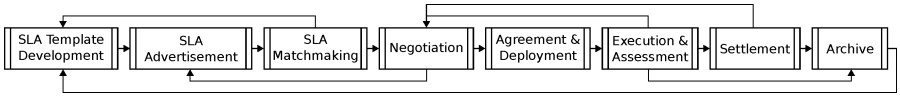


Figure 3.1: SLA life-cycle

evaluation are significant due to a lack of an appropriate holistic approach and respective tooling.

3.4.2 Performance Evaluation at Different Development Stages

As one of the first, [ZB93] discuss the industrial integration of performance engineering into a software product life-cycle. Like many researchers, e.g., [Smi81], they state positive effects of early integration of performance in the software engineering process. Since the infancies of performance engineering, several approaches targeting different development stages have been proposed, summarized by [Bru+15]. Until now, there is no broad integration of SPE in industrial software engineering processes. Deficiencies identified by [Men02b] include a lack of required scientific principles and models in software engineering, a lack of education in performance, the workforce composition, as well as single-user and small database mindsets.

While early works often refer to a software engineering life-cycle, recent work discusses performance evaluation at different stages of the software development cycle. The first implicitly assumes a waterfall development model where each stage is performed only once, while the latter refers to agile processes frequently repeating development cycles. Independent of development paradigm, different performance evaluation artifacts are available and different performance engineering tasks occur throughout the software development.

While the way of formulating performance indices and SLAs remains the same throughout the application life-cycle, SLAs definition, refinement, and processing can be based upon different performance evaluation approaches. The performance evaluation and consequently all SLA related actions should take into account that each life-cycle stage has different capabilities and optimization goals. To ensure the desired performance levels, SLAs should be evaluated throughout the entire software development life-cycle. This need is also underlined by the SLA life-cycle, depicted in Figure 3.1, which has been proposed by [Kri+13].

In the following, we consider the software development cycle shown in Figure 3.2. In each of its stages, different approaches are used to express and evaluate SLAs [Kri+13]. It starts with *planning* and *defining* stages. In

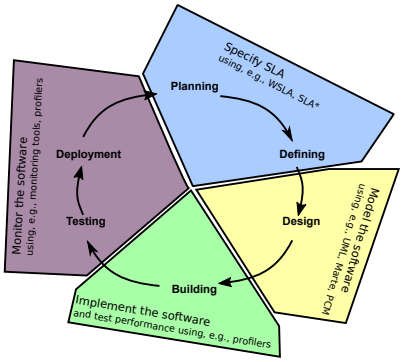


Figure 3.2: Handling of performance during software development cycle.

these stages, SLAs are created usually using some well-established standard, such as WSLA or SLA*. As most of these standards are XML based, this task is usually performed using an XML editor. In this stage, SLAs are defined based on the specified requirements. In the software *design* stage, software architecture can be modeled using UML (or similar notations) and annotated for non-functional properties like performance using UML Profile for Modeling and Analysis of Real-Time Embedded Systems (MARTE) or PCM. After the implementation (the *building* stage), the software goes into *testing* stage and then is *deployed* to production. During the design stage, based on software models, one can perform performance predictions [BKR09], to assess how the software will behave in relation to the defined SLA. Preliminary assessment of the implemented software performance can be already during implementation and then in testing stages. At this, the measured metric values are compared to the SLA. At run-time, SLAs can be evaluated against values obtained using a multitude of tools for APM and measuring software performance [Heg+17]. Also, model-based analysis approaches can be applied at run-time. Due to run-time requirements demanding for fast solutions, different modeling formalisms and solvers are applied [Spi+17a].

The life-cycle stage and evaluation scenario specific requirements determine which evaluation mechanism will be used and with which parametrization. However, for each of these mechanisms, SLA has to be translated into the respective tool format, which is a tedious and time-consuming task. Different SLA concerns, e.g., ensuring SLA conformance through auto-scaling or analysis of different software configurations, dictate an optimization of SLA evaluation process in terms of accuracy, time for evaluation, or system overhead.

3.4.3 Discussion

There are two main perspectives on integrating performance evaluation into software development: continuous performance testing and performance evaluation driven by development stage realities and needs.

Continuous performance testing can help to detect performance regressions by measuring performance metrics at the testing stage. Ideally, performance tests would run after each code change. "However, given the complexity of performance tests (e.g., requiring large lab setups, complex manual configurations, and lengthy executions times), per-checkin performance tests are rarely feasible in a large-scale industrial setting." [Ngu+14]. Model-based performance evaluation might reduce analysis time, but requires additional efforts for automation to be integrated into fully automated CI pipelines. Driven by specific information needs of stakeholders, performance engineering can be performed at different development stages. A multi-stage performance evaluation is challenging as the availability of performance evaluation artifacts and mechanisms changes. Throughout the software life-cycle, different performance evaluation artifacts are available and different performance engineering tasks have to be solved.

Even though fixing performance problems requires significant efforts [HHF13; ZAH11; ZAH12] and performance degradation can negatively impact revenues [Eat18], performance engineering is not widespread in the industry. Besides named challenges, there is a heterogeneous landscape of tooling for performance engineering and a lack of its integration. Therefore, the interoperability is costly and setup efforts currently overweight benefits. Non surprisingly, industrial projects usually do not exploit different types of performance analysis. During the last decades, approaches and tools for software engineering improved and increased developer productivity. In that regard, the frequency of producing new software versions increased rapidly which additionally fosters the requirement for automation.

Part II

Declarative Performance Engineering

Chapter 4

Declarative Performance Engineering Language and Processing Framework

Performance is a software quality attribute that defines the degree to which a system satisfies its requirements for timing behavior and resource utilization. Performance is of particular relevance to software system design, operation, and evolution because it has a significant impact on key business indicators. For instance, poor performance leads to a loss in revenue for companies relying on the availability of their systems and to penalties being due for service providers violating performance requirements. Consequently, during the life-cycle of a software system, performance analysts continually need to provide answers to and act on performance-relevant concerns such as about response times, hardware utilization, bottlenecks, trends, anomalies. Example performance-related concerns published by [RET18] include: How can we

- … reduce license costs by reducing our capacity?
- … reduce the number of servers or CPU cores?
- … estimate the required instances during a cloud migration?
- … put a new enterprise application into operation with a given capacity?
- … allow additional users with our current capacity?
- … calculate our IT costs based on the number of users and their capacity needs?
- … deal with changes in our user behavior?

During the past decades, various established methods, techniques, and tools for modeling and evaluating performance properties have been proposed—ranging from model-based prediction and optimization in early design phases, over measurement-based approaches in phases where implementations exist,

up to solutions that continuously evaluate and control a system's performance during operation. Hence, in principle, techniques to answer the aforementioned questions and goals are available.

Challenges Several challenges hinder existing Software Performance Engineering (SPE) techniques from being widely applied in industry for holistic performance management.

- There are manifold performance evaluation needs caused by different
  - stakeholders (e.g., infrastructure provider, service end-user, software developer, and software operator),
  - development stages (elaboration, construction, and transition)

  each supported only by a small subset of tools. Therefore, performance evaluation requires to apply multiple tools and toolchains.

- There is a notable set of strategies and composite algorithms for performance evaluation including measurement methods, simulation, and analytical solutions (cf. Chapter 2). Achieving interoperability between solution approaches can be cumbersome as it requires to cope with different kinds of system representations, interfaces, and model transformations. For example in [Eis+18], we apply five subsequent model transformations to solve an architectural performance model based on Queueing Petri Net (QPN) simulation. Interchangeably providing multiple strategies is associated with technical and conceptual challenges.

- There is an abstraction gap between the level on which performance-relevant concerns are formulated and the level on which performance evaluations are actually conducted [Wal+16a]. Established model-based and measurement-based methods exist. However, their application can be sophisticated, even for performance experts [WFP07]. It is often challenging to parameterize the chosen approaches appropriately and to filter and interpret the obtained results according to the initial concern. At this, the performance engineer has to consider non-functional constraints and tradeoffs.

- Performance evaluation is associated with significant efforts. It takes time to implement performance evaluation. Either a decision maker may not be willing to invest resources, or the development process does not account sufficient time for performance evaluation and management. Time-to-result is a particular challenge in light of agile software development

processes implying faster release cycles which have become more and more widespread in practice. Consequently, performance evaluation is only feasible by having end-to-end automation.

Research Questions In this chapter, we describe a novel declarative approach for an automated SPE addressing the previously described challenges. At this, we consider the following research questions:

- *What are performance analysis needs relevant in practice? How can they be classified? There is no unification or classification of performance concerns yet supporting a holistic performance evaluation.* We define concerns including evaluation of metrics and Service-level Agreements (SLAs), variations, what-if analyses, resource optimization, upper bounds for usage behavior.
- *How to formalize performance concerns independent of the performance evaluation approach? Can multiple evaluation techniques be generalized without information loss? What information is required to configure the evaluation techniques and what is supplementary? How to consider non-functional requirements (e.g., time-to-result or system perturbation) for the parametrization of performance engineering approaches?* We propose a generic language to specify concerns independent of the performance evaluation approach.
- *How to provide an easy to use specification that is both human and machine-readable? How to support the human language learning process?* To specify performance-related concerns, we chose not to use XML, which is machine-readable but can be difficult for humans [Par01; Kos+08]. We develop our own Domain Specific Language (DSL), as it is efficient in writing and easier to read and maintain, while still being machine-readable. Moreover, we provide a languages editor that supports the formulation of concerns using auto-completion and automated syntax highlighting.
- *How to automatically and efficiently answer performance concerns? How to reuse high-level performance evaluation algorithms for multiple evaluation approaches? How to design the architecture to accommodate future features into the declarative performance engineering framework? How to design the architecture to ensure scalability? How to design the architecture to ensure maintainability?* We provide a generic framework to automatically answer the concerns. Its features include: (i) extension point for element types, metrics, and statistic types, (ii) separation of concerns improving maintainability, (iii) high-level algorithms to process concerns, and (iv) concern oriented result visualization. For this framework we provide adapter for measurement-based and model-based performance evaluation.

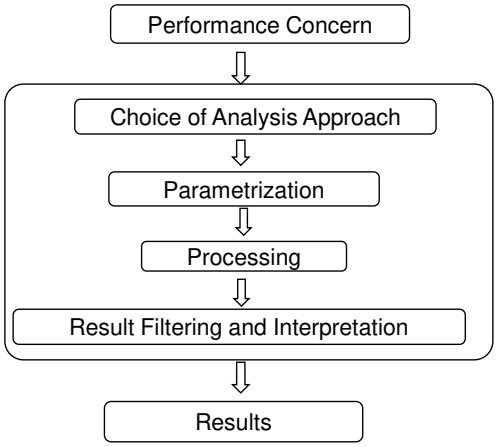


Figure 4.1: Performance concern answering workflow.

Chapter Outline The remainder of this chapter is organized as follows: Section 4.1 describes our approach to automate SPE based on a declarative language. Section 4.2 formalizes performance concerns based on a formalization of the domain. Section 4.3 presents a language to specify performance concerns. Section 4.4 explains our framework to process performance concerns. Section 4.5 elucidates the integration of measurement-based and model-based performance evaluation into our declarative performance analysis framework. Section 4.6 concludes this chapter.

4.1 Overview

During the life-cycle of a software system, performance analysts apply performance evaluation techniques to answer performance-relevant questions. A performance concern formulates the desire for a result of a performance engineering activity. Searching for solutions, performance analysts typically follow the process depicted in Figure 4.1 to answer to a given performance concern. Starting with a performance concern (specifying a specific question that needs to be answered), the process consists of the choice of a performance evaluation approach, its parametrization, processing, result filtering, and interpretation.

Problem statement

The application of the process to answer performance-related concerns can be complicated, time-consuming, and error-prone even for performance experts.

Existing performance engineering techniques require expert knowledge to apply them correctly. Different performance analysis techniques are applicable at the various stages of the software life-cycle. For performance engineering, currently lacks a unified interface that provides generalization, automation, and decision support. In particular, we identify a lack of a high-level language allowing to specify goals and queries independent of the solution approach (e.g., using measurement-based or model-based performance evaluation techniques). Further, there is no decision support for the selection of an appropriate performance engineering technique and tool for answering a performance query, considering functional and non-functional requirements (e.g., system perturbation) as part of the selection and parametrization of performance engineering techniques

The Declarative Performance Engineering Approach

To face these challenges, we propose the Declarative Performance Engineering (DPE) approach. The high-level objective of DPE is to support stakeholders, like system developers and administrators, in performance-relevant decision making. Performance analysts formulate their concerns using a declarative domain-specific language. The goal is to hide complexity from the user by allowing users to express their concerns and goals without requiring any knowledge of performance engineering techniques.

Definition 4.1. Declarative Performance EngineeringDPE decouples the description of the user concerns to be solved (performance questions and goals) from the task of (automatically) selecting and applying a specific solution strategy to answer a given concern [Wal+16a].

The goal of DPE is to enable the formulation and answering of performance-relevant questions and goals for a software system in a human-understandable manner by using a high-level *declarative language* to interact with performance engineering tools. The proposed language processing exploits a high degree of automation through a corresponding interpretation and execution infrastructure, which builds on established low-level performance evaluation methods, techniques, and tools. We aim to reduce the abstraction gap between the level on which performance-relevant concerns are formulated and the level on which performance engineering techniques are typically applied. The strict separation of *what* versus *how* enables the development of different techniques and algorithms to automatically select and apply a suitable approach for a given scenario. The core characteristics of DPE are:

- Enabling the performance analyst to *declaratively* specify *what* performance-relevant questions need to be answered without being concerned about *how* they should be answered. Particularly, the declarative language is independent of the performance evaluation approach. This flexibility enables the integration of measurement-based and model-based analysis.
- Hiding complexity from the user by automating the selection and execution of a solution approach, which may involve the application of multiple performance engineering techniques. To allow for flexibility, it should be possible to process the results from the various applied techniques manually, semi-automatically, or by using full automation.
- Supporting the whole software system life-cycle, including design, operation, and evolution. In particular, we consider modern software development paradigms, where development and operation merge the merge of development and operation (DevOps) [BWZ15].
- Supporting extensibility of the declarative language and the respective tools in terms of possible integrations of new performance evaluation techniques and concerns.

The language should be (i) generic (to support measurement-based and model-based performance evaluation), (ii) understandable, (iii) modular, and (iv) comprehensive, should (v) support tradeoffs and (vi) enable the combination of solution approaches. The following requirements illustrate our motivation for this work. The requirements aim at reducing the complexity of the performance analyst role so that it could be performed without extensive previous education and assigned to management, developers, or operators.

Requirement 1 *"I would like to evaluate performance with different performance evaluation methods."* Current practice evaluates software performance mainly in production. It would be beneficial to evaluate performance, e.g., using models, in the early evaluation of software development. The performance evaluation during the whole life-cycle would allow for performance problems to be avoided in later stages.

Requirement 2 *"I would like to evaluate various kinds of performance concerns."* There are manifold performance concerns such as the parametrization and surveillance of quality of service guarantees, sensitivity analysis, optimization.

Requirement 3 *"I would like to specify non-functional processing goals on time-to-result and accuracy which automatically trigger customized processing."* Dependent on descriptive processing constraints like "accurate" or "fast", technical configuration parameters like observation interval lengths or required confidence intervals should be derived automatically.

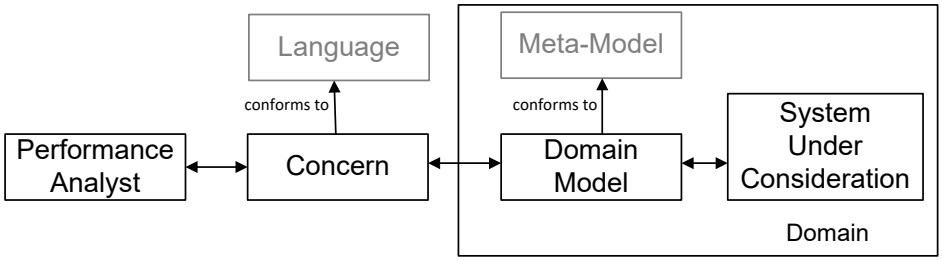


Figure 4.2: Relation between performance concerns and analysis context.

Symbol	Description	Part of domain model
$R = \{R_0, \ldots, R_j, \ldots, R_x\}$	Set of referable elements	Set of conceptual classes
$I = \{I_0, \ldots, I_j, \ldots, I_x\}$	Set of input parameters	Set of system attributes
$O = \{O_0, \ldots, O_j, \ldots, O_y\}$	Set of output variables	Set of derived attributes
$I_R = \{I_{R,0}, \ldots, I_{R,j}, \ldots, I_{R,n}\}$	Set of input parameters assigned to a referable element	Attributes of a conceptual class
$O_R = \{O_{R,0}, \ldots, O_{R,j}, \ldots, O_{R,m}\}$	Set of output variables assigned to a referable element	derived attributes for a conceptual class
$f_{O^a_{R,j}} : I_{O^a_{R,j}} \rightarrow O_{R,j}$	Evaluation approach a to calculate $O_{R,j}$ where $I_{O^a_{R,j}} = (I_k, \ldots, I_m) \subset I$ is the set of required input parameters for $f_{O^a_{R,j}}$	

Table 4.1: Notation used in formal problem description.

Requirement 4 *"I would like to evaluate performance throughout the whole software engineering life-cycle using an integrated approach that requires reasonable effort."* There is no integrated tooling for performance evaluation throughout the software life-cycle. Interoperability of tools and approaches is challenging. Realization currently requires cumbersome connection of several tools and approaches.

Please note that the remainder of this chapter focuses on the generic performance concern language and corresponding processing framework. The highly related topics of performance model creation and decision support will be discussed in Chapter 5 and Chapter 6, respectively.

4.2 Formalization of Performance Concerns

The definition of performance concerns requires a description of the problem domain to which the declarative language features can be mapped. The high-level interplay of performance analyst, concern, language, domain model, and system under consideration is depicted in Figure 4.2. The approach presented in this thesis allows for any type of model that can be mapped to our formalization.

At this, we refer to the already established concepts and terminology of domain modeling (e.g., [Lar01]). Domain models, also referred to as conceptual models, are solution-independent descriptions of a problem domain. They do not capture any system design choices but focus on the perspective and language for the domain under consideration.

The *domain model* describes the problem domain which is the system under consideration. It reflects our understanding of real-world entities, their relationships, and their responsibilities. The domain model is described in a formal description language called meta-model offering semantics to interpret concrete instances. As we intend to generalize over concrete meta-model implementations and allow for different system views, we rely on the concepts for conceptual classes, attributes, and associations [Lar01]. Technically, domain models representations are similar to UML class diagrams except that conceptual classes do not allow to specify methods. To perform performance analyses, these domain model concepts have to be mapped to performance evaluation disciplines as discussed later.

The performance analyst can formulate concerns based on a subview of the domain model. The formal notations, summarized in Table 4.1, allow understanding the problem from a formal perspective. For concern processing, we distinguish input parameters and output variables. These basic concepts have different names depending on the analysis discipline, like experimental design (factors, response), regression analysis (predictors, criterion), sensitivity analysis (input and output factors), or optimization theory (input variables, criterion). Mapped to the domain model, input parameters describe attributes and output variables correspond to derived attributes.

Definition 4.2. Domain model view We define a system domain model view as a tuple $S = \{R, I, O, A\}$, where $R = \{r_0, \ldots, r_v\}$ is a set of referable elements, $I = \{i_0, \ldots, i_w\}$ is a set of input parameters (system attributes), $O = \{o_0, \ldots, o_x\}$ is a set of output variables (derived attributes), and $A = \{a_0, \ldots, a_y\}$ is a set of allowed adaptation operations, $v, w, x, y \in \mathbb{N}$

In extension to classical domain modeling, the processing of derived attributes may include accessing elements connected using associations. To formulate concerns, we reuse the grouping provided by the conceptual classes, we refer to them as Context References (CRs). A CR may include different aspects, e.g., criteria to search for, input parameters, and associations to other elements. The latter enables to depict relations, interconnections and hierarchies form the context. CRs can be defined at different granularity levels. For example, the infrastructure can be investigated as a whole or focusing on a single server or even a single resource like CPU. For services provided exter-

nally one may investigate metrics visible to the end user or also investigate the internal behavior of processed subservices.

Output variables correspond to performance metrics like response time, utilization, or throughput. Their values depend on input variable parametrization. Input variables are usually subject to technical or organizational restrictions. Therefore, input variables have a (limited) set of allowed parameterizations. These parameterizations could be specified by a range and a step size or by an explicit list of alternatives. For example, the number of cores for a CPU could be limited to {1, 2, 4, 8, 16, 32, 48}.

Definition 4.3. ParametrizationThe parametrization of an input variable $i \in I$ means assigning it a value from its value range.

Input variables are not located in an empty space but can be assigned to corresponding entities. We refer to them as CR For example, a CPU frequency input variable can be mapped to its CPU. An arrival rate input variable can be mapped to a workload description. A CR holds the input variables naturally tied to it and not necessarily the ones required to derive its output variables. Output variables can be derived as a function of required input variables which themselves may be spread over multiple CRs. For example, the response time of a service depends, besides others, on CPU clock frequencies. Output variables are numerical values for performance indicators, e.g., service response times or resource utilization. They can be derived using elementary performance analyses as defined below. Note that the method to derive a particular output value can be ambiguous. There can be alternative ways to derive the same output variable, e.g., using analytical or simulation-based analysis.

Definition 4.4. Elementary AnalysisAn analysis that derives elementary performance or costs metrics can be formalized as a function that maps input variables to real number(s), $f_{O^a_{i,j}} : I_{O^a_{i,j}} \rightarrow \mathbb{R}^n$, denoting the value(s) of output variable $o_{i,j}$ for a system configuration. The value of n is one if it is an aggregated value. The concrete method to derive output variables is indicated by a.

Definition 4.5. Analyzable ConfigurationA system configuration is analyzable regarding an evaluation approach a if $I^a_{O_{i,j}}$ is fully parameterized, where $I^a_{O_{i,j}}$ denotes the set of required input parameters for an approach a to calculate $o_{i,j}$. For the evaluation of $o_{i,j}$ required inputs may differ for evaluation methods $a, b : I^a_{O_{i,j}} \neq I^b_{O_{i,j}}$.

Based on the results of elementary analyses, there can be superimposed analyses. Examples of interpreting analyses include bottlenecks analysis [And+13; Jai91] or root cause analysis [HHF13].

Definition 4.6. Interpreting AnalysisAn interpreting analysis takes the results of multiple elementary analyses and returns references to context elements to indicate problems or challenges to be solved. It can be formalized as a function: $f : O \rightarrow R$.

Variation and Adaption While some problem scenarios focus on a single system state, others investigate multiple (potential) system states. Values for design space exploration can be derived using allowed input parameter ranges. In extension to classical domain modeling, we allow for the specification of adaptation operations. An adaptation may change the parametrization of input variables and create or delete conceptual classes and associations.

Definition 4.7. Adaptation An *adaptation* is a function a that transforms an initial system configuration S into a resulting system configuration S'. It can be formalized as a function $a : S \rightarrow S'$. An adaptation may include three types of change operations on the domain:

- creation or deletion of conceptual classes,
- parametrization of an attribute, or
- instantiation or removal of associations between conceptual classes.

IT systems may be varied in multiple ways. Several adaptations cause no functional change to the system, for example, changing resources or software tuning parameters. In that context, finding optimal configurations is a challenging problem for many scenarios due to a large number of parameters that can influence performance. Variations may be at hardware infrastructure, middleware, and application software level. Moreover, adaptations may change the functional behavior of the system. For example, advanced websites allow discarding optional content in case of heavy load. In contrast to design space exploration commonly denying such variations [Koz11], our approach allows for such variation scenarios in what-if analyses. Besides actively specifying adaptations for which effects should be investigated, modeled adaptations may be used to determine a sequence of operations towards a targeted system configuration.

While a system administrator intentionally triggers some adaptations of the analysis scenario, other changes stem from external use.

Exemplary adaptations not controlled by the administrators include changing usage patterns of the system, changing system behavior due to software or hardware failures, or changed system behavior triggered by different kinds of network attacks [Iff+18b].

Definition 4.8. Degree of freedom A Degree of Freedom (DoF) specifies options for a variation of the system (or its representation). It can be formalized as a set

of options $D = \{d_0, ..., d_n\}$ for which the corresponding adaptation function $f_{d_i} : (S) \rightarrow S'$ sets the value d_i at the variation point D in S. All DoF adaptation functions have to be transitive relations: $\forall\ f_{d_i}, f_{d_j}, d_i, d_j \in D$: $(f_{d_i} \circ fd_j)(S) = (f_{d_i}(f_{d_j})(S)) = f_{d_i}(S)$.

Broadly speaking, the parametrization of a DoF does not change the system in a way that the DoFs cannot be varied again in the same manner. This allows spanning a vector space of system configurations. DoFs include all input parameters and a subset of adaptations. Some adaptations, such as replications of resources or services can be modeled as DoF. Other adaptations, such as migration of one component instance from one infrastructure to another infrastructure, are not covered by the DoF concept and cannot span a vector space of system configurations.

Presumptions, Constraints, and Contracts Systems can be evaluated based on presumptions about the delivered Quality of Service (QoS). In the software domain, such presumption is commonly referred to as SLAs defining upper bounds for output variables.

Definition 4.9. SLAAn SLA contract c can be described by a set of Service-level Objectives (SLOs), $c = \{slo_0, ..., slo_n\}$, where each slo is a tuple of output variable and threshold value, $slo_i = \{o_x, threshold\}$, with $o_x \in O$ and $threshold \in \mathbb{R}$.

SLAs are usually written as contracts connected to penalties on violations. Further, there are additional refinements for SLA definition which are not relevant for the conceptual discussion here and are discussed later at the language definition. SLAs can be associated with manifold performance evaluation concerns: A system configuration S complies to an SLA if the values for all output variables are below the corresponding thresholds. Also, S can be evaluated quantitatively by counting its SLA violations or respective penalties. Moreover, one may search for cost-efficient system configurations satisfying a given SLA. Last but not least, thresholds can be parametrized based on performance evaluation results.

System Configuration Optimization System configurations can be optimized for utilized resources and performance indices. Optimization of a single objective without constraints usually adds no valuable information. For example, optimizing response time or throughput of a system without considering resource efficiency results in providing the maximum of available resources. Optimizing resources without considering performance indices withdraws the system all resources. Optimization problem definitions can be specified

using objective functions based on either maximum or minimum. For the sake of conciseness, we refer only to minimization without losing generality, as a maximization function is equivalent to minimize its negative and vice versa. Investigations may provide useful information when quality metrics are constrained by SLAs or there are multiple optimization functions. Translated to performance engineering, cost optimization functions may target maximum throughput, minimum response times, or lowest resource costs. The constraint vector of upper bounds can be derived from SLA definitions. The unconstrained multi-objective optimization problem can be defined as follows:

Definition 4.10. Multi-objective optimizationMulti-objective optimization, also referred to Pareto optimization, searches for non-improvable compromise solutions in the presence of trade-offs between two or more conflicting objectives. It can be formalized as

$$\{s \in S | \min(c_1(s), \ldots, c_n(s)) :$$

$$\forall s' \in S : cost_i(s') < cost_i(s) \rightarrow \exists cost_j : cost_j(s') > cost_j(s)\}$$

where $k \geq 2$ and $s \in S$ is a feasible system configuration and $cost_i$, $i = 1...n$ a set of cost functions.

For multi-objective optimization, there is typically no solution that minimizes all objective functions simultaneously. Therefore, multi-criteria optimization searches for solutions that cannot be improved without degrading one of the other objectives. Such solutions are also referred to as Pareto optimal solutions. From the Pareto optimal solution, a human decision maker has the select an appropriate configuration to be instantiated. At this, the decision maker implicitly weights the cost functions. If weights are known in advance, one may merge multiple objectives, cost functions respectively, into a single one. Single objective optimization is a special case of optimization where $n = 1$. Single objective optimization comes with the advantage that it returns a single optimal system configuration without the requirement of a decision maker.

Definition 4.11. Transforming multi-objective to single objective functions Multiple objective functions can be merged into a single objective function using a weighted sum $c'(s) = \sum_{i=1}^{n} w_i c_i(s)$ with $w_i \in \mathbb{R}$ where $s \in S$ is a feasible system configuration which is minimized $\min(c'(s))$.

When transforming a multi-objective into a single objective function, it can be challenging to assign appropriate weights to scale the cost functions. However, this effort is rewarded by a single optimal configuration.

Performance Concerns The previous formalization can be summarized to define performance concerns. We consider a performance concern as a function that answers a performance-related questions or contributes to its goal fulfillment.

Definition 4.12. Performance ConcernA performance concern formulates the desire for a result of a performance engineering activity. We formalize the activity as a function f that may return real numbers, Boolean values, context elements R, adaptations, or system configurations on the input of a system description S, variations, adaptations A, output variables O or constraints (SLAs) C, and non-functional processing constraints N,

$$f : \{S \times A \times (O \mid C)\}N \rightarrow \{\mathbb{R}^i \times \mathbb{B}^j \times R^k \times A^l \times S^m\}$$

where $i, j, k, l, m \in \mathbb{N}$. Particular concerns can be based on a subset of inputs and point to a subset of outputs.

As shown in the formalization, the answers to performance concerns can be manifold and include performance indices, Boolean SLA adherence, quantities of violations, penalty costs, context elements like bottleneck resources or services denoting the root cause for a performance degradation, adaptations (required to satisfy an SLA) and cost-efficient configurations (still satisfying a given SLA).

Performance concerns can be classified according to different dimensions touched in the previous formalization. Summarizing dimensions, concerns can be subdivided according to:

Post-processing Complexity Concerns may be classified according to their post processing complexity. *Elementary concerns* include the evaluation of basic metrics for a specific state that can be directly derived from measurements, simulation, or analytical solvers and require no post-processing. In contrast, *high-level concerns* are based on algorithms that provide further processing of elementary performance indices. Examples of high-level concerns include SLA evaluation or the detection of bottleneck resources.

Investigated Number of System States Performance concerns may be evaluated based on a single system state or a set of system states. Consequently, we may subdivide into analyses requiring variation or adaptations of the system under consideration (or its respective model-based representation) or not. Varied systems further can be compared according to relative differences or absolute behavior.

Level of Presumption The investigation of IT systems may be assumption-free or based on hypotheses and quality goals. *Assumption-free analyses* dismiss assumptions or constraints about the system. This can be, for example, queries for bare metrics. In contrast, *presuming analyses* investigate the system based on previous assumptions or goals on resource efficiency or QoS (transcribed as SLAs).

Result Type The classification may be according to the result type. Possible result types include: (i) binary (e.g., SLA conformance), (ii) (Sets of) numerical values (e.g., performance indices, the minimum number of resources, maximum arrival rate, amount of penalties for SLA violations), (iii) system elements (e.g., a bottleneck resource), (iv) system adaptation operations (required to ensure resource efficiency and QoS), or (v) system configuration.

The presented dimensions each highlight a particular aspect of formulating and processing concerns. The *post-processing complexity* dimension implies that there are generic processing algorithms that can be reused for multiple performance evaluation approaches. The dimension on *investigated number of system states* implies that a subset of concerns requires automated mechanisms for system re-parametrization and adaptations. The *level of presumption* dimension states that some concerns require SLA input and a corresponding result presentation. Also, the *result type* dimension affects the presentation of results.

In general, a structured view using a taxonomy can be helpful to understand a problem domain —which is in our case understanding a performance concern language or its solution framework. However, except for SLA evaluation (which implies post-processing) the structuring at named dimensions is disjoint. Therefore, a strictly hierarchical classification, also referred to as taxonomy, cannot be implemented based on the presented dimensions. Performance concerns allow for hierarchical structuring considering only a subset of criteria but not all at the same time.

Based on the introduced notion, we can define many different types of analyses carried out in the next section. We do not distinguish concerns according to their performance evaluation methodology as every concern can be answered based on model-based or measurement-based performance evaluation results. The described view generalizes over the meta-model for system description. CR may be partially defined, e.g., a context reference may define only parameters but no derivable metrics. To not specify every context element on its own, we may rely on CRs types that specify input and output variables, adaptations, and links, that can be reused in different contexts. For example, all CPUs can

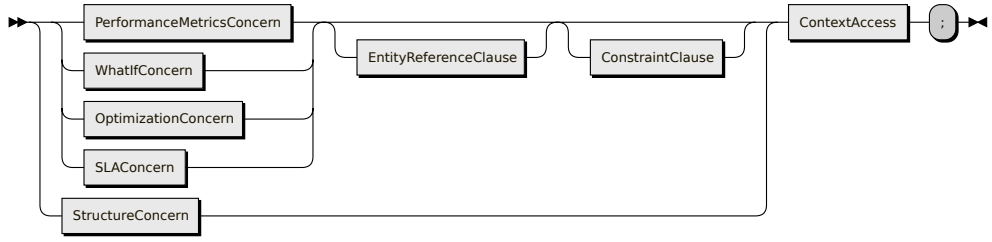


Figure 4.3: Concern language syntax overview.

be investigated for utilization. Also, processing frequency or the number of used cores can be modified for all CPUs.

4.3 Declarative Performance Concern Language

The goal is to define a descriptive domain-specific language to communicate performance concerns about arbitrary systems. In the following, we describe fundamental and orthogonal language concepts in Section 4.3.1. Then we describe the syntax for self-description in Section 4.3.2, evaluating basic performance indices in Section 4.3.3, relative concern formulation in Section 4.3.4, and the syntax to evaluate SLA related concerns in Section 4.3.5. Section 4.3.6 describes the syntax to formulated optimization problems. To present concepts concisely, we do not introduce all language constructs to the last detail but refer to a full specification added to the appendix.

4.3.1 Syntax of Basic Concepts

Several fundamental language concepts can be orthogonally applied for the formulation of multiple performance concern types. They can be applied to solve multiple types of concerns. Therefore, we introduce them prior to the specification of specific concern types.

Context and Evaluation Method(s) Specification

Requirement 1 demands to support different performance evaluation methods. Hence, the language has to provide means to specify the performance evaluation method. Additionally, the system under consideration or its representation has to be specified. To formulate concerns, we need to connect to model instance and meta-model. To be interpretable, each languages statement

Figure 4.4: Syntax ContextAccess.

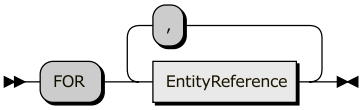
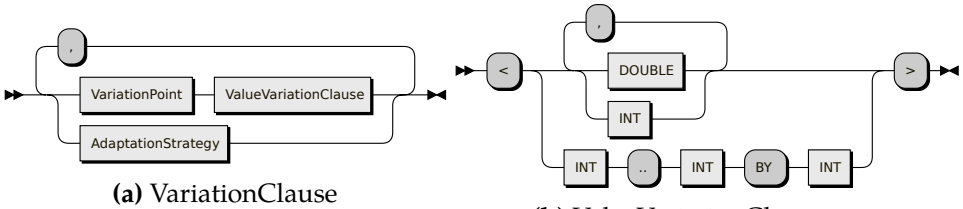


Figure 4.5: Syntax EntityReferenceClause.

ends with a `ContextAccess` (Figure 4.4) to specify the system under consideration and the performance evaluation method. The `ContextAccess` starts by a `USING` terminal, followed by an identifier of the `connector` implementation that bridges processing of performance indices to an existing performance evaluation mechanism. A domain access specification finalizes the statement. An @ operator separates connector and domain access. The domain access is text that specifies the access to the system under consideration and can be a link or path to a performance model, log files, or to a script that triggers load generation and performance measurements. In general, this context access information could be cached on initial input or entered for every query.

Aliasing

Usually, the performance analyst has low control over how external performance evaluation toolchains and respective performance evaluation artifacts name the system entities. Entity descriptions may be lengthy, inconvenient, or misleading. Therefore, we allow introducing aliases when specifying concerns. Metric or element descriptions can be concise when using custom identifiers and abbreviations. The `EntityReferenceClause` can be used to define aliases for queryable elements like resources or services. It provides a mapping of a query to the context description. Context descriptions might differ for different analysis methods, for example, some are based directly upon the system under consideration and others are based upon a performance model. Supporting the use of different identifiers, the `EntityReferenceClause` contributes to interoperability between different approaches. The `EntityReferenceClause` starts with the `FOR` terminal followed by one or more `EntityReferences`. An `EntityReference` starts with the type terminals, followed by `EntityID` derived from the context and the `AS` keyword and the alias. By default, we support `RESOURCE` and `SERVICE` as type terminals. However, the set of types can be expanded using an extension point.



(a) VariationClause

(b) ValueVariationClause

Figure 4.6: Syntax for variation of the system.

Processing Constraints

Performance can be evaluated using various approaches and configurations differing in analysis costs and accuracy. Hence, our framework enables to vary solution strategies. We propose not to trigger a particular solution strategy explicitly but to specify an optimization goal for processing instead. This relieves the analyst from learning in-depth knowledge of solution techniques and solvers.

Besides functional specification of desired results, there can be non-functional constraints on performance evaluation. Requirement 3 demands a language construct to support cost and accuracy tradeoff and optimization of processing accordingly. The optional `ConstrainedClause` may optimize processing according to non-functional constraints. The clause starts by `CONSTRAINED AS` and a constraint specifier. For example, constrained as `fast` triggers approximative solutions for model-based analysis, or may be interpreted by a measurement-based performance evaluation method to trigger shorter measurement periods and extrapolate afterward. Constrained as `accurate` may aggravate the termination conditions for simulation. The interpretation is specific to the performance evaluation method and depends therefore on the concrete performance evaluation method and its implementation.

System Variation and Adaptation

A `VariationClause` can be used to describe system variation and adaptation. It is a comma-separated list of variations and adaptations as shown in Figure 4.6a. Each variation starts with a variation point identifier (with an optional alias) and is followed by a `ValueVariationClause` that provides parameter settings (see Figure 4.6b). We support lists of parameter values of type Integer or Double as well as interval definitions of type Integer.

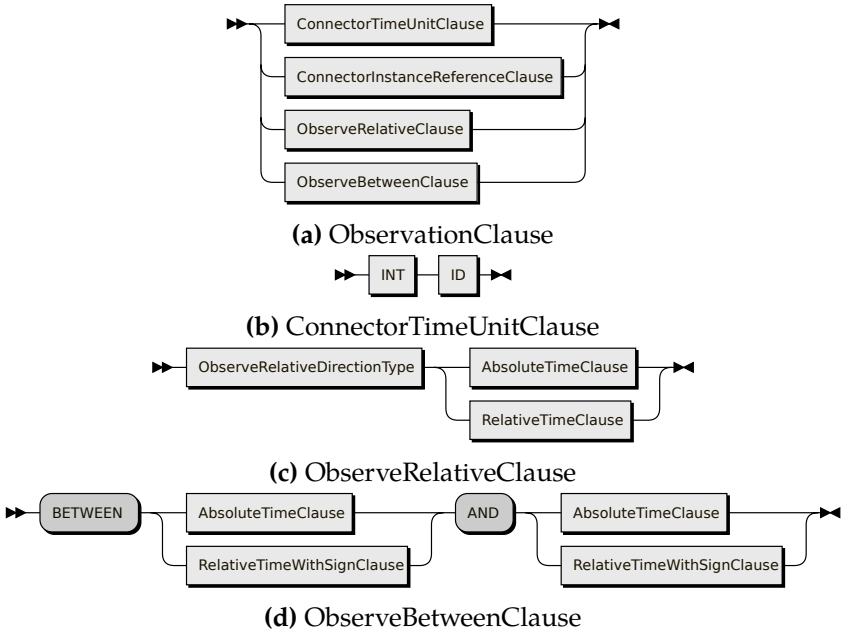


Figure 4.7: Syntax to explicitly describe the observation interval.

Observation Interval

Observed metric values depend on the observation interval. In case of an infinite workload definition, we implicitly assume the analysis of steady-state behavior. Otherwise, if the workload model specifies experiment time, we apply the time interval of the workload definition. As a third option, the observation interval can be specified explicitly using an observation clause. Figure 4.7 shows the respective syntax.

4.3.2 Self-description

The user can request information about queryable elements and their metrics and respective statistical resolution. The elements and metrics that may be asked depend on the domain description, more specifically context meta-model and model instance. `ModelStructureConcerns`, depicted in Figure 4.8 provide means to inspect the model description. They enable to investigate what can be asked. Our language subdivides into Context types (Elements the connector allows to ask for), their instantiations (Context elements), and supported options for system variation (points and strategies).

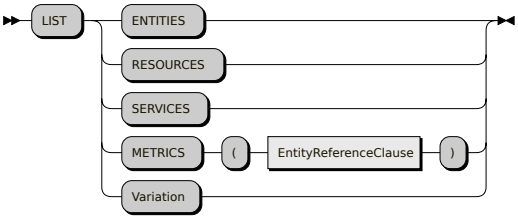
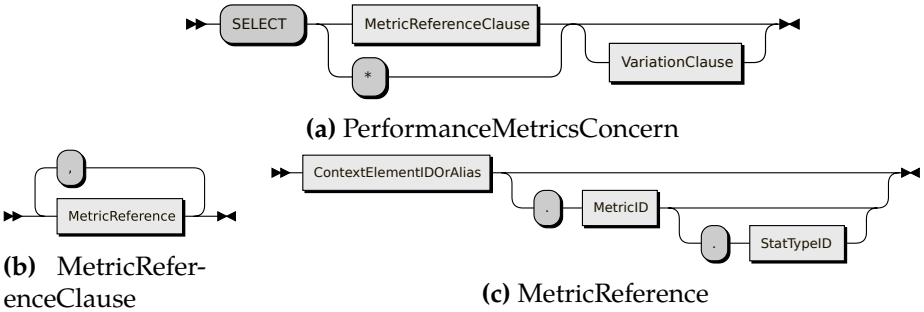


Figure 4.8: ModelStructureConcern enabaling self-description of the system.



(a) PerformanceMetricsConcern

(b) MetricReferenceClause

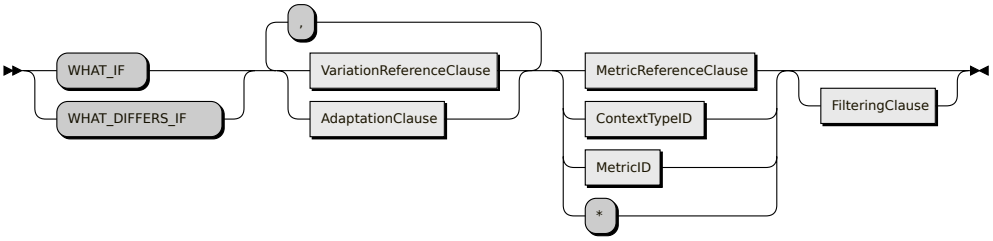
(c) MetricReference

Figure 4.9: Syntax to query basic performance indices.

Structure queries enable to request capabilities of the language for a given context. Moreover, the framework processes `StructureQueries` internally to generate options for auto-completion to enhance usability.

4.3.3 Performance Indices Evaluation Syntax

Querying basic performance indices represents a fundamental information need [GBK14]. Figure 4.9 summarizes the respective syntax. Figure 4.9a shows the syntax of a performance metrics concern. The performance metrics concern specifies metrics of interest using a metrics reference clause, which is a comma-separated list of `MetricReferences`, presented in Figure 4.9b. The syntax of `MetricReferences`, presented in Figure 4.9c, refers to the information need in a three-tier hierarchical manner: context reference, metric, statistic type. In contrast to aggregation functions, modeled by related work (cf. Related work on modeling performance in Section 3.1), we introduce statistic types. There are many reasons to describe what should be derived and instead of how to derive. For example, strictly speaking, a mean response time of a service derived using Mean Value Analysis (MVA) is no aggregation. As an alternative to the explicit



(a) WhatIfConcern

(b) FilteringClause

Figure 4.10: Syntax to trigger What-if analysis relative to current state.

specification of desired metrics, we included a wildcard operator '*' to query all metrics that can be derived from a system.

In extension to the investigation of a single system configuration, the performance metrics concern can be used to investigate multiple system configurations. This happens using a `VariationClause` (introduced in Figure 4.6b). A typical SPE activity we support by this feature is sensitivity analysis. Sensitivity analysis is the study of how the uncertainty in the output of a system can be apportioned to different sources of uncertainty in its inputs. For example, having an expected arrival rate of 150 requests per second which is associated with an estimated uncertainty of 50, one may also analyze arrival rates of 100 and 200 requests per second or multiple values in between within a single concern.

4.3.4 What-if Evaluation Syntax

The question on what happens on changes to the system or workload configuration is a common analysis concern [TNG06; The+10]. It can be answered formulating a `PerformanceMetricsConcern` containing a `VariationClause`, introduced in the previous section. This answers how the system behaves for a particular configuration, however, not in relation to the current state. In contrast to `PerformanceMetricsConcerns`, `WhatIfConcerns` implicitly capture a comparison to the current state. This *state-dependent* concern description requires a fully parameterized system description to efficiently formulate an interpretation of metrics for system variation(s) in relation to the current state.

Figure 4.10 shows the structure of `WhatIfConcerns`. It starts with a terminal which is either the keyword `WHAT_IF` or `WHAT_DIFFERS_IF`. Then a definition

of variation(s) follows. Afterward, it has to be specified which metrics should be considered. This can be done referencing metrics of concrete elements, generically referring to a type, or investigating all available performance metrics which is marked by a wildcard operator '*'.

The `WHAT_IF` terminal requests performance evaluation results for the current and changed state(s), the `WHAT_DIFFERS_IF` is about the resulting delta(s). Let us illustrate this by an example: We assume the system utilization is requested, the current utilization is 50%, and the utilization of the changed setting is 80%. `WHAT_IF` returns both values, `WHAT_DIFFERS_IF` returns 30% which corresponds to a subtraction 50% from 80%. `WHAT_DIFFERS_IF` suits to automatically detect degradations. `WHAT_IF` provides more insights to a human decision maker.

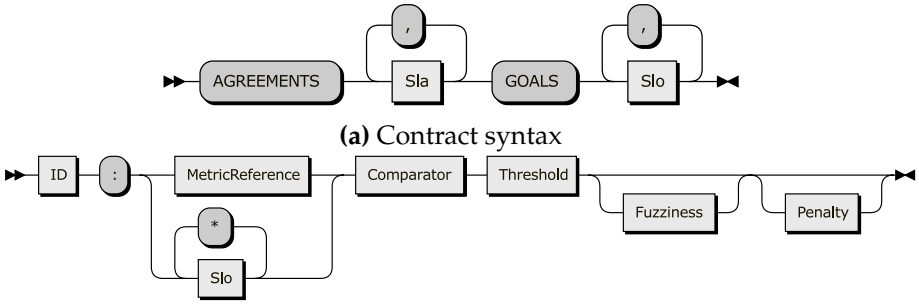
4.3.5 SLA Processing Syntax

In the presence of quality of service requirements to satisfy customers and (partly) outsourced processing of requests, SLAs depict a core concept. SLAs can be evaluated according to different goals including: (i) evaluation of a single configuration, (ii) design space exploration investigating multiple configurations or load levels, (iii) the recommendation of reconfiguration options as an input for auto-scaling. Reactive analysis can detect that there was a violation in production, proactive analysis detects that there will be a violation.

The syntax definition of SLA related concerns can be split into plain SLA contract definition which is then embedded into processing statements. While this work does not contribute new SLA language concepts, the *novelty* here is also to specify processing constructs to allow for different analyses. In contrast to most frameworks using XML markup language definitions, we provide a DSL definition that interprets order of constructs and therefore allows more concise formulation.

Contract Language

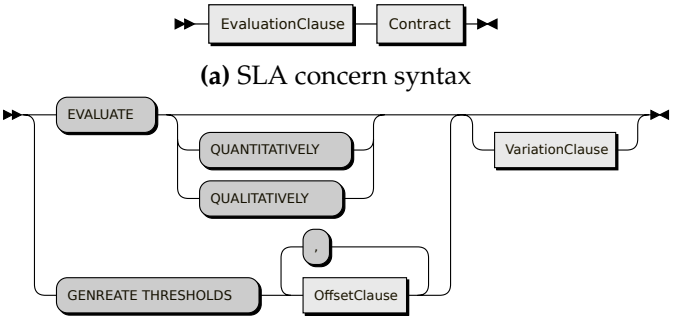
Our contract syntax, summarized in Figure 4.11, orients at concepts from established specifications such as WSLA or CSLA. Figure 4.11a provides a skeleton of the contract definition. An SLA definition starts with the `AGREEMENTS` identifier and contains one or more SLA definitions. An SLA references a set of weighted SLOs using the `CONTAINS` identifier. The weighting of SLOs is used for their prioritization. The SLO block starts with the `GOALS` keyword and contains one or more SLO definitions. An SLO definition, depicted in Figure 4.11b, consists of its unique identifier, a `MetricReference`, a `Comparator` ($<, \leq, >, \geq$) and a `threshold` value. `MetricReference` represents the entity and its attribute that



(a) Contract syntax

(b) SLO syntax

Figure 4.11: Syntax to specify SLA contracts.



(a) SLA concern syntax

(b) Evaluation clause syntax

Figure 4.12: SLA contract processing syntax.

is being evaluated which is typically the response times of a service. Fuzziness and penalty definitions can be optionally included. `FUZZINESS` describes an allowed deviation from the desired threshold of an SLO, to blur the metric's threshold. This optional parameter may be used to cope with uncertainties and fluctuations which may occur in cloud environments [KL12]. The penalty definition starts with the `PENALTY` keyword, and it is possible to restrict the penalty to apply for multiple violations. We allow specifying thresholds based on (i) the absolute number of violations, (ii) violations per time interval, or (iii) proportional to processed operations.

Contract Processing Syntax

An SLA contract definition can be used for different types of analyses, as illustrated in Figure 4.12. Figure 4.12a depicts the coarse-grained syntax to process

contracts, which is an `EvaluationClause` followed by a contract definition (introduced in Figure 4.11). The kind of concern to be processed can be specified in the `EvaluationClause`. Its syntax, depicted in Figure 4.12b, allows specifying multiple types of analysis. The processing identifiers specify the action that will be performed based on the SLA contract definition. Note that SLA constrained optimization will be discussed later together with optimization syntax.

Quantitative and Qualitative Evaluation

The most common analysis concern is to evaluate conformance to SLAs. The evaluation of SLA conformance starts with the `EVALUATE` keyword. The following refinement allows distinguishing `QUANTITATIVE` and `QUALITATIVE` analysis. The first triggers evaluation of all violations and calculates penalties resulting in a detailed report. The latter describes if an SLA has been satisfied or not. If omitted, we trigger a quantitative analysis by default.

Threshold Parametrization

Guaranteed SLAs are essential when selecting infrastructure or service providers. Thresholds should not be overcautious as customers would then decide for another provider with similar or even worse service quality but with better guarantees. Also, thresholds should not understate quality metrics due to costly penalty fees on violations.

SLA thresholds can be parameterized by an educated guess or based upon performance evaluation results investigating system capabilities. The latter means that SLA thresholds can be defined based upon performance evaluation results. Based on experiments, contracts can even be automatically generated. Our language and analysis framework allows, for example, to use measurements or simulation data to specify thresholds tailored to application capabilities. Triggering SLA threshold parametrization means to rewrite thresholds based on performance indices. The corresponding language statement starts with the `GENERATE` keyword which can be followed by an optional `OffsetClause`. We allow for multiplicative and additive offsets. For example, `<MULTIPLY 1.25, ADD 0.5>` specifies that performance indices derived through performance evaluation have an offset of twenty-five percent followed by a 0.5 increment. If a service response time aggregate, for example, mean, is two seconds, the generated SLO threshold would be three seconds. If the metric is not aggregated to a single value, the maximum of evaluation results is taken.

Finding Upper and Lower Bounds

System description may contain variation points or adaptation operations. In general, variations can be orthogonally appended to the presented SLA processing. Usually, one is not interested in varying all DoFs due to design space explosion. Therefore, we intentionally do not provide a wildcard operator. The `VariationClause` allows specifying variations. When the connector allows modifying the system (or the model representing the system), multiple analyses can be started. This allows, for example, to evaluate SLAs for different user counts and arrival rates, or evaluation using different numbers of resources.

In addition, multiple system configurations can be analyzed for SLA conformance to find upper and lower bounds for system and workload parametrization. Based on `VariationPoints`, one can find resource efficient (and thereby cost-efficient) configurations that still satisfy SLA contracts. The search for upper and lower bounds starts with the `SELECT` keyword, followed by an objective, which is a pair of an optimization function and an adaptation point. Separated by comma, additional objectives can be appended to describe a multi-objective optimization.

Terminals `MIN` and `MAX` describe optimization functions. They enable for example to request the maximum arrival rate that can be satisfied for given SLA definitions (cf. Figure 7.10) or the minimum number of resources required to satisfy a given SLA definition.

4.3.6 System Configuration Optimization

The quality of software architectures and system configurations can be rated according to multiple criteria. A common concern is to optimize the system configuration for conflicting objectives such as resource or cost minimization compared to delivered performance [Koz11; Goo+12; WK18].

Generally, multiple Pareto optimal solutions for multi-objective optimization problems exist. Consequently, solving such problems can be more elaborate than solving a single-objective optimization problem. In decision making, three methods to handle optimization problems with multiple criteria can be distinguished based on when decision makers have to articulate their preferences for solutions: a priori, a posteriori, and interactive preference articulation.

A posteriori optimization approaches aim to provide all the Pareto optimal solutions or a representative subset of the Pareto optimal solutions. Subsequently, a decision maker may select the most preferred Pareto optimal solution according to subjective preferences. This commonly relies on a human decision maker who should be expert in the problem domain. A priori optimization

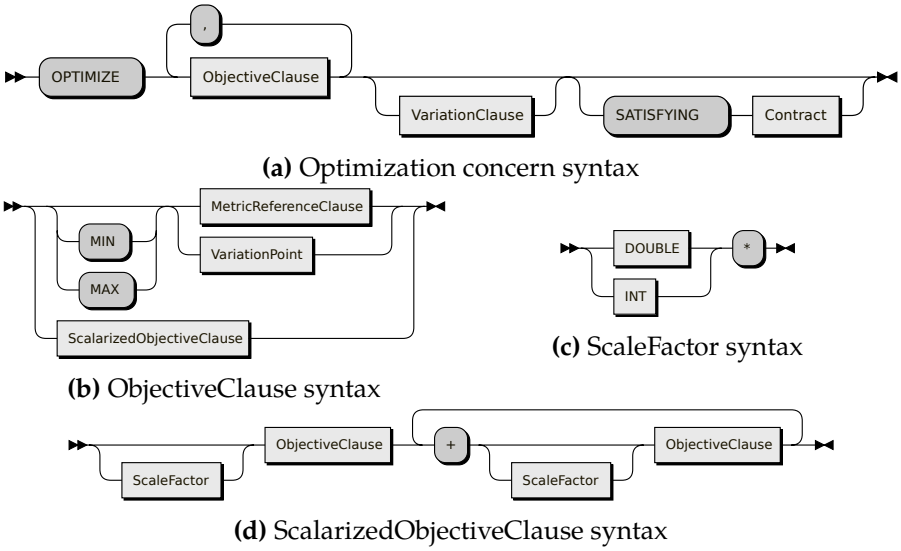
methods assume such preference formulations from the start. In interactive methods, the solution process is iterative, and the decision maker continually states preferences when searching for the most preferred solution. Interactive methods terminate on psychological convergence which is when the decision maker is confident of detection of the most preferred solution available.

We integrate the optimization of systems configurations into our declarative language as shown by the syntax presented in Figure 4.13. Figure 4.13a shows the syntax for optimization concerns. For all types of optimizations, the system configurations to be considered can be specified explicitly within the language using a variation clause, or by omission. For the latter, the processing internally maps to a structural concern. It derives all DoFs of the internal DoF-model without restrictions. Further, all optimizations can optionally be restricted by SLA contracts. The restriction cause that only system configurations are permitted that can ensure all service level objectives specified by the contract. The contract syntax can be found prior to this optimization syntax. The syntax allows describing a Pareto optimization problem using a comma-separated list of two or more objectives. An objective function, defined in Figure 4.13b, can be described by its output variable and corresponding desirable extrema refinement, which is either minimum or maximum. If obvious (which means a default configuration is hardcoded) it can be left out.

A multi-objective problem description can be converted into a single-objective optimization problem, called scalarized problem. The scalarized objective clause, refined in Figure 4.13d, describes a single objective optimization problem by combining multiple objectives using a weighted sum, based on scaling factors, described in Figure 4.13c, into a cumulated objective. It requires a priori knowledge to define a suitable weighting function. Graphically, the weighted sum problem description can be interpreted as sampling the Pareto front with tangents [Koz11]. The disadvantage of this method is that the tangents skip non-convex parts of the Pareto front.

4.4 Generic Performance Concern Processing Framework

In the previous section, we presented a language to specify performance concerns. In this section, we explain how to derive answers to language statements automatically. To implement a generic performance concern processing framework, we expanded the framework presented in [GBK14] applying several adaptations and extensions concerning generic processing algorithms, extensibility for additional metrics, and visualization. Figure 4.14 provides an overview of



(a) Optimization concern syntax

(b) ObjectiveClause syntax

(c) ScaleFactor syntax

(d) ScalarizedObjectiveClause syntax

Figure 4.13: Syntax to specify optimization concerns.

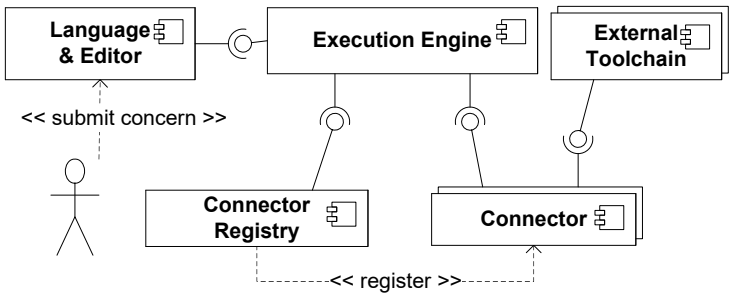


Figure 4.14: Architecture for the declarative performane analysis framework.

the components of the declarative performance analysis framework. We briefly explain their responsibilities and features.

Language and Editor The editor supports the formulation of performance concerns by auto-completion and syntax highlighting. It further provides an interface to trigger analysis of performance concerns. On a trigger event, it parses concerns and forwards them to the `Execution Engine`. Being the interface to users, this component is related to all evaluation questions concerning usability.

Execution Engine The execution engine interprets concerns and provides performance evaluation results. It contains code to process generic tasks independent of underlying performance evaluation approach. The interpretation and answering of performance concerns happens in the `Execution Engine`. Plain evaluation of performance indices can directly be forward to a connector implementation which is capable of returning the results. Usually, performance evaluation approaches do not provide support for composite, complex, and high-level concerns. To nevertheless be able to answer them, our `Execution Engine` contains a set of generic algorithms that contain a mapping to basic performance metrics processing. This component is related to the research questions on what can be generically answered and how to generalize it. We discuss generic processing algorithms for the presented concerns and appropriate result visualization.

Connectors to External Toolchains and Registry Performance concerns can be answered based on various performance evaluation approaches. Connector implementations bridge generic performance evaluation concerns to processing using external toolchains. The connector registry enables to register and de-register connectors using OSGI arbitrarily. The connector concept allows for flexibility of performance evaluation functions.

4.4.1 Generic Algorithms to Process Performance Concerns

In this section, we discuss generic algorithms of performance concern evaluation implemented at the `Execution Engine`. The sequence diagram depicted in Figure 4.15 shows the general procedure of processing performance-related concerns. It focuses on the interaction between framework, connector, and external performance evaluation toolchain. It distinguishes phases that are independent of performance evaluation approach and phases that are specific to the evaluation method.

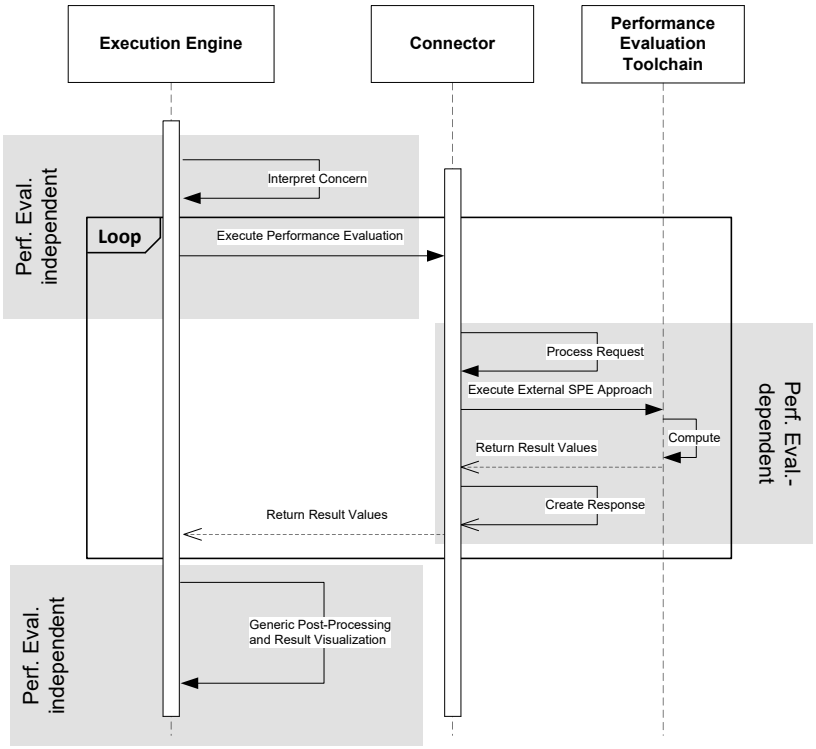


Figure 4.15: Processing of performance concerns.

In the following, we explain the processing of elementary performance metrics, what-if concerns, evaluation of SLA conformance, automated parametrization of SLAs, to the analysis of multiple system variations including resource efficiency optimization under SLA constraints. All concerns can be answered based on performance indices independent of how they have been derived.

Algorithm 4.1: Processing of what-if concerns.

Input: Current system configuration $\boldsymbol{c}$
Input: Adaptation and variation operations $\boldsymbol{a} = a_1 \ldots a_n$
Input: Metrics $\boldsymbol{m} = m_1 \ldots m_n$ to be evaluated
Input: Boolean option $delta$ where true reflects `WHAT_DIFFERS_IF`
Output: What-if analysis result $\boldsymbol{r}$

```
1  r ← new Map<Configuration, Map<Metric, Value(s)»   // result map
2  c_set ← ∅                          // derive configurations to compare
3  c_set.add (c)
4  for i = 1 to n do
5      foreach c' ∈ c_set do
6          c_set.add (a_i(c'))                        // add adapted systems
7  foreach c' ∈ c_set do
8      r.put(c', runPerformanceMetricsQuery (c', m))
9  if delta then
10     for i = 1 to m do
11         d ← diff (r_1.get (m_i),r_2.get (m_i)
12         if d > threshold then
13             r.put (m', d))
14 return r
```

Processing of Elementary Performance Metrics

Apparently, the complexity of querying elementary performance metrics is manageable. Despite, we present in more detail as we refer to it when processing more complex concerns. The processing involves to look-up a connector and its invocation, operations to parse the concern, preparation of the request and the setting of configuration options for the connector. Subsequent to triggering the connector, the processing is specific to the performance evaluation tooling. For now, we assume that a connector provides the desired performance

Algorithm 4.2: Checking SLA compliance for a system configuration.

Input: Service level agreement $\boldsymbol{c} = slo_1, \ldots, slo_n$
Input: System configuration $\boldsymbol{S}$ to be evaluated
Output: Conformance of system configuration $\boldsymbol{S}$ to SLA $\boldsymbol{c}$

```
1 r ← new Map<Metric, Value(s)>                    // perf eval results
2 for i = 1 to n do
3     if slo_i.metric ∉ r.keys then
                                  // if metric has not been evaulated
4         r.put (slo_i.metric, runPerformanceMetricsQuery
           (c, slo_i.metric))
5     if r.get (slo_i.metric) ≤ slo_i.threshold then
6         return false                    // quit on the first violation
7 return true
```

metrics and refer to Section 4.5 on additional details how to resolve them using measurement-based or model-based performance evaluation.

When the result is available, it is returned to the `Execution Engine` where generic post-processing operations and visualization take place. We feed performance evaluation results back to the framework and bridge to generic processing algorithms presented in the following.

Processing of What-if Concerns

Algorithm 4.1 shows how to process what-if concerns. In the beginning, we derive a set of all configurations from comparing the current system state by performing a cross product of adaptation operations (lines 2 to 6).

Then, we trigger performance analyses for all configurations based on performance metrics queries (lines 7-8). Finally, based on the Boolean $delta$ parameter, either, all values are returned (and visualized using a difference chart), or the diff between the initial configuration and variations is returned.

SLA Conformance Evaluation

The framework supports *qualitative* and *quantitative* evaluation of SLAs. The qualitative evaluation determines if all SLAs are satisfied or not. Analysis stops when the first violation of some SLO is detected. This is sufficient, for example, when exploring the design space for non-SLA-violating configurations.

Algorithm 4.3: Quantitative SLA evaluation counting violations and penalties.

Input: Service level agreement $\boldsymbol{s} = slo_1, \ldots, slo_n$
Input: system configuration $\boldsymbol{c}$ to be evaluated
Output: Penalty of system configuration $\boldsymbol{c}$ when adhering to SLA $\boldsymbol{s}$

```
1  r ← new Map<Metric, Value(s)>                    // perf eval results
2  v ← new Set<SLO>                                  // violated SLOs
3  for i = 1 to n do
4      if slo_i.metric ∉ r.keys then
                                  // if metric has not been evaulated
5          r.put (slo_i.metric, runPerformanceMetricsQuery
               (c, slo_i.metric))
6      if r.get (slo_i.metric) ≤ slo_i.threshold then
7          v.put (slo_i)
8  violation ← 0
9  foreach slo ∈ v do
10     violation += penalty (slo, r.get (slo.metric))
11 return violation
```

Quantitative evaluation can be used to determine the number of violations and penalty costs.

Algorithm 4.2 shows pseudo code for qualitative evaluation. It iterates over all referenced SLOs in the contract definition. Our implementation caches results of SLO evaluations, to perform only a single performance evaluation if multiple SLOs reference the same metric. In case of performance evaluation is required, the SLO is mapped to a performance metrics query that is executed directly afterward. The SLO evaluation is based on resulting performance metric values. The resulting performance indices are compared to the threshold of the SLO. The comparator has to handle fuzziness, single values, and sets.

Algorithm 4.3 shows pseudo code for quantitative evaluation. Instead of termination on the first violation, it collects violated SLOs and sums number of violations or penalties —dependent on the SLO's `penalty` function— at the end.

Algorithm 4.4: SLA parametrization using performance evaluation results.

Input: System configuration $\boldsymbol{c}$
Input: Offset $\boldsymbol{o} = o_1, ...o_n$, o_i=(operator, value)
Output: Service level agreement $\boldsymbol{s} = slo_1, \ldots, slo_n$ parmeterized with minimal thresholds that can be satisfied by a system configuration $\boldsymbol{c}$

```
1 r ← new Map<Metric, Value(s)>                    // perf eval results
2 foreach slo ∈ s do
3     if slo_i.metric ∉ r.keys then
                                  // if metric has not been evaulated
4         r.put (slo.metric, runPerformanceMetricsQuery (c, slo.metric))
5 foreach slo ∈ s do
6     slo.threshold ← r.get(slo.metric) o.operator o.value
7 return s
```

SLA Threshold Parametrization

SLA negotiation impacts business decisions. Being able to provide guarantees on lower thresholds than competitors brings the own company into a favorable business position. The idea is to orient SLAs on what the system only just can satisfy. This happens by parameterizing SLO thresholds using performance indices. To consider fluctuations, we allow specifying multiplicative and additive offsets. Algorithm 4.4 describes the parametrization of SLA based on performance evaluation results. It can be used to update SLAs when there has been a software improvement, the underlying hardware changes, or when an SLA has often been violated.

System Configuration Optimization and Upper Usage Bounds for SLA Constrained Systems

System configuration optimization and upper bounds for usage behavior in constrained systems are popular performance-related concerns [Mar+10].

Solution processes to optimization problems can be specified at the framework level. Based on previous language formalization, optimization in performance engineering can be reduced to generic optimization problems. To solve such problems efficiently, we can refer to established approaches from mathematics and operations research. In general, one aims at investigating a

Algorithm 4.5: Processing of optimization concerns.

Input: System configuration to be evaluated $\boldsymbol{S}$
Input: Degrees of Freedom $\boldsymbol{DoF} = d_1, ..., d_n$
Input: Set of objectives $\boldsymbol{o} = o_1, ...o_m$, where each objective is a tuple of cost function $f_i(\boldsymbol{S}) \rightarrow \mathbb{R}$ and desired extrema to be optimizted for $\{max|min\}$
Input: OPTIONAL: Service level agreement contract $\boldsymbol{c}$
Input: OPTIONAL: Non-functional processing constraint $\boldsymbol{constraint}$
Output: A set of configurations $\boldsymbol{configs}$ that that satisfies objectives $\boldsymbol{o}$ in a cost optimal way

```
1  if |objectives| == 1 then
       // single objective optimization
2      objective ← o.get (0);
3      if constrained by SLA then
4          ▷ cost-weighted breadth first search;
5      else if objective function is linear then
6          ▷ return S' paremeterized using extrema values for DoFs ;
7      else
8          ▷ return results of expanded NSGA-II(S, DoF, o, constraint)
             execution ;
9  else
       // multi-objective optimization
10     ▷ return results of expanded NSGA-II(S, DoF, o, constraint)
         execution ;
```

small as possible set of system configurations to provide a fast response. There are popular optimized algorithms for linear or quadratic relationships. However, the "relationship between the application performance and the resource allocation is highly non-linear and multi-dimensional as it depends on multiple factors including application architecture, system configuration, and resource demands" [Spi17].

Optimization algorithms can be classified to be either complete or approximate algorithms. Complete methods suffer from an exponential complexity in the worst-case which often leads to computation times far too high for practical purposes. Approximate methods sacrifice the guarantee of finding optimal solutions for the sake of getting sufficiently accurate solutions in a significantly reduced amount of time. For the general case, we rely on metaheuristic algorithms —which take no assumption about the performance evaluation function. Generally speaking, metaheuristics "are high-level strategies for exploring search spaces by using different methods" [BR03].

Algorithm 4.5 shows how to provide efficient processing for different optimization problems generically. The way of processing differs for optimization of single or multiple objectives.

Single objective optimization The optimization of a single objective splits into three types of processing to be differentiated:

If the objective function is differentiable, specific methods exist. For differentiable functions without constraints, we know that $x \in \nabla f(x) = 0$, which describes the null values of the derivative. If the cost function is linear, we know that its derivative is zero. Therefore, the unconstrained and linear optimization may return extrema without complex calculations (cf. Algorithm 4.5 lines 5,6). Infrastructure costs where the costs of all infrastructure components sum up is an example of a linear cost function.

In the presence of constraints formulated as SLAs, the method of Lagrange multipliers optimizes a function subject to equality constraints. However, we cannot rely on the knowledge of derivatives for constraints and cost function. To allow for efficient processing, we assume cost functions and performance index functions are monotonous. Adding resources may not improve performance but does not decrease it. Adding resources does not decrease costs. We are aware that this can be a severe limitation. However, this enables to start with the cheapest configuration and then iteratively increase resources of the configuration until the SLA is satisfied using a cost-weighted breadth-first search (cf. Algorithm 4.5 lines 3 and 4).

For scalarized optimization, we cannot estimate the null values of the cost function derivate. As a general fallback solution, one may apply metaheuristic algorithms to solve the single objective problem [BR03] Alternatively, one may refer to the costly naive approach which is to evaluate the metrics for all configurations and then evaluate constraints.

Multi objective optimization The optimization of multiple objectives is a highly researched field. Evolutionary methods are the superior and most commonly used multi-objective population-based metaheuristic [Koz11]. Based on the literature [Koz11; CDJ10], the NSGA-II algorithm has proven to be superior regarding accuracy and detection quality. PerOpterix provides an adaptation of the algorithm in a generic quality optimization framework for component-based system models [KKR11]. [Swo16] describes how to map processing of optimization problems to performance metric concerns.

Optimization under Quality of Service Constraints Quality of service constraints —commonly specified using SLAs— reduce the search space of the optimization problem. Therefore they should not only be considered in a post-processing step but during optimization to speed up optimization. [KNR11] showed that constraint handling is beneficial when having strict quality bounds. They discard candidates that are infeasible in the reproduction step of the genetic algorithm (also called death-penalty method [Coe02]). Their extensions to NSGA II to support quality of service constraints is applicable to any other evolutionary optimization technique. Their adaptation to consider quality constraints finds solutions in the interesting regions of the objective space in average 25.6% faster than the unconstrained approach.

More sophisticated methods for constraint handling for optimization approaches have been suggested which could be integrated to improve the generic framework further. For the sake of conciseness, we list available methods and refer to related literature, such as [Deb01], for details. [Deb01] classifies constraint handling into methods (i) based on preserving feasibility of solutions, (ii) based on penalty functions, (iii) biasing feasible over infeasible solutions, (iv) based on decoders, and (v) hybrid approaches. Approaches based on penalty functions apply an additive penalty term or penalty multiplier to transform a constrained optimization problem into an unconstrained problem. Penalty methods can be categorized into (i) death penalty, (ii) static penalties, (iii) dynamic penalties, (iv) annealing penalties, (v) adaptive penalties, and (vi) segregated genetic algorithms.

Parametrization of Optimization Algorithms Genetic algorithms can be parameterized to trade off accuracy and time-to-result. Our declarative approach offers to name preferences based on constraints such as *fast* or *accurate*. For genetic algorithms, we can map the processing constraints to the number of iterations and the population size. The time-to-result depends, besides processing time for a single run, on the number of performance evaluations which can be commonly derived by multiplying both variables. For NSGA-II the number of evaluations has to be divided by two as only a half of the population is investigated during each iteration. A popular ratio is to use a population size that is ten times the dimensionality of the search space [Sto96]. However, as search space grows exponentially, the ratio should be adapted [CMB15]. While it is obvious that parametrization can be derived from constraints, the concrete mapping of constraints to parametrization should be supported by extensive experiments.

4.4.2 Visualization of Performance Evaluation Results

Awareness of application performance can be derived through various quantitative analyses, model-based, and measurement-based evaluation approaches. A suitable visualization supports the understanding of application performance. Usually, analysis tools include a single type of visualization or no visualization at all. Our declarative approach allows to generalize performance analysis visualization and reuse it for multiple performance evaluation techniques.

In addition to provide metrics as plain data, there can be a multitude of visualizations helping to understand problems when reporting to stakeholders. In [Wal+16b], we propose to decouple the result visualization from the performance evaluation methodology using the Performance VisualizatiOn (PAVO) framework which will be presented in the following. This decoupling is very relevant to compare different performance evaluation results within the same interface but also to validate performance models. The relevance and usefulness are underlined by the fact that [Kro+16] developed a similar library called PET in parallel. However, PET currently supports only a subset of the visualizations of PAVO.

PAVO encapsulates the complexity of visualizations into a ready to use component. Dependent on the result type, different visualizations can be drawn. For example, series of continuous values can be represented as line charts while discrete values should not be connected and can be shown for example as a bar chart. In general, there are multiple alternative visualizations, e.g., continuous and discrete series both could be depicted as a box plot.

Performance Visualization Library Design

The goal of PAVO is to provide a *generic* visualization of quantitative performance evaluations for all stages software life-cycle stages. To benefit from PAVO visualizations, a performance evaluation approach can be integrated into the declarative performance analysis framework or trigger PAVO as a standalone. To perform the latter, one has to transform the results into the result data format and pass it to the visualization interface. Then the user receives a visualization GUI. On performance result retrieval, PAVO decides on the visualization type. Currently supported visualization types include scatter plot, line chart, bar chart, box plot, pie chart, difference chart, histogram, heat map, and bubble chart. Moreover, PAVO provides the following features

- **Automated Selection** Based on the passed result, PAVO chooses one or more suitable diagram types for the visualization.
- **Slicing** PAVO subdivides the visualization in multiple diagrams if the result does not suit to be depicted in a single diagram.
- **Diagram Switch** In case the user prefers a different visualization, the user can switch diagram types within the GUI. PAVO filters diagram types to the ones applicable for the given result.
- **Updates** The visualization of an online result updates itself on receiving new values which enables integration into performance dashboards.
- **Enrichment** Where applicable, PAVO provides various enrichments for basic diagrams that can be turned on and off. Figures can be enriched using derived values like minimum, maximum, and mean. In case the result is equipped with confidence intervals, they are displayed as error bars.
- **Export** PAVO provides a button to export the raw data or images of the selected chart in various data formats.

PAVO receives a result of a quantitative performance analysis and returns a visualization GUI. The basic structure follows the model-view-controller pattern, as depicted in Figure 4.16. The `Controller` contains the `VisualizationSelector` component which controls instances of the model elements `VisualizationType` and `ResultModel`. The `View` component contains the view logic. To update the visualization of changing results, e.g., from a live system, we apply the observer pattern.

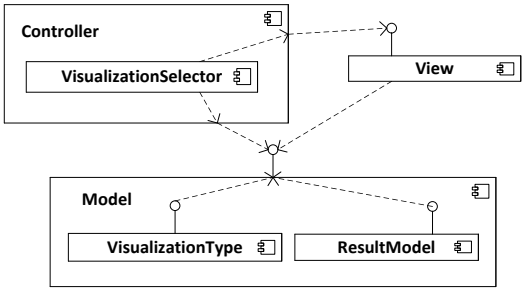


Figure 4.16: Visualization library architecture based on the model-view-controler design pattern

Result Model The choice of visualization is based on the resulting model for which we set up two requirements. PAVO requires a reduced model easily to adopt and meaningful to decide between different visualization types. Based on these requirements, we decided against reusing existing models, e.g., Structured Metrics Meta-Model (SMM). SMM's current version 1.1.1 provides no differentiation if the measures are continuous or discrete which is relevant for our visualization library. Moreover, we considered its implementation to be too heavyweight for our application scenario. Besides this, our model is compatible with results yielded by SMM. Figure 4.17 presents the result meta-model. A `ResultContainer` contains possible multiple results. `AbstractQuantitativeResult` describes the surveyed element, metric, and statistic type. It subdivides into `ValuesResult` and `SeriesResult`. Series can be continuous or discrete (`isContinuous` set to false).

Visualization Types All visualization types implement the `IVisualizationType` interface which requires two methods. The `visualize` method creates for a result the corresponding visualization. The `canVisualize` is used to select applicable visualization types that can be chosen from the drop-down menu. For, example a pie chart requires two or more elements of `ValueResult` with the same metric. A line chart requires one or more `AbstractQuantitativeResult` that are either continuous `SeriesResult` or `ValueResult`. For bar charts, there is no restriction on `SeriesResult` to be continuous. Besides these two methods, the interface enables to specify various enrichment functionality, all following an identical construction. If the `supports` method returns true, PAVO creates a radio button within the GUI. A `draw` method enriches the visualization accordingly. The same also applies for other enrichment features like minimum, maximum, and mean. The `IVisualizationType` provides empty default im-

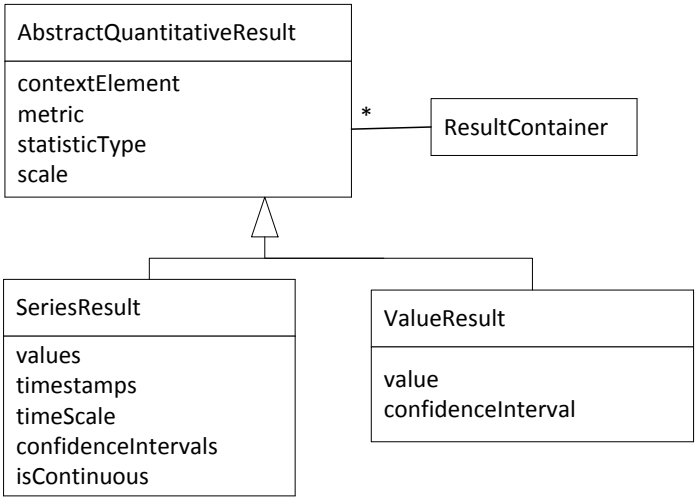


Figure 4.17: Quantitative performance evaluation result meta-model.

plementations of the enrichment methods so that the developer has only to specify the applicable ones.

Controller The controller subdivides the elements within the passed result container according to metrics. PAVO creates one visualization tab per metric. The selection algorithm, located in `VisualizationSelector`, walks for each subset through all visualization alternatives and checks whether they can be applied or not using the `canVisualize` method of visualization types. So far, we randomly chose one of the applicable visualization types for initial visualization. For the future, we intend to implement a more sophisticated selection that considers data distribution. For example, box plots provide better insights for asymmetric distributions with outliers than for normal distributions.

View For the `View` component, our implementation uses JFreeChart to draw diagrams [Gil02]. Among the common Java chart libraries, JFreeChart offers substantially more features and documentation compared to, e.g., JOpenChart or Chart2D.

Integration into the Declarative Performance Engineering Framework

The DPE vision [Wal+16a] aims to support performance analyses in all stages of performance engineering. To support visualization of performance concern re-

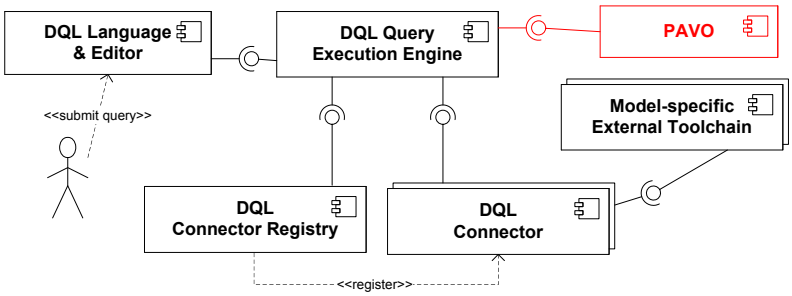


Figure 4.18: Integration of result visualization into the DPE architecture.

sults, we integrated the PAVO library into DQL. Due to PAVO's modular design, the integration caused low overhead. On result retrieval, the `Query Execution Engine` forwards results to PAVO as depicted in the component diagram in Figure 4.18. Exemplary, Figure 4.19 shows a visualization for measurements triggered by DQL using a JPetstore instantiation.

Further Visualization Types

The future development of the library will focus on integrating additional visualization approaches. Visualizations in performance engineering include several graph-based and chart-based approaches. Graph-based approaches visualize structures. These can be obtained from execution traces. Examples include the ExplorViz [FRH15] approach to visualize software landscapes using a city metaphor. ExploreViz visualizes the whole system while charts usually focus on a particular aspect. Chart-based visualizations for quantitative analyses are included in several APM and model-based analysis tools.

To detect and understand performance regressions, control charts [Ngu+14] or differential flame graphs [BPG15] can be applied. Differential flame graphs visualize the disparity between the performance profile of two execution traces. Kiviat graphs [Mil+95] may help to integrate multiple metrics into a single visualization. Utilizations can enrich traffic intensity patterns overlaid on physical structure maps. For example, [Fow+14] represent the system topology by a canonical map and superimposed aggregated traffic activity in the form of heat maps. We assume that innovative views and reports combining multiple types of visualizations may further improve the understanding of performance.

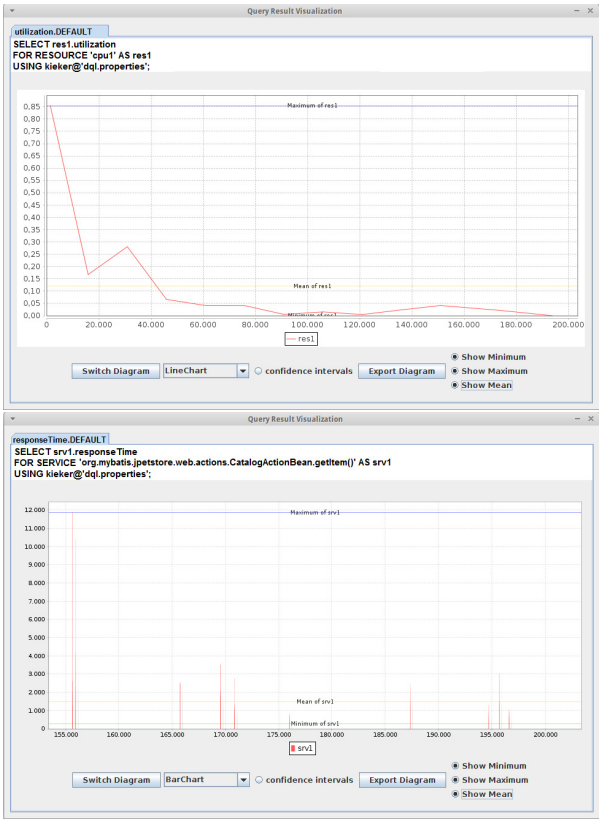


Figure 4.19: Performance analysis result visualization using PAVO

4.5 Integration of Performance Evaluation Approaches

To answer performance concerns using our declarative performance analysis framework, existing performance evaluation toolchains need to be connected. Appending a performance evaluation approach to the declarative performance analysis framework happens by contributing a connector implementation. To this end, connectors have to implement interfaces that specify interactions with the domain model. This puts the connected performance evaluation approaches into the position to benefit from reusing the concern language and a set of generic processing algorithms.

To demonstrate our approach, we integrated representative open source tools for measurement-based and model-based performance evaluation available online. Figure 4.20 provides a high-level overview of how concerns can be answered. The demonstration of integrating measurement-based performance

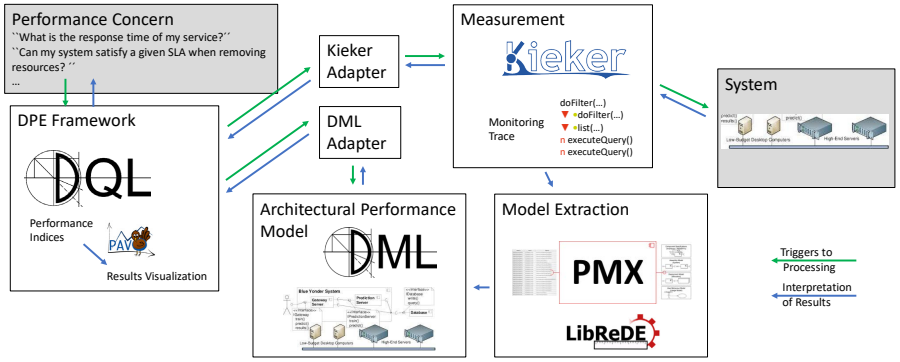


Figure 4.20: Overview of tools for declarative performance engineering (from [Wal+18]).

evaluation is based on the Kieker monitoring framework [HWH12]. The exemplary integration of model-based performance evaluation is based on the Descartes Modeling Language (DML) [Hub+17]. Besides inspecting performance models created manually, we contribute an automated approach called PMX capable of transforming Kieker execution traces into DML models, introduced in Chapter 5. In general, there can be adapters providing multiple diverse performance evaluation approaches at once based on a shared terminology. For example, connectors may incorporate model-based and measurement-based analysis at once and decide based on constraints and available evaluation artifacts how to process a given concern.

Technically, a developer has to create a new OSGi Bundle and add it as OSGi Service to the run-time environment. At this, the developer may decide which features he or she is willing to support. Partial implementations can be for manifold reasons. For some evaluation methods, only a subset of metric evaluations is supported. Also, DoFs and adaptation operations can be challenging to generalize (as they are technology specific when operating on the actual system). The effort to implement interfaces depends on various factors and the underlying performance evaluation approach.

For each aspect of concern specification that is related to the system (queryable elements including its metrics and statistic types, DoF, and adaptation operations) a specialized interface exists. In contrast to [GBK14], we separate the definition concern types and interfaces. Not all types of concerns should require to implement a separate interface. Instead, we aim at reusing interfaces for multiple concern classes. Providing generic interfaces and decoupling them from concrete query classes, allows for appending additional classes of concerns in

later revisions without changing interfaces while ensuring the compatibility with existing implementations.

The declarative performance analysis framework allows for designing stateful and stateless connector implementations. We do not prescribe but want to discuss implications of both alternatives. At this, distinguishing application state and resource state is essential. Application state describes storing data to identify incoming client requests, their previous interaction details, and current context information. The resource state describes the current state of a resource and does not relate to any interaction between client and DPE system. To achieve scalable connectors, we propose that connector implementations shall be designed to have no internal state. To learn from executed performance analyses, connectors may store additional information gained from previous analyses to a system description. For example, in case of unsuitable parametrization of performance evaluation, information can be stored with the system or its model-based representation.

Update cycles for performance models and knowledge base (e.g., complete performance model extraction [Wal+17c] and agent-based updates [SWK16]) can be performed independently of statelessness of connector implementation. Instead of local data appraisal in a self-aware system, we propose to collect data globally and distribute improvements, like a repair of a systematic over- or underestimation, globally to all instances.

4.5.1 Model-Based Performance Evaluation

To demonstrate the integration of model-based performance evaluation into our declarative performance analysis framework, we selected DML. We apply DML due to efficient existing model-solvers. DML offers flexibility in solution techniques based on model-to-model transformations, e.g., to Queueing Networks (QNs) or QPNs.

Figure 4.21 illustrates the DML prediction process. Every request consists of a performance model instance, and a performance concern. The resolution step includes the selection of an appropriate service behavior abstraction, resolution of component instantiation, call paths, and parametric dependencies as well as the parametrization of model variables. Next, an appropriate solver is selected, and the model is solved to provide performance predictions. According to the performance concern, DML internally switches between two fully automated analysis approaches.

The first analysis method transforms the performance model into a QPN, utilizing the mappings proposed in [MKK11]. The resulting QPN is solved using the SimQPN simulation engine [KB06]. SimQPN utilizes discrete event

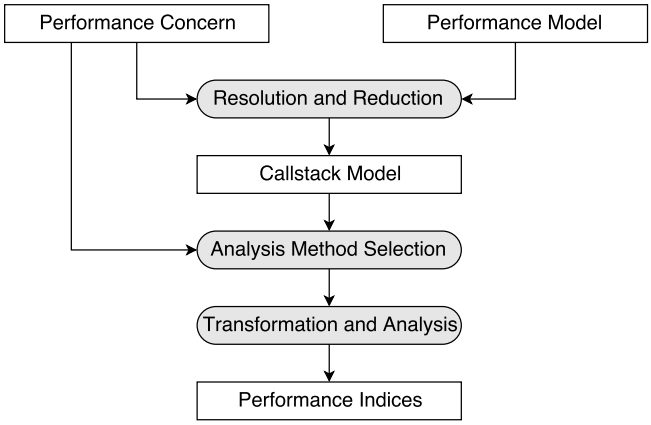


Figure 4.21: Tailored prediction process.

simulation to predict performance indices such as the utilization, response time, response time distribution and throughput of a QPN.

The second analysis method provides lower bounds for response times and upper bounds for throughput, applying bounds analysis to synchronous and asynchronous behavior. To this end, classical bounds analysis has also been extended to support asynchronous behavior [Bro14]. Since no bounds analysis requires no simulation, it can be significantly faster than QPN simulation. However, the approach can only solve models containing a single usage scenario, an open workload, and no passive resources.

During research for this thesis, we provided several extensions to the existing DML and its solution to support an expanded set of modeling features and analyses:

- **Model parametrization** We implemented solving for empirical distributions and replays. As it is not always feasible to model everything explicitly, subparts of the model can be parameterized by empirical values or distributions which also allows for trace replays. Efforts for integration included: (i) an expansion of the model transformation to QPN, and (ii) an extension to SimQPN to analyze empirical distribution.

- **Parametric dependencies** Reflection of parametric dependencies within a performance model happens by explicit modeling of input parameters and description of resource demands as a function of input parameters. DML provides so-called relationships to model dependencies between various parameters and therefore predict the impact of workload change.

In [Eis+18], we introduced novel concepts for modeling parametric dependencies:

Instance-level dependencies Instance-level dependencies describe interrelations between component instances. Modeling these dependencies for component types, as supported by existing approaches, would apply the dependency to all instances of the component.

Multiple descriptions The description of parameters using multiple independent parametric dependencies provides alternatives for run-time model parametrization. A typical use case can be, for instance, the specification of component-level dependencies and instance-level dependencies for the same variable.

Correlations as dependencies At a runnable system state, monitoring data may reveal correlations between parameters. Modeling these correlations as dependencies can be used to derive characterizations in case parameters cannot be measured. Existing parameter models can only capture strictly causal correlations as dependencies since they enforce that a parameter may only depend on prior parameters from the same call path [Bon+05; Ham09; Koz08].

- **Response time distributions** We implemented support for querying of response time distributions (the initial implementation supported only analysis of means). Efforts for integration included: (i) an expansion of the model transformation to QPN, and (ii) the expansion and interpretation of SimQPN log files.

- **System variation** We implemented the specification of options for system variation based on the DML meta-model. In contrast to the application-specific modeling of DoF and adaptations (cf. [Hub+15]), these can be reused for all models and automated. Meta-model based DoFs that have been integrated include the number of CPU cores, CPU frequency, user count and think time in closed workloads, and arrival rate in open workloads. Also, branching probabilities can now be automatically varied. Meta-model based adaptation operations that have been implemented include the migration of one component instance from one host to another.

In the future, DML is planned to be expanded by additional analysis methods. The current selection of analysis methods is based on a hardcoded decision tree which is unsuitable to maintain. To overcome this weakness, we developed a framework for automated decision support based on capability models, which we explain in Chapter 6.

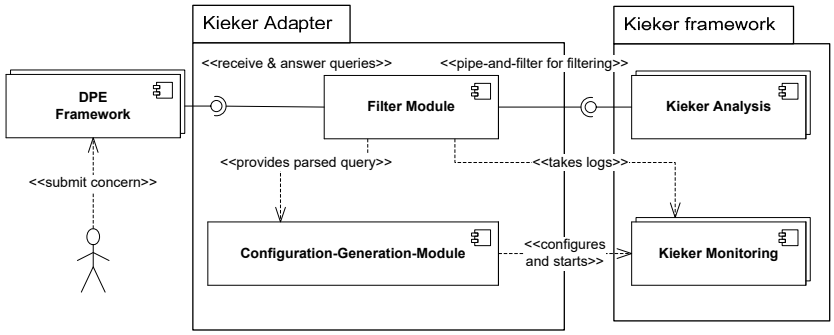


Figure 4.22: Kieker adapter architecture.

4.5.2 Measurement-Based Performance Evaluation

To demonstrate the incorporation of measurement-based performance evaluation into our declarative performance analysis framework, we selected the open source Kieker monitoring framework [HWH12]. The connector implementation presented in the following is a result of joint work with the University of Stuttgart [Blo+16] which has later been expanded to support additional metrics for sessions.

When using the connector for the Kieker monitoring tool, the processing of the performance concerns is as follows. The declarative performance analysis framework forwards a query to the connector. The connector then analyzes this query to generate a Kieker monitoring configuration tailored to the requested metrics. This configuration enables only the monitoring that is relevant for answering of the query, which reduces the monitoring overhead. Obtained data is then filtered and interpreted using a pipes and filters architecture. The filtered data is then sent back to the framework to further process and present the results. The adapter architecture, displayed in Figure 4.22, divides into two separate modules: the `Monitoring Configuration Generation` module and the `Interpretation` module.

Tailored Monitoring Configuration Generation

This module is responsible for generating the Kieker configuration files, tailored to the metrics referenced in the performance queries. Only software parts required to answer the performance concern will be instrumented. The resulting

configuration is stored in the aop.xml file, which can be used to parametrize Kieker monitoring.

For example, if the performance query requires the monitoring of the CatalogService, the automatically generated monitoring configuration will be as shown in Listing 4.1. It includes services and resources to be monitored (Line 3), and with which monitoring probe (Line 6).

```
1  <aspectj>
2  <weaver options="">
3  <include within="... .CatalogService"/>
4  </weaver>
5  <aspects>
6  <aspect name="... .OperationExecutionAspectFull"/>
7  </aspects>
8  </aspectj>
```

Listing 4.1: Example monitoring configuration.

While the result interpretation process is the same whether we apply a tailored monitoring configuration or intensely monitor the whole application, the performance overhead (and thereby investigated performance metrics) may change. In general, the monitoring overhead can be reduced when using tailored monitoring configurations. However, general statements to quantify this reduction are infeasible as the overhead depends on the characteristics of the investigated application. To provided a rough idea on the overhead, we executed the adapter with the default Kieker configuration, which allows monitoring of the whole system, as well as the tailored configuration for the provided queries. The evaluation was performed using JPetStore application that is available with the Kieker distribution. All tests were performed using the bundled load generator and were run for 200 seconds. They were conducted on a notebook with Intel Core i5-4310M, $8\,GB$ RAM, SSD and Windows 10 running Xubuntu 15.10 in a VM. On our test system response times using tailored monitoring configuration can be reduced by about 16%. Also, the file size of monitoring logs can be significantly reduced when using tailored monitoring.

Interpretation of Monitoring Data

This module reads and interprets the data from Kieker monitoring logs. It filters the data obtained from the configuration generated with Configuration-Generation-Module, but it can also filter the data generated using different the Kieker configurations. This can be useful if the monitored system does not

allow the instrumentation to change. The module performs pre-processing of the data using filters.

We employ the Kieker analysis framework and its extensible pipe-and-filter architecture. The module is essentially a set of interconnected filters that can extract the results according to a given DQL query. It converts the resulting monitoring data to declarative performance analysis framework internal data structures.

Besides querying response time of services, the adapter can derive session metrics. A session contains a sequence of multiple service requests. We allow querying end-to-end response times and processing times. The session processing time excludes delays between session requests. In other words, processing time sums up the individual response times of a session. Processing time metric can be helpful to compare measurements to model-based predictions as it excludes the time during a session spent at the load generator and network which can be challenging to model.

4.6 Concluding Remarks

In this chapter, we presented the vision of DPE separating performance concern formulation from its automated and optimized processing. According to this vision, we propose a language to formulate performance concerns independent of performance evaluation methodology. Besides querying of metrics, our language-specification integrates state-of-the-art SLA concepts into a machine-*and* human-readable DSL and enriches them with processing identifiers. We present an architecture and a framework assisting performance analysts by automating and optimizing concern processing including generic processing algorithms and visualization tailored to concerns. Further, we provide implementations to connect model-based and measurement-based performance evaluation. All code of presented tooling has been made open source and available online [1].

[1] `www.descartes.tools/dql`

Chapter 5

Automated Creation of Architectural Performance Models

During the life-cycle of a software system, performance analysts continually need to evaluate performance metrics. In addition to measurement-based approaches, system evaluation can be performed using model-based predictions. Model-based predictions allow for an exploration of alternative deployments, architectures, and configurations without the need to test them in a live system. Compared to black-box machine-learning approaches and classical stochastic performance models like queueing networks, layered queueing networks, or queueing Petri nets, architectural models also preserve architectural information alongside the performance information. Architectural performance models have been applied for various purposes, in different domains, and at different development stages in online and offline scenarios [Bru+15]. Different notations for architectural performance modeling have been proposed focusing on specific applications and analysis types. Examples of formalisms include DML [Kou+16], Palladio Component Model (PCM) [BKR09], CACTOS [16], SAMM [08], UML Marte [MAD09], and ACME [SG98]. Each of the mentioned languages focuses on a different application scenario and is therefore supported by different tools and approaches for model analysis.

While architectural performance models provide many benefits, like compositionality, it is challenging to derive them. Their construction requires measurements of performance metrics as an input to not depend on human presumptions. The construction complexity of architectural performance models is caused by many aspects including control flow extraction, the high degree of interconnection between concepts within architectural performance modeling languages [Str+16], and by the deduction of resource demands. The construction of architectural performance models based on application-specific scripts and manual operations requires significant effort which lies between weeks to months for experienced performance engineers [Hub+10; SW01; LB16; Goo+12].

In recent years, the industrial use of application performance monitoring has been increased. Also, advances have been made to automate model learning based on common execution trace information which can be derived using monitoring tools such as Application Performance Management (APM) tools or profilers. Based on this information, the model creation can be generalized and automatically learned from execution traces. Construction of model learning software has mostly been focused on creating models for individual modeling languages. Such model learning tools require a substantial initial implementation effort of years but can be reused to study multiple applications.

Existing performance model extraction tools [WK16; BK17; Bre16] are not open source and allow only for the extraction of a single formalism. This single-formalism approach requires the reimplementation and maintenance of tooling for each architectural performance modeling formalism. Even though the mentioned architectural performance modeling formalisms focus on different aspects, they have a high semantic overlap. Fundamental concepts such as compositionality, components, interfaces, and resource demands can be found in all of the mentioned languages. While formalisms have a substantial semantic overlap and extraction software is challenging to build, there is no software reuse for the extraction of different formalisms.

Challenges The availability of architectural performance models opens a broad set of performance evaluation opportunities. However, their creation faces several challenges described in the following:

- Modern software systems are subject to frequent changes of implementation, composition, and deployment. Therefore, subparts of the model may quickly become out of date and need to be updated to reflect the current system architecture and configuration. While model-driven software development (MDSD) assumes sufficient time for manual modeling, modern development paradigms like DevOps assume frequent release cycles. Frequent release and redeployment cycles limit the time and expenses that can be invested into model creation.

- The complexity of today's software systems is reflected within system models. Architectural performance models of small or medium size systems already consist of hundreds of EMF elements. Their creation in a graphical editor from scratch is connected to significant efforts. Therefore, a purely editor-based creation limits the size and complexity of analyzable systems. Architectural performance models comprise many aspects including amongst others descriptions of components and their

behavior, deployment, and resource landscape. Consequently, there is a variety of information required to build an analyzable model. Relevant aspects include control flow extraction, workload definition, and the deduction of resource demands. Further, the construction complexity of architectural performance models is caused by a high degree of interconnection between concepts within architectural performance modeling languages [Str+16]. Therefore, the construction requires a deep understanding of the modeling formalism.

- Software systems differ at many characteristics affecting modeling and prediction accuracy and may require modeling at different granularities. The granularity should be tailored to the analysis use case and system characteristics. Different system architectures, ranging from monoliths to distributed micro-services, come with particular challenges. Further, virtualization at different layers including classical Virtual Machines (VMs) and containerization affects performance behavior of systems [Spi17].

- Before model creation, one has to select a formalism suitable for the modeled system and the analysis use case. This supposes a basic understanding of strengths and weaknesses of multiple formalisms and their corresponding toolchains. Formalisms and their tooling often focus on specific analysis scenarios. Several specific analysis approaches are only implemented for a single formalism specific analysis toolchain and the benefit of models is connected to their applicable analyses.

- The application of performance models in the software life-cycle requires applying complementary analysis toolchains to benefit from an extended set of analysis techniques. However, to ensure interoperability between modeling formalism toolchains is connected to significant efforts. Reusing the same data and methodology to derive models of multiple formalisms would seamlessly provide an expanded set of model-based analysis approaches for performance analysts.

Research Questions In this chapter, we present a framework for the automated creation of architectural performance models from execution traces. The goal of this chapter is an integrated approach that may reuse the same execution traces for different performance modeling formalisms. In particular, we consider the following research questions.

- *Can the extraction of performance models be fully automated only based on APM data? What are sufficient inputs to automatically create performance models*

from scratch? What are limitations and assumptions? We provide a model extraction software that can deal with different inputs and automatically refers to a suitable resource demand estimation method.

- *What are core concepts shared among architectural performance modeling formalisms? Can we provide performance model extraction for multiple formalisms reusing a shared framework?* We propose a generic builder interface for model creation that allows to consider and implement model creation and model learning separately. Based on this separation, we provide model extraction software for different modeling formalisms.

- *To what extent can model extraction be generalized? Can we apply a single model extraction approach for systems having different architectures and different kinds of virtualization? What is the effect on prediction accuracy?* To ensure a straightforward and fair comparison, we can rely on the same generic Kieker APM tool for all types of systems.

- *How to provide the model extraction software to be easily included in performance engineering processes?* While web services for modeling and simulation already have been released, we provide the first web service for automated extraction of architectural performance models.

Chapter Outline Section 5.1 investigates architectural performance modeling formalisms for shared concepts. Section 5.2 introduces our proposed model extraction framework. Section 5.3 explains additional components required to provide model extraction as a web service. Section 5.4 discusses limitations and assumptions. Section 5.5 explains expandability of the framework. Section 5.6 concludes this chapter.

5.1 Shared Concepts Among Performance Modeling Formalisms

Our approach builds upon the observation that architectural performance modeling formalisms share equivalence classes of elements that build upon the same information for creation.

The identification of equivalence classes orients at established component-based formalisms for performance and architecture. Component-based architectural performance models share four central modeling concerns: component service specification, component assembly, deployment, and usage profile [Koz10].

Concept	Description
Component	Component descriptions include provided and required roles, and behavior descriptions for the signatures (derived from provided roles).
Interface	Interfaces contain a set of method signatures.
Method signature	Operation description that may be equipped with a method parameters and a return value.
Providing and requiring roles	Roles equip component definitions with an external access definition. They contain an interface reference. Mapping between first-class entities interface and component.
Assembly	Assembly contexts support the instantiation of the same component type in multiple parts of the system.
Host	A host is an infrastructure element where assembly components can be deployed on.
Assembly connector	Assembly connectors link a required role in of a component in a given assembly context with the provided role of a component in a different assembly context. The referenced provided role and the referenced required role refer to the same interface.
Service behavior	Service behavior including resource demands, internal control flow, and external calls. Associated to a component and signature.
Workload	Frequency and kind of system requests.

Table 5.1: Core performance annotated architectural description language (ADL) concepts.

Table 5.1 summarizes shared concepts. The component specification defines the relation between service input parameters and the performance effect of a service call. Services of components are grouped by roles. A provided role maps the internal component specification to an offered interface. A required role specifies a dependency of a component to the implementation of an interface. The component assembly defines a software system from a set of component instances. It connects required and provided roles of components. The deployment specification maps component instances to available resources, e.g., servers. The usage profile defines a set of usage scenarios that describe the investigated user behaviors. The explicit specification of usage profiles allows evaluating whether a system adheres to SLAs for a set of given user behaviors.

Example architecture modeling languages that share these concerns are DML [Kou+16], PCM [BKR09], CACTOS [16], SAMM [08], UML Marte [MAD09], and ACME [SG98]. Besides these shared core concepts, the languages differ in terminology, modeling details and connected analysis toolchain tailored to their domain and focus (e.g., run-time vs. design time). The different application areas and corresponding refinements justify not to be subsumed by each other.

The terminology of Performance Model eXtractor (PMX) builds upon the terminology used in DML, PCM, SAMM, and CACTOS. The following outlines the correspondence between the chosen terminology and ADL, if it deviates. For UML Marte (AADL) for Embedded Systems, *ClientServerPorts* with *kind = provided* and *kind = required* correspond to provided and required roles, respectively. *WorkloadBehavior* from the package *PAM_Workload* has the role of a service behavior. In ACME [GMW00], *ports* subsume both required and provided roles of a component. The role of a port in the *connector* between two components identifies a port as either required or provided. The *connectors* correspond with assembly connectors. Service behavior in a set of component *properties* that can be parameterized.

5.2 Automated Extraction of Architectural Performance Models

The creation of software that uses execution traces for model extraction causes substantial implementation efforts and requires extensive knowledge in software performance engineering. The goal is to provide a framework that enables a straightforward reuse of model extraction software for multiple architectural performance modeling formalisms relieving the developer from extraction complexity. PMX provides performance engineers with a solution that integrates established tooling for monitoring and log processing [HWH12], system model and callgraph creation [Hoo14], and resource demand estimation [Spi+15]. To

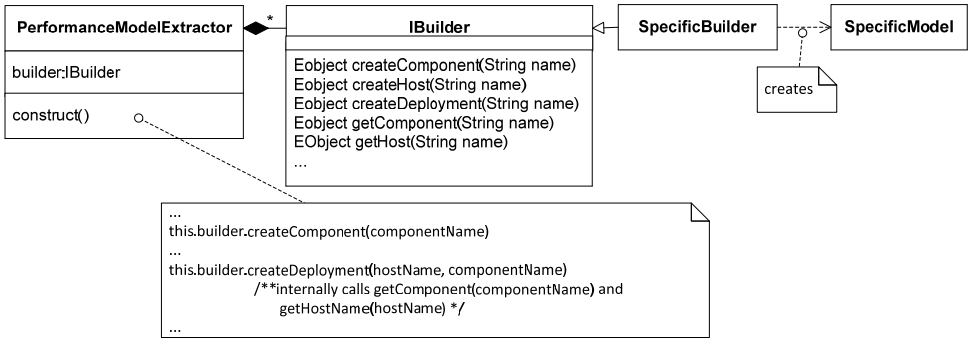


Figure 5.1: Builder pattern architecture of PMX.

enable reuse, the PMX framework approach separates the learning of shared concepts and concerns found in many architectural performance modeling formalisms, from formalism specific construction and mapping routines.

5.2.1 Two-step Model Extraction

The creation of software that uses execution traces for model extraction causes substantial efforts and requires knowledge. PMX provides performance engineers with a solution that integrates established tooling for monitoring [HWH12], log processing [Hoo14], and resource demand estimation [Spi+15].

By decoupling resource demand estimation and modeling learning methods for the concepts shared among architectural performance models, PMX reduces the effort for implementing automated performance model extraction.

To leverage PMX for model construction, developers only have to implement a model builder interface that maps language independent concepts to language-specific representations. Figure 5.1 presents a high-level view of the decoupling used for model creation. Technically, PMX employs the builder pattern [Gam+95] to decouple language-specific mappings from common model extraction concerns. The intent of the builder design pattern is to separate the construction of a complexly composed entity from its representation. Through this, the same construction process can create different representations. The construction process itself is implemented using a template method pattern [Gam+95]. The creation of architectural performance models includes linking of created sub-elements. Architectural models often separate their information within linked sub-models. For example, the specification of deployment creates pointers to the resource landscape and a component description. Linking through pointers is common for architectural

languages as sub-elements may be reused in several contexts. For example, a single component may be deployed on multiple servers and a single server may host several components. Hence, beside classical object creation, the builder interface requires getter functions to enable referencing of previously created elements.

The model extraction process follows a two-step approach of information gathering and model construction. The high-level process description of PMX is provided in Figure 5.2. We build our framework to derive the core aspects of architectural performance models like components, deployment, control flow, resource demands, and workload. In a second step, PMX uses this generic information to trigger the methods of the builder interface for the creation of an architectural performance model. While the deduction of generic information can be parallelized, the model creation order is essential to allow referencing of previously created system elements. To obtain the builder interface, the aim is to derive a pragmatic set of creation routines to minimize the overhead for builder creation. This means a pruning of shared concepts to what can be automatically derived from execution traces.

5.2.2 Deriving Required Information from Execution Traces

Based on the discussion of shared concepts, a review of which information can be automatically derived can be used to define a pragmatic builder interface. We build upon established information extraction techniques discussed in [Wal+17b]. Neglecting the explicit modeling of parametric dependencies [Kou+16], the information required to construct architectural performance models can be extracted from execution traces of a single configuration analysis run. The Kieker monitoring framework is an open-source example of a monitoring tooling that supports the collection of required monitoring data [HWH12]. There are also many commercial tools providing equivalent information, like, for example, DynaTrace or AppDynamics.

Algorithm 5.1 represents the generic process how PMX derives performance models. It receives the path to the application monitoring log files and a builder instance as input. Lines 2-7 depict the information extraction independent of targeted modeling language which is used to trigger the `construct` method in Line 8. The `construct` method, later refined in Algorithm 5.2, triggers model creation using the passed builder. The extraction of generic information is based on several filters that are connected using a pipes-and-filter architecture provided by the Kieker analysis framework [HWH12]. The processing of monitoring information is triggered in Line 4. PMX extracts a Kieker `systemModel`. It contains static system properties (names of hosts, components, interfaces,

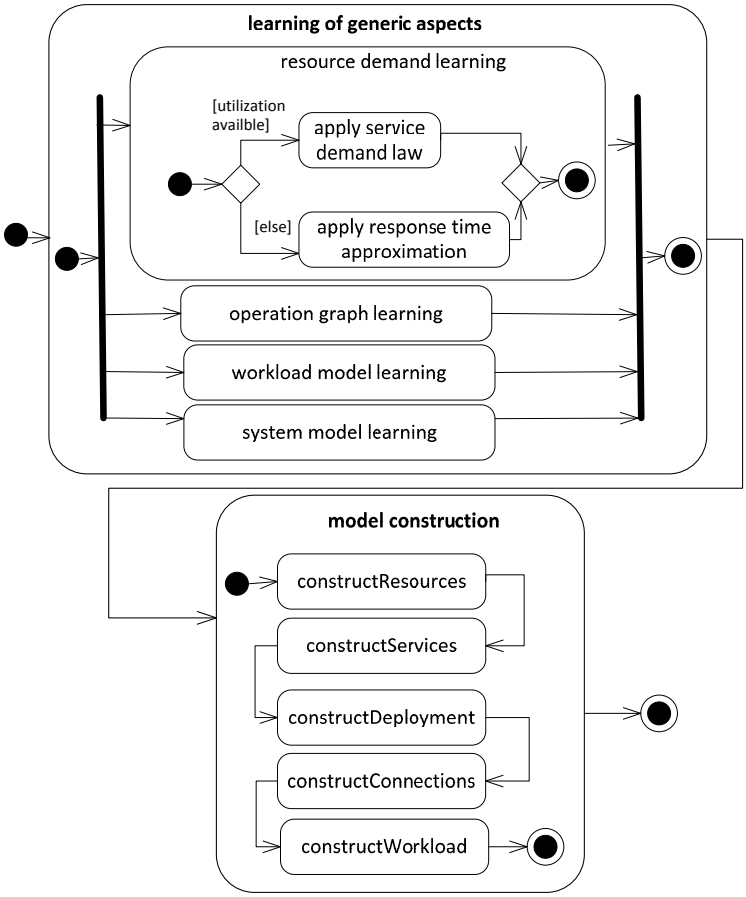


Figure 5.2: Two-step model learning process.

allocations) which can be received counting divergent identifiers within operation execution records. This requires to connect several monitoring log filters. The extraction of control flow is based on existing filters of the Kieker analysis framework that build an operation callgraph including calls weights that allow deriving call probabilities [Hoo14]. The vertices within the graph represent methods call including information about component and host. The edges within the graph represent calls to other methods. The information required to build this graph is provided by operation execution records which include method and resource identifiers, a trace id, an execution order index, and an execution stack size.

Measuring resource demands per method call causes overhead to the investigated system. To lower measurement overhead, estimation techniques

Algorithm 5.1: Extraction of architectural performance models using a generic builder.

```
   Input: logFilePath, builder, outputPath
1  APMLogs ← readAPMLogFiles(logFilePath);
2  analyzer ← compose analysis filters;
3  analyzer.analyze(APMLogs) ;
4  systemModel ← analyzer.getSystemModel();
5  operationGraph ← analyzer.getOperationGraph();
6  rds ← analyzer.getResourceDemands();
7  workload ← analyzer.getWorkload();
8  construct(systemModel, operationGraph, rds, workload, builder);
   // detailed in Algorithm 5.2
9  builder.save(outputPath)
```

can be used to derive resource demands. The extraction of resource demands filters method call arrival times and response times, and utilization per resource (if provided by a separate log type within monitoring data). Then, after filtering, the framework triggers the *Lib*rary for *Re*source *D*emand *E*stimation (LibReDE) [Spi+15] to estimate resource demands. In summary, PMX uses the maximum available information from monitoring data. In case resource utilization information is available, estimation is based on service demand law [BHK11]. Otherwise, estimation uses a response time approximation approach. To get accurate estimations using the second approach requires a calibration run without resource contention.

The PMX internal workload description stores arrival times separately for each system interface. Values can be derived filtering all operation execution records with execution stack size zero. The extracted information allows for the creation of empirical workload models as same as for aggregated probabilistic ones. More elaborate workload models, e.g., storing session information, could be easily integrated. For architectural performance models, the workload model is, besides system interface identifiers, independent of the system description and therefore easily interchangeable.

5.2.3 Generic Builder Interface

An interface could be derived creating a superset of shared concepts for all architectural performance modeling formalisms, which is cumbersome to implement. Instead, our goal is to minimize the effort for understanding and

implementation of a builder. We determine a pragmatic set required to run performance predictions orienting at what can be automatically derived from standard APM data, as described previously.

The identified core concepts lead to the *builder interface* we present in the following.

```
public interface IModelBuilder {
public EObject createHost(String hostName, int numberOfCores)\;
public EObject createComponent(String componentName)\;
public EObject createInterface(String InterfaceName)\;

public EObject createMethod(String interfaceName, Signature
    signature)\;
public EObject createAssembly(String assemblyName, String
    componentName)\;
public EObject createAllocation(String assemblyName, String
    hostName)\;
public EObject createProvidedRole(String componentName, String
    interfaceName)\;
public EObject createRequiredRole(String componentName, String
    interfaceName)\;
public EObject createServiceBehavior(String componentName, String
    methodName, List<ExternalCall> externalCalls, String
    processingResource, double meanResourceDemand)\;
public void createResourceDemand(String service)\;
public void createWorkload(HashMap<String, List<Double>> workload)\;
```

The creation methods of the interface, presented above, represent creation and connection functionality. Other interface methods reference the created objects during model creation. The implementation of creation methods for host, component, and interface requires only object instantiation. The implementation of such pure object creation methods, for a particular modeling language, is straightforward. Those core elements are referred to in the performance model composition process at multiple stages. For example, the `createMethod` references the builder interface to append previously generated signatures. Moreover, the instantiation of roles within `createProvidedRole` and `createRequiredRole` refers to previously initialized components and interface definitions. Also, the `createResourceDemand` method enriches a previously created service behavior description of a component by an internal resource demand parametrization. The resource demand is not a method parameter as it can be taken from a global resource demand map structure using the method identifier (cf. Algorithm 5.1). Further, cross-references during the model creation include, for example, the creation of connections for assemblies.

The connectAssemblies method has to enrich the connection element by pointers to the connected assemblies. At this, the function addComponentToAssembly sets the component for an assembly.

```
public EObject connectAssemblies(String providingAssemblyName,
    String requiringAssemblyName)\;
public void addComponentToAssembly(String assemblyName, String
    componentName)\;
```

The model creation process needs to store several created elements to access them later for connection and references. Access to previously created elements needs to be done based on identifiers. Hence, the interface also includes various getter functions.

```
public EObject getRole(String role)\;
public EObject getAssembly(String assemblyName)\;
public EObject getMethod(String methodName)\;
public EObject getInterface(String interfaceName)\;
public EObject getServiceBehavior(String componentName,String
    methodName)\;
```

To relieve the developer of a builder implementation from implementing all getter functions, we included an AbstractModelBuilder class into the framework which implements all getter functions of the IModelBuilder interface. It stores created elements in hash maps so that they can be referenced within the extraction algorithm. Hence, builder implementations also have to extend AbstractModelBuilder to be compatible with the framework.

Applying Generic Information to Build specific model using builder interface

Implementations of the builder interface can be used to construct architectural performance models. The construct method, detailed in Algorithm 5.2, depicts the integration of builder interface and framework by providing a skeleton model building process of PMX. Lines 1-4 apply the systemModel of the Kieker analysis framework containing static system properties to create hosts, components, interfaces, and deployment. Each of the named lines iterates over all learned elements of the particular element type and triggers the respective builder interface method for them. The creation of interconnection elements is performed using callgraph processing. The vertices within the graph represent methods (including information about component and host). The graph's edges represent external method calls. Lines 5-20 process all callgraph vertices. Lines 9 and 10 describe the creation of an assembly per edge which is enriched by its component definition. For each outgoing edge, a service call is created.

Algorithm 5.2: Application of the builder for architectural performance model generation.

```
Input: systemModel, operationGraph, resourceDemands, workload,
       builder
Output: architectural performance model (stored within the passed
        builder)
1  createHosts(systemModel, builder);
2  createComponents(systemModel, builder);
3  createInterfaces(systemModel, builder);
4  createAllocations(systemModel, builder);
5  foreach source ∈ operationGraph.vertices do
6      component ← source.component.name;
7      host ← source.host.name;
8      assembly ← component + host;
9      builder.addAssembly(assembly);
10     builder.assign(assembly, component);
11     foreach edge ∈ source.outgoingEdges do
12         target ← edge.getTargetVertice;
13         tComponent ← target.component.name;
14         tHost ← target.host.name;
15         tAssembly ← tComponent + tHost;
16         builder.assign(tAssembly, tComponent);
17         builder.connect(assembly, tAssembly);
18         calls ← outgoing.getExternalCalls();
19     rd ← resourceDemands.get(signature);
20     builder.addBehavior(component, signature, calls, host, rd);
```

Line 20 creates a service behavior description resource demands and external calls.

5.3 Model-Extraction-as-a-Service

Architectural performance models can be leveraged to explore performance properties of software systems at design-time and run-time. While design-time analysis requires a fully parameterized model, approaches for proactive model-based resource management often depend on a model skeleton as input [SWK16].

The creation of architectural performance models can be based on manual construction or scripting of model extraction code tailored to a specific application. Both require significant effort which ranges from weeks to months for experienced performance engineers [Hub+10; LB16; Goo+12].

We see a reluctance from industry about model-based analysis approaches due to modeling efforts including understanding and writing model creation scripts. We assume that building models in an editor from scratch does not scale in an industrial context due to modeling overhead.

To improve acceptance in the industry, model-based approaches need to be easy to use. Novel model extraction approaches aim at automatic extraction of models from execution traces [WK16; BK17; Bre16; Wal+17c]. However, applying research software is still challenging for layman users. Therefore, we propose Model-Extraction-as-a-Service (MEaaS). In the following, we explain the creation of model extraction web services based on our PMX [Wal+17c] tool. The web services for DML and PCM receive application execution traces and (optional) parameters specifying the number of CPU cores as input and returns an architectural performance model.

Provisioning of model extraction as a web service offers many benefits for users, developers, and maintainers of that service. Users of the web service circumvent cumbersome and error-prone set up of research software. Moreover, users do not have to update their software, as the web service always provides the latest software version. In contrast to running on multiple customer environments, maintenance needs to be done only for a single run-time environment. Researchers developing and maintaining performance model extraction tooling benefit, as we configured the web service to collect submitted execution traces as training data for model extraction. Furthermore, a web service allows investigating how external users use the software. Based on the submitted execution traces we may learn and improve automated model

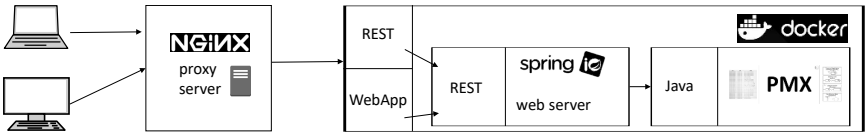


Figure 5.3: Architecture of model extraction web service.

extraction mechanisms. Finally, an increased number of users and experiments improves external validity of model extraction software.

State-of-the-Art

Related approaches can be divided into works providing modeling and simulation as services [Cay13; She+16] and approaches for model learning from application performance execution traces [WK16; BK17; Bre16; Wal+17c].

There are several attempts to lower the effort for modeling and simulation tools [Cay13; She+16]. This includes Simulation-as-a-Service [She+16] as well as providing modeling tools as web services [Cay13]. Recent works often provide services using REST interfaces and Docker container technology. None of the existing Software-as-a-Service (SaaS) solutions provides the extraction of architectural performance models based on application performance monitoring trace input.

In research, it is still quite popular to create performance models manually. While for some domains and applications this might be sufficient, we argue that this does not scale for medium and large-scale systems. Existing architectural performance model extraction tools [WK16; BK17; Bre16; Wal+17c] are either not open source or lack a publicly available and easy to use description on how to setup and run model extraction. None of the performance model extraction approaches provides a public web service.

Architecture

The goal of the presented work is to relieve the performance engineer from the complexity of model extraction *and* setup. The non-web-service version of PMX [Wal+17c] already provides performance engineers with a solution that integrates established tooling for monitoring and log processing [HWH12], system model and callgraph creation [Hoo14], and resource demand estimation [Spi+15]. The presented web service relieves the user from setup and updating.

Figure 5.3 depicts the coarse-grained architecture of our web service. Our MEaaS includes several software components, which we describe in the following.

Java extraction module The Java extraction module transforms execution traces into architectural performance models. The model extraction logic, separating generic and formalism specific parts, has been described in [Wal+17c]. Generic modeler extraction parts and formalism specific object creation routines can be packed into an executable jar file including all dependencies. The Java extraction module can be executed on the command line.

Spring I/O web server A web server allows executing model extraction within a browser. The web interface provides an interaction layer to the user to interact with the model extraction layer (PMX) using a graphical interface. The web server contains

- REST service implementation
- HTTP pages user interface
- software to trigger model extraction service
- software to return results (file download)

We decided on the Spring Framework to set up the web server, as it allows for smooth implementation of the REST API that is used to upload files, trigger the model extraction, and download. Although the framework does not impose any specific programming model, it has become popular in the Java community as an alternative to, a replacement for, or as an addition to the Enterprise JavaBeans (EJB) model.

Docker container The Docker container simplifies and speeds up deployment by packing web server and Java application into a single container. The Docker container allows deploying the application and accessing it locally within a web browser. We decided on the Docker technology, as it provides light-weight easy to deploy images.

NGINX proxy server The proxy provides an interface between external users and Docker container and organizes communication. For example, it provides a reverse proxy to do port forwarding and a secure layer for https. We decided the NGINX proxy server because it is light-weight, easy to use, and very popular. By also providing load balancing functionality, NGINX allows for an increased PMX user base.

Application

Input The model extraction requires Kieker [HWH12] application execution traces as input. To receive accurate models, the traces have to contain monitor-

ing information of a low load run (avoiding contention) or provide additional resource utilization information to improve resource demand estimation accuracy [Spi+15]. Additional parameters are optional and can be found at the web-service website.

Provided Web Services Kieker application performance execution traces and additional extraction parameters can be passed using two different services: First, we provide an interactive web interface including an upload dialog. Second, we also provide a pure REST interface suited to be used in automated processes. Passing parameters requires to expand the URL.

The result of the REST call is a zip file containing several files describing the performance model (resource landscape, software architecture, deployment, repository) as well as logging information.

To illustrate and evaluate our approach, we provide web service implementations for the DML [Hub+17] and the PCM [BKR09] modeling formalism. Both formalisms provide complementing toolchains. Palladio focuses on design-time analysis, while DML focuses on run-time scenarios.

5.4 Assumptions and Limitations

The presented model extraction software includes several assumptions and limitations. In the following, we distinguish assumptions and limitations on: (i) performance model extraction and (ii) software-as-a-service provisioning.

5.4.1 Performance Model Extraction

Providing a general model extraction has to consider several special cases. Although several years of research and development have been spent on the creation of PMX, there are several known limitations. The presence of several system characteristics may decrease performance prediction accuracy.

PMX extracts the behavior of a single system configuration. The extraction of parametric dependencies, requiring measurement on multiple configurations, is not yet supported. To the best of our knowledge, there is no approach published on how to identify parametric dependencies automatically without extensively measuring all parameter configurations. However, the knowledge of a parametric dependency allows performing focused measurements of different parameter configurations to train parametric behavior. This includes also reflecting changing resource demands, e.g., due to compiler optimization

triggered on high workloads. In particular, existing resource demand estimation methods do not consider effects of run-time optimization, e.g., auto-tuning or byte-code optimization.

PMX refers to commonly available data to create performance models. The model creation of PMX builds upon a probabilistic callgraph. If a service is called in different contexts *and* its behavior changes context wise, we derive a component behavior description that mixes behaviors. Further, preservation of an internal state cannot be captured within the derived component behavior description. In general, service behavior description, like loop counts and branching probabilities, are probabilistic. Variations in thread pool sizes and caching effects are not explicitly covered by our approach, as we cannot derive it from standard execution traces.

The model extraction assumes that the system under investigation has CPU bound processes. Processes that are bound to other resources, may decrease prediction accuracy of derived models. Learning of memory or I/O resource demands is not yet captured within the current implementation. Also, the capturing of limiting software resources has not yet been included. However, even in that case our approach still provides savings by providing an initial model.

In case performance prediction capabilities of extracted models do not meet accuracy requirements, the classical software performance engineering model refinement process can be applied, depicted in Figure 5.4 [SW01]. The termination of performance model improvement depends on the achievement of a sufficient prediction accuracy. Our current implementation requires to manually trigger this feedback loop. However, we intend that future versions automatically test accuracy of derived models. Classical software architects have to trade-off between analysis accuracy and modeling effort [Hub+10]. To ensure accurate predictions, some system aspects have to be modeled in more detail than others. Our approach cannot automatically propose a suitable model granularity and depends on the monitoring configuration. The prediction accuracy of models learned by PMX may be improved either by repeating model learning using refined measurements or by applying manual refinements to the learned model. The refinement of the monitoring configuration may include to either aggregate or exclude functions (to reduce system overhead) or to expand monitored functionality (to get a more detailed model). Besides adjustment of monitoring, manual refinement of learned models allows to include user knowledge which cannot be learned using standard monitoring information from APM tools. Candidates for refinement include, e.g., modeling of thread pools, parametric dependencies, or database behavior. The PMX

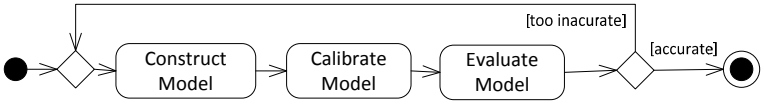


Figure 5.4: General performance model creation process.

framework implementation is limited by the shared concepts encapsulated within the builder interface and by the available input information.

5.4.2 Software-as-a-Service

We introduce the concept of MEaaS to improve the usability of model extraction software. However, the current implementation has limitations concerning scalability and security.

The online provisioning introduces a new layer of potential failures compared to local execution. In particular, the model extraction might take longer than HTTP timeouts (especially if input traces are huge). Future web service versions should apply asynchronous communication to resolve this problem. The technical solution could be based on continuous streaming using, e.g., web sockets.

In general, we do not expect network attacks shortly and rebooting the service can be easily performed using the Docker image. However, services provided online are subject to various types of attacks. In particular, there is a vulnerability of PMX for Denial-of-Service (DoS) attacks. The significant resource demand caused by a single model extraction request makes the system prone to such attacks. To cope with this security breach, we might be forced to use login mechanisms. Monitoring logs might contain sensitive information concerning customers. Later commercial use of PMX might require to add data security mechanisms. A straightforward option to improve privacy for monitoring data could be ensured running a separate service instance behind the firewall of a company.

5.5 Extensibility

In the following, we sketch how PMX can be expanded using (i) additional inputs, (ii) different monitoring tools, and (iii) additional modeling features.

Additional Input for Model Creation The model extraction of PMX may be expanded to include additional information. In general, the builder interface may

remain untouched for all extensions but the introduction of new architectural modeling constructs. For example, direct measurements of resource demands, as proposed in [BK17], could complement estimation methods if a monitoring framework enables to measure resource demands per request.

In general, PMX has limitations that stem from its limited input information. The model creation of PMX builds upon a probabilistic callgraph. This graph does not preserve whether a loop behavior has been created using a "for" or "while"-operator. PMX cannot distinguish whether a method call has been triggered by an interface call, or triggered by an event listener. Additional information (e.g., from source code) could improve the extracted model and retract limitations. Moreover, additional run-time information of underlying layers, e.g., VM operations like garbage collection [Wil+15], could be included as events of underlying layers may blur measurements and model extraction.

Monitoring Data Formats Performance models are applied to various domains, e.g., data centers, automotive systems, and embedded systems. The application domain and technical setup usually determine the application monitoring infrastructure. There exist a number of commercial and open-source APM tools that allow the capturing of execution traces within distributed software systems [Oka+16]. They usually provide the same information but use different monitoring data formats [Oka+16].

The Kieker tool natively supports monitoring of Java applications. For non-Java applications, PMX offers two ways to connect to monitoring tailored to domain-specific requirements. The Kieker monitoring tool itself has been designed to be extensible for different programming languages and domains by providing a data bridge. The Kieker Data Bridge (KDB) is designed to support a wide range of monitoring sources, allows to add monitoring to any language and to be extensible considering the means of data relay. Implementations for Perl, C#, VB6, Cobol, and IEC61131-3 languages for programmable logic controllers have been proposed. A second approach to achieve interoperability would be to apply a custom monitoring tool and transfer logging information to the Kieker data format. At this, the OPEN.xtrace can be applied as an interchange format. It enables data interoperability and exchange between APM tools and SPE approaches [Oka+16]. Performance engineers may transform their trace format to OPEN.xtrace which already provides export to the Kieker format.

New Modeling Features It might be required for some application scenarios to expand PMX to extract additional language features available only in a subset

of formalisms. Expanding the prevalent set of core modeling techniques encapsulated in our builder interface, there are modeling concepts not supported by all formalisms. Moreover, they can be integrated differently into modeling languages. A prominent example is modeling of parametric dependencies (e.g., parametric resource demands and branching probabilities). They have been integrated differently into architectural performance modeling languages.

To integrate extraction of additional modeling features, we propose to use the template method pattern [Gam+95] extending the skeleton of the `CONSTRUCT` method, deferring object creation again to builders. The creation requires to expand the builder interface. However, extensions in this direction should not require adaptations for every builder. To this end, we strongly recommend providing a default implementation not to break existing builder implementations and remain downward compatible.

Performance modeling languages may offer to model the same systems properties in various ways. For example, DML offers different granularities (black-box, coarse-grained, fine-grained) for the behavior description. This can be addressed by providing different builder implementations for the same language.

5.6 Concluding Remarks

In this chapter, we presented an approach for automated extraction of architectural performance models from execution traces that is extensible for different modeling formalisms. Our approach isolates learning for concepts shared across several formalisms from specific model creation routines. A major contribution of this work is to provide an object creation interface that generalizes over multiple formalisms and pragmatically focuses on what can be automatically extracted from execution traces. Instead of implementing the entire model extraction code, developers adopting our framework only need to implement a builder interface covering the language-specific mapping of common concepts. The implementation of our approach, called PMX, can be reused to create architectural performance models of various modeling languages. Summarizing, we see the following benefits:

- *Software reuse*: Our approach enables lw overhead reuse of model extraction software for multiple formalisms. The reuse of model extraction software relieves developers from substantial efforts for the implementation or integration and composition of libraries for the deduction of the system model, callgraph, and resource demands.

- *Interchangeability of modeling formalisms*: Interchangeability between modeling formalism is connected to significant efforts. Applying our approach, switching of formalisms only requires to implement the builder interface or to reuse an existing builder implementation.
- *Complementary use of multiple analysis tools*: Advanced analysis approaches are often implemented for a single modeling formalism. Applying complementary analysis toolchains in parallel provides an extended set of analysis techniques.
- *Comparability and benchmarking*: Enormous efforts for modeling approach interoperability cause low competition and low pressure to improve analysis toolchains. Our approach provides models of the same system using different formalisms. This enables comparison and benchmarking of different analysis toolchains based on the shared model extraction approach.

We implemented builders for the PCM and the DML formalism as part or our evaluation presented in Section 7.1. Based on these formalisms, we use different downstream analysis tools to analyze resulting models. Our evaluation discusses: (i) savings using PMX compared to the implementation of formalism specific extraction software from scratch (ii) savings compared to case study specific modeling efforts, and (iii) the accuracy of model extraction.

To improve the usability of model extraction, we explained how we provide the first publicly available web service for the extraction of architectural performance models. Based on the core model extraction software, we present additional software components to provide MEaaS. Provisioning as a web service reduces the initial and long-term efforts to derive architectural performance models for all kinds of users.

Last but not least, the automated model extraction is a crucial building block for our DPE approach to enable seamless interchangeability of measurement-based and model-based analysis.

Chapter 6

Decision Support for Performance Engineering Approaches

Performance evaluation comprises various approaches such as measurement, simulation, analytical solutions, model transformations to stochastic formalisms, algebraic solutions, fluid approximations each having specific analysis goals and capabilities. Performance evaluation strategies can be applied for various purposes and at different development stages [Bru+15]. Interoperability of evaluation techniques can be achieved by automated performance model extraction [BK17; Wal+17c], and by transformations of monitoring data [Oka+16] and performance models [Bro+15]. Integrating all aspects into a unified interface, declarative performance evaluation approaches can be applied to automate the whole process of deriving performance metrics [Wal+16a].

Performance evaluation techniques can be compared based on different criteria like analysis speed, accuracy, or system overhead. They come with strengths and limitations depending on user concerns to be answered and the characteristics of the system under consideration. While numerous alternative performance evaluation approaches exist, the choice of an appropriate performance evaluation approach and tools to solve a given performance concern still requires expert knowledge. Even the desired resolution of the data and statistic types like distribution, mean, quantiles, percentiles, or maximum, impact the choice of a suitable performance evaluation technique [Bol+06]. For example, mean value analysis can be significantly faster than simulations [SSB93]. For measurements, in-place aggregation of metrics may reduce the communication overhead. Performance evaluation approaches can be used to evaluate a variety of metrics like utilization of resources, or throughput and response times of services. The requested metrics impact the choice of a performance evaluation approach, but also the characteristics of the system under consideration or its model representation may impact the choice. For monitoring tools, there are technological restrictions on what languages and infrastructures can be monitored [Wat17]. For model-based analysis, approaches come with different

scalability regarding time-to-result and memory overhead [Bro+15; Mül+16; PC17], supported model elements [Bro+15], and restrictions to specific model structures [Bol+06; WSK15a].

To summarize, various methods, techniques, and tools for measuring, modeling, and evaluating performance properties of software systems have been proposed over the years, each with different strengths and limitations. The choice of appropriate tooling to solve a given user concern tailored to the application scenario requires expert knowledge. Due to the numerous performance evaluation approaches it is impossible to be expert for all of them. To provide guidance, case studies, as presented by [Bro+15] or [Mül+16], compare different solution approaches quantitatively considering accuracy and overhead. However, they investigate only a few scenarios, and they mostly do not provide automation. Given the complexity of selecting a solution approach, it is essential to introduce automated decision support.

Challenges An approach for automated decision support for performance evaluation approaches has to address a number of challenges. In the following, we describe these challenges:

- There is a significant amount of performance evaluation tools and approaches. Automated decision support has to allow for comparison of different kinds of performance evaluation approaches including measurement-based and model-based analysis approaches, like analytical and simulation-based solvers. Considering the pure amount of performance evaluation tools and approaches, it is challenging for a single human to be expert even for a subset of them.

- There are different performance evaluation needs. While some performance evaluation scenarios demand the most accurate performance evaluation results, other scenarios impose time-to-result constraints. Also, the decision support has to consider different kinds of metrics like response time or throughput and statistics like mean or 90th percentile.

- The evaluation of the suitability of a single evaluation approach can be associated with significant complexity. Various factors, like required metrics, statistics, constraints, and optimization attributes determine the suitability of an evaluation technique for a given scenario.

- Performance evaluation artifacts can be heterogeneous: IT systems apply diverse technology stacks, performance models are provided using different formalisms. The decision support has to consider characteristics and

limitations of the system under consideration. Hence, decision support requires and has to be based on a generic system definition that unifies characteristics of diverse technology stacks.

- There is an evolution of performance evaluation approaches and tooling. This requires to update the decision support recurringly. The decision support should allow for adjustments and extensibility without introducing a dependency to already integrated approaches or comparison attributes. Existing approaches to automate decision making for performance evaluation, e.g., [Bol+06; Bro+15], support a very limited set of analysis techniques and comparison attributes *and* do not provide an extensibility concept to integrate new approaches or comparison attributes. We see three types of evolution:
  - Addition of new solution strategy or removal: The development of new model transformations, model solvers, or measurement tools requires to update solution strategies.
  - Adaptation of solution strategies: Besides adding new approaches, decision support should allow to quickly adapt the capabilities of solution approaches to be able to track enhancements of existing approaches.
  - Addition of new evaluation criteria: It should be possible to add more comparison attributes while separately preserving the information if a solution approach is applicable.
- Run-time scenarios often imply time constraints for performance predictions. The estimation of time-to-result for model-based performance prediction is challenging. The run-time depends on various model characteristics as well as on the concrete analysis configuration. Even knowing the impact factors does not provide an estimation of time-to-result. Moreover, relevant features may differ for each performance evaluation method [Bro+15; Mül+16].

Research Questions In the last decades, a plethora of different approaches and tools for performance evaluation have been proposed (cf. Chapter 2). These approaches differ in many properties. The superordinate research question is how to automatically decide which performance evaluation technique to apply in a given scenario. In this chapter, we address the following subquestions:

- *Which factors impact the choice of a performance evaluation approach? Is there a difference in recommending model-based and measurement-based analysis*

approaches? We discuss inputs for decision support that allow for a scenario aware choice and formalize them by a description of system and performance concern.

- *How to design decision support systems in order to put them in position to cope with the evolution of performance evaluation tooling?* We propose a software architecture decoupling the decision process into solution approach capability models and a generic decision engine. In contrast to state-of-the-art tree-based approaches, this separation of concerns supports adaption of decision support on evolution scenarios.

- *How to describe the capabilities of a performance evaluation approach? Which capabilities are reasonable to model? How to model overlap of related performance evaluation techniques?* We propose a capability meta-model to describe functional and non-functional aspects of performance evaluation approaches and tooling.

- *How to predict time-to-result for model-based performance evaluation?* We describe a machine-learning-based approach that can estimate the time-to-result and improves prediction accuracy when predicting analysis time for models previously analyzed using other configurations.

Chapter Outline We describe a software framework architecture for decision support that is based on a decomposition into a decision engine and a capability model in Section 6.1. Section 6.2 proposes the corresponding capability meta-model to describe performance engineering approaches. Then, Section 6.3 explains the generic decision algorithms within the framework that recommend based on registered capability models. In Section 6.4, we explain our machine-learning-based approach to time-to-result prediction. Finally, Section 6.5 concludes this chapter.

6.1 Recommendation Architecture

The contribution presented in this section is a methodology for automated decision support for performance evaluation approaches that enables to select an appropriate approach for a given system and user concern automatically. We propose an architecture decoupling the complexity of the approach selection into capability models of performance evaluation approaches and a generic decision engine. The integration or modification of solution techniques requires no knowledge about previously integrated performance evaluation

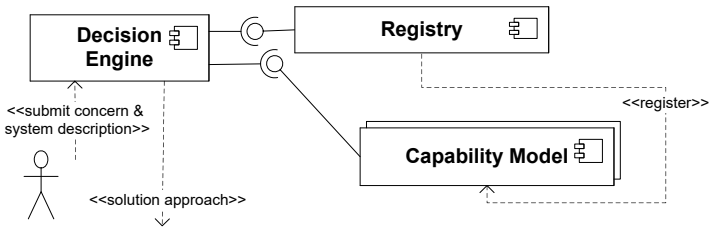


Figure 6.1: Decision architecture.

approaches. Besides automation, the benefit of our capability model concept is that it supports extensibility concerning solution approaches and comparison attributes.

The choice of an appropriate solution approach to solve a given performance concern requires expert knowledge. The goal is to provide a framework that automates the decision based on configurable concerns supporting measurement-based and model-based analysis while being extensible with respect to solution approaches and comparison metrics. Our methodology provides a solution to filter applicable approaches and optimize for given concerns automatically.

Figure 6.1 presents the coarse-grained architecture of our decision framework. The decision support builds upon capability models of different performance evaluation approaches and a generic decision engine. Instances of the capability meta-model represent analysis approaches like measurement, simulation, or analytical solvers. They have to be registered in a central registry.

To propose a solution approach, the decision engine receives a performance concern and a description of the analyzed system as input. The required inputs are in coherence with the declarative approach presented in Chapter 4, even if we introduce a brief tuple notation slightly different from performance query notation.

`Concern` We define a `Concern` as a tuple of element, metric, statistic and optional accuracy constraints $(const)$ and cost types to optimize for (opt):

$$cn = \{element, metric, statistic, const?, opt?\}.$$

An exemplary concern tuple is given by [service, response time, sample, accuracy = high, time-to-result]. This tuple states that the fastest solution approach providing a sample of service response time at a high accuracy should be chosen.

System A `System` tuple describes the system under consideration. It may contain sets of applied languages $(language)$ and middleware technologies $(middleware)$, and an architectural system description $(model)$. The system description is given by the tuple:

$$system = \{language*, middleware*, model?\}.$$

An exemplary system tuple is given by [(Java, Scala), (JBoss), /application/myModel.properties]. The tuple states that the application uses Java and Scala code, runs on a JBoss application server middleware, and the relative path to an architectural model of the system is given by "/application/myModel.properties".

Our system tuple offers alternative descriptions in parallel to evaluate applicability for different kinds of approaches. The system description for model-based analysis is specified by its model (denoted by its path in the file system) and meta-model (indicated either by file ending or file content).

The automated extraction of system descriptions including technologies required to evaluate measurement tools is more challenging. Currently, there are neither standardized representations nor interfaces to derive information automatically. Hence, if a scenario cannot answer if a limitation applies, our decision engine points the question to the user. While nowadays the description of applied technologies has to be put together by hand, self-describing systems might automatically provide a model of their applied technologies in the future. Evaluating its inputs, the engine walks through all registered solution approach capability models, filters and rates according to evaluation scenario specific requirements, and selects an appropriate solution strategy. At first, the decision engine decides for a given performance engineering approach if it provides the required functional capabilities to process the concern for the given system settings. In case multiple solution approaches are capable of providing the same required metrics and statistics, the decision will be made based on non-functional requirements specified in the `Concern`.

6.2 Solution Strategy Capabilities Meta-Model

In the following, we present a meta-model for performance engineering tools and approaches, enabling the description of their capabilities. This enables a performance engineer to model what is provided by a solution strategy. The implementation of our capability model is based on the Eclipse Modeling Framework (EMF) [Ste+09]. This allows for the instantiation of solution strategy

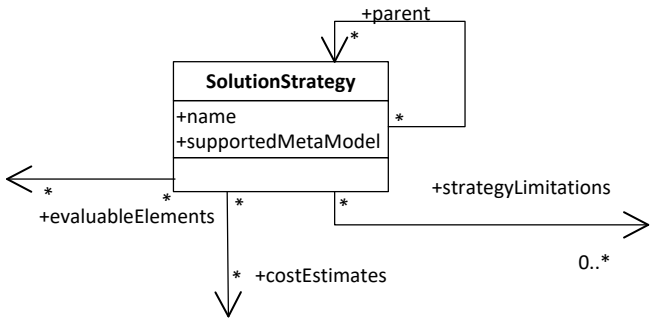


Figure 6.2: Solution strategy capability model root.

models by performance engineering experts using an automatically generated graphical editor [Ste+09].

Figure 6.2 presents the root of our solution strategy capability meta-model. The root points to submodels that specify what can be derived at which accuracy, as well as the costs and limitations when applying the considered approach. The comparison of solution approaches requires the specification of a common terminology. We cannot directly compare a stochastic solver for QN to a native simulator of an architectural performance model. They can only be compared only if there is a transformation from the architectural model to the stochastic QN solver. The shared meta-model would be the architectural model. To reflect this, a solution strategy links to a `supportedMetaModel` to represent the linked terminology meta-model, which can be either an architectural description or a stochastic model.

Solution approaches can be classified into groups according to their capabilities. For example, when multiple solution approaches depend on the same model transformation, an abstract model may be employed to describe their common aspects. Another example for an analysis group with high overlap is the simulation of performance models. Except for few limitations and improved time-to-result, parallel simulation provides the same capabilities like sequential simulation [WSK15a].

Our meta-model reflects similarities of models by allowing to define abstract approach capability models from which concrete models may be derived. Inheritance provides benefits if the capabilities of two solution strategies differ only in a few capabilities. In such cases, the knowledge has to be persisted only once in the form of an abstract capability model. This enables the reuse of capability specifications for multiple tools. A capability model may be defined using a hierarchical structure. Instances of an abstract capability model should

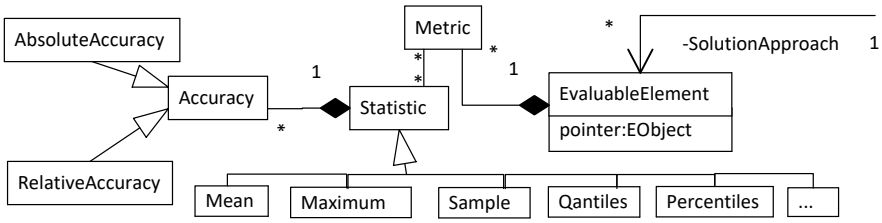


Figure 6.3: Functional capabilities and accuracy meta-model.

include sufficient information for decision making so that a developer may specify a method's capability model just by inheritance. Abstract capability models enable recommendations considering approaches without the need for a concrete tool implementation.

6.2.1 Evaluable Elements

Based on the meta-model definition in the solution strategy, there can be multiple element types for which one may request metrics. To model this, we define `evaluableElements` as scenario model elements for which the solution strategy yields analysis results. `EvaluableElements`, described in Figure 6.3, define functional capabilities of a solution approach and may be extended by a description of accuracy. Example types of `EvaluableElement` include service or resources like CPU or HDD in a monitored system or an architectural performance model such as PCM or DML. Considering other system abstractions like QNs or QPNs, example types of `EvaluableElement` are queues, places, or transitions. To specify the type, an `EvaluableElement` contains a pointer to define the type element of the connected meta-model terminology. An `EvaluableElement` contains a set of metrics. We modeled an initial set of metrics that implements the abstract class metric including, e.g., response time, throughput for services and utilization of resources. Each metric definition has to contain at least one statistic. Statistics refine supported ways to determine a metric by the given solution strategy. We integrated an initial set of common statistics, like mean, sample, maximum, quantiles, and percentiles. This set can be arbitrarily extended by implementing the `Statistic` interface. The statistic definition may be linked to a specification of the solution accuracy. The instantiations of the abstract accuracy class allow to model a maximum absolute or relative deviation from ground truth data. This allows modeling

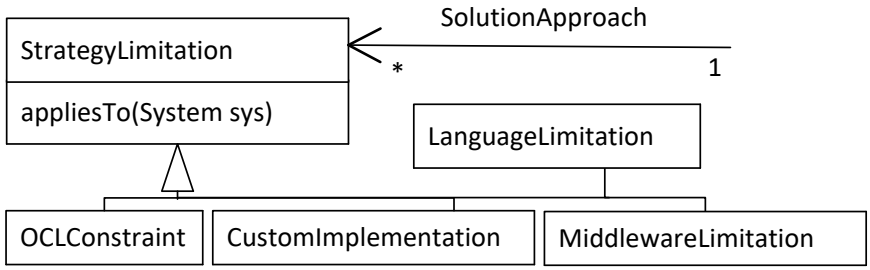


Figure 6.4: Solution strategy limitations meta-model.

different stopping criteria for simulation, as well as approximative solvers in general.

6.2.2 Limitation Modeling

The applicability of solution strategies can be limited by several constraints. Figure 6.4 shows the modeling concepts to reflect this in the capability meta-model. It depends on the characteristics of the given system description if a constraint applies. For example: While some QN models are in product form, others are not and therefore cannot be analyzed using specialized product form algorithms. To allow for a generic decision engine, the evaluation mechanism has to be transcribed in the limitation description. Therefore, `StrategyLimitations` have to implement the `boolean appliesTo(System sys)` interface evaluating applicability based on a passed `System` description.

The `System` is basically a container that may contain different kinds of system descriptions. To evaluate limitations for model-based analysis, the passed `System` has to contain the performance model. Then solution strategy limitations can be defined using Object Constraint Language (OCL) [Ste+09]. OCL allows to declaratively query constraints in objects. In particular, OCL allows to define constraints on invariants pre- and postconditions, initial & derived values, definition of external attributes or operations, body definition, and Guards for transitions. Exemplary constraints for model-based analysis are on applicable input models (e.g., for product form solutions) (described e.g., in [Bol+06; Bro+15]) or limitations of model-transformations (described e.g., in [Bro+15]). Besides using OCL, the evaluation of applicability can also be based on `CustomImplementation` to tailor a more efficient evaluation if a limitation applies.

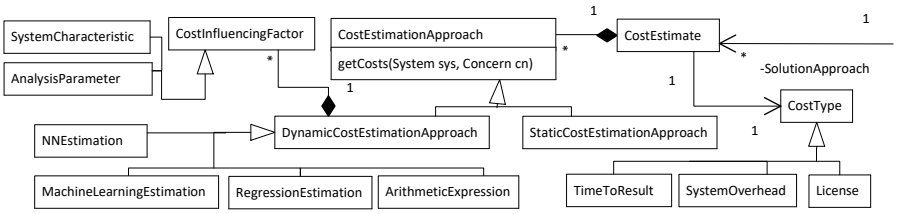


Figure 6.5: Cost meta-model for performance evaluation approaches.

While limitation modeling concepts can be transferred, measurement tools are usually limited to specific technologies. To model supported languages and technologies, `LanguageLimitation` and `MiddlewareLimitation` contain a textual list of supported technologies. This allows, for example, modeling a limitation to Java technology for a monitoring tool. To evaluate if provided technologies match to a `System` description, a matching of supported tool technologies to system capabilities can be performed.

In the future, we envision a connected database of limitations storing limitations that occur frequently. This database may contain predefined `StrategyLimitations` for solution strategies in association with an architectural meta-model. This database will be linked to our capability model by a reference.

6.2.3 Cost Modeling

Solution strategies differ in several cost types like time-to-result or system overhead, as depicted in our cost meta-model in Figure 6.5. Which cost type is the most relevant depends on the specific application scenario. Therefore, a `CostEstimation` has a dedicated `CostType`. For model-based analysis, this can be, e.g., `TimeToResult`. For measurement-based analysis, `SystemOverhead` and `License` costs are the common comparison and optimization criteria while for model-based approaches system overhead can be considered to be zero.

Costs can either be static (e.g., fixed license costs, time-to-result always high or low) or dependent on the system and analysis configuration. Static costs can be used as a simplification when complex relationships are not known. In the more general case, the costs may depend on `CostInfluencingFactors`, which may depend either on the system characteristic or the analysis configuration. A `SystemCharacteristic` defines a property of the system or model to be analyzed, e.g., instantiation type or count of elements, recursion depth, or a composed metric. An `AnalysisParameter` describes a parameter of the analysis set-

ting of the given solution strategy. This can be, for example, the accuracy configuration for a simulation run, as simulation runs parameterized to be more accurate are expected to require more run-time to return the result. `CostEstimation` offers a method to predict the cost for the analysis of a given `System` and `Concern` based on the interpretation of cost influencing factors. The abstract concept of system- and concern-aware `DynamicCostEstimationApproach` can be implemented by concrete prediction approaches. This enables the integration of arithmetic expressions capturing expert knowledge and various estimation techniques, e.g., using neural networks, machine-learning approaches, or regression-based approaches.

6.3 Solution Strategy Decision Engine

Based on the capability meta-model defined in the previous section, we can build a decision engine to decide which solution strategy to use to process a given performance `Concern` for a given `System`. The recommendation is based on filtering and sorting of registered evaluation approaches.

The decision process follows a two-step approach. In Section 6.3.1, we explain how the framework selects applicable solution approaches that come into question. The decision engine decides if a given tool is able to solve a performance concern based on the tool's capability model and system description without consideration of costs. This also includes filtering of approaches based on strategy limitation evaluation on system description. Then, described in Section 6.3.2, the applicable approaches are rated according to cost and accuracy criteria as defined in the user concern. Extending the decision logic to consider constraints about speed, perturbation, and accuracy when recommending a performance engineering solution approach.

6.3.1 Filtering Applicable Approaches

Many approaches cannot be applied in a given context. The decision engine decides if a given strategy is applicable for a given `Concern` and `System` description. This includes the subsequent evaluation of `isApplicableForConcern` and `isApplicableForSystem` for each registered solution strategy. If both functions evaluate to true, the decision engine adds them to applicable evaluation strategies. Algorithm 6.1 describes the process in pseudocode.

In a first step, the `isApplicableForConcern` function, checks for matches of the evaluation element, metric, and statistic defined in the concern element. This function may determine, for example, that mean value analysis is not appli-

Algorithm 6.1: Select applicable strategies for a given system and concern.

```
Input: Solution strategies s = s_1, ..., s_n, Concern concern, System sys
Output: Solution strategies applicable for concern and system
1 applicableStrategies ← ∅
2 foreach strategy ∈ strategies do
3     if isApplicableForConcern (strategy,concern) then
                                                  // see Algorithm 6.2
4         if isApplicableForSystem (strategy,sys) then
                                                  // see Algorithm 6.3
5             applicableStrategies.add (strategy)
6 return applicableStrategies
```

cable if a probability distribution is requested or that system-level monitoring tools cannot provide application performance metrics. When provided in the concern definition, the `isApplicableForConcern` method also filters applicable approaches according to accuracy constraints. Algorithm 6.2 shows the filtering according to concern in pseudocode.

Algorithm 6.2: Check if strategy can be applied for a requested element, metric and statistic type.

```
Input: SolutionStrategy strategy, Concern cn
Output: Booelan, is applicable
1 type ← cn.getType ()
2 metric ← cn.getMetric ()
3 stat ← cn.getStatisticType ()
4 foreach rqc ∈ strategy.requestClasses do
5     if rqc.metric ≡ metric then
6         if stat ∈ rqc.metric.stats then
7             return true
8 return false
```

In a second step, the `isApplicableForSystem` function evaluates if a solution strategy can be applied under given system settings, that is, the decision engine checks if one of the solution approach `StrategyLimitations` appears within the system.

The `StrategyLimitation` itself provides the evaluation of a limitation by evaluating `appliesTo(System sys)`. Through this, appending new limitations requires no extension of the decision engine. Algorithm 6.3 shows the filtering according to limitations in pseudocode.

Algorithm 6.3: Evaluate limitations of a solution strategy for a system description.

```
Input: SolutionStrategy strategy, System system
Output: Booelan
1 foreach limitation ∈ strategy.strategyLimitations do
2     if limitation.appliesTo (system) then
3         return false
4 return true
```

6.3.2 Rating According to Costs

The filtering for applicable approaches may result in a set from which a suitable performance evaluation approach has to be selected. After filtering applicable approaches, remaining approaches have to be rated according to their costs. In case the concern includes a cost type to optimize for, the decision engine rates applicable approaches and returns the most cost-efficient approach.

The selection of a suitable performance evaluation approach is described in Algorithm 6.4. As users might not want to specify cost comparison attributes in every concern, we additionally allow to specify a default order of cost comparison attributes within the decision engine. This allows omitting the cost type in the concern definition.

So far, we assumed a complete definition of all registered capability models. On the one hand, functional capabilities often can be specified based on interfaces which justifies this assumption. In contrast, even superficial knowledge of non-functional capabilities requires multiple analysis executions. Especially, the estimation of analysis costs is challenging. The question arises how to decide when meta-information about the costs is not provided for a capability model? To cope with this, we extend the decision logic to forward information on what is missing to the user. Exemplary, a response of our recommendation system could look like this:

Algorithm 6.4: Rate solution strategies according to analysis costs.

Input: List<SolutionStrategy> strategies, System sys, Concern cn
Output: Sorted list of <SolutionStrategy, cost> tuples

```
1 costType ← cn.getOptimizationAttribute ()
2 if costType is undefined then
3     costType ← defaultCostType              // e.g., time-to-result
4 List<SolutionStrategy, double > ratedApproaches ← ∅
5 foreach strategy ∈ strategies do
6     ratedApproaches.add(strategy, getCosts (strategy, sys, cn))
7 ratedApproaches.sort ()
8 return ratedApproaches
```

> "We assume that approach X is the optimal approach concerning `time-to-result`. However, we have no information about `time-to-result` for approaches Y and Z."

if time-to-result estimation is given for X, or like the following:

> "We assume that either X,Y, or Z is optimal concerning `time-to-result`. However, we have no information about `time-to-result` for the named approaches."

if not modeled for any solution approach.

6.4 Estimation of Analysis Costs

The costs for performance analysis approaches are commonly quantified globally not explicitly considering concrete analysis parametrization and characteristics of the investigated system (cf. Section 3.2). This applies for model-based analysis was well as for measurement-based analysis. Solution techniques for performance models are commonly classified to be either fast or slow. APM tools are promoted by their maximum system overhead. This categorization is easy to comprehend and can therefore be useful. However, at a second glance, such simplifications do not cover the entirety of analysis use cases, can be vague and may lead to wrong assessments.

There are various cost types to describe performance evaluation approaches (cf. Section 6.2). While we assume the methodology presented in the following to apply to multiple cost types, we focus on time-to-result which we consider to be the most predominant cost type for model-based performance prediction.

6.4.1 A Machine-Learning-Based Approach to Estimate the Time-to-Result for Model-Based Performance Prediction

The machine-learning-based approach to estimate the time-to-result for model-based performance prediction presented in the following is a result of joint work with Cristina Llamas Beltran which has not yet been scientifically published.

Especially for run-time performance and resource management, the time required to execute a model-based analysis plays an important role [Hub+15]. There are evaluation scenarios where the prediction results need to be available in a limited period of time such that the system can be adapted before SLAs are violated [Hub14]. Consequently, the model-based analysis itself has to satisfy time constraints. Otherwise, if performance prediction itself does not satisfy time constraints, reconfiguration would happen after performance problems have manifested themselves within the production system. The analysis time is naturally constrained by the time when reality catches up or even overtakes model-based prediction processes. Additionally, the time required to execute reconfigurations should be considered when determining upper limits for model analysis time.

A major problem for model-based analysis at run-time is that we do not know the time required for model analysis. The prediction of time-to-result is challenging as the computational overhead depends on performance model characteristics *and* analysis parametrization. Also, the response time behavior of alternative solvers may depend on *different characteristics* of the model and the analysis configuration. For example, some model-based analysis approaches are insensitive to performance model complexity but sensitive to workload definition and vice versa [Mül+16]. The choice of an appropriate solver is difficult as model solvers differ in conceptual and implementation details. Implementations are often optimized for certain analysis use cases (e.g., small models) resulting in worse performance for other use cases (e.g., large-scale models). Comparison of solution approaches often appears when presenting new solvers. In this context, authors tend to focus on models and configurations that are beneficial for their newly proposed solver.

While time-to-result predictions for analysis scenarios would provide many benefits, there is no solution to automatically derive them yet. Commonly, the selection of a model-based performance evaluation approach is based on a superficial knowledge that some approaches mostly provide quick results why others do not. This allows to derive an ordering of solvers, but no estimation of required time-to-result. There have been exploratory studies (e.g., by [Bro+15; Mül+16]) that compare accuracy and time-to-result of solution approaches and solvers based on example models. To derive predictions for a

concrete analysis setting, users would have to compare their own model and analysis characteristics to the analysis settings applied in a similar case study. Besides the lack of automation, there is only a small set of investigated models used in existing case studies are limited and their selection is often biased to demonstrate benefits of a single solution approach. Comparing accuracy and time-to-result, most studies do not provide actionable rules to decide when to use which solution approach. Static decision trees, as presented in [Bol+06; Bro+15], provide ordering of solution strategies but do not answer if a solution strategy is applicable under given time constraints. Based on the current state-of-the-art, one showstopper for the integration of model-based predictions into time-critical online processes is the lack of automated time-to-result estimation.

To answer the question about when analysis terminates, we formalize the problem of time-to-result prediction for model-based analysis and reduce it to a non-linear regression problem. We contribute an approach to estimate the analysis run-time based on model characteristics and analysis parametrization. At this, one challenge is to select performance-relevant features. Therefore, we analyze the impact of model characteristics and analysis parameters on the time-to-result using statistical tests. We determine performance-relevant features and implement their automatic extraction from analysis specification. Based on performance-relevant features, we perform systematic experiments to train a prediction model using machine-learning algorithms. We then apply the trained prediction model to perform a time-to-result estimation for concrete analysis scenarios.

Our approach allows for a cost-aware choice of performance analysis approaches tailored to a given analysis scenario. We apply statistical learning to address the problem and to model the correlation between model characteristics, analysis parametrization, and the time-to-result.

Compared to white-box performance modeling of the analysis logic, our black-box approach does not require to understand and model the internal behavior of the performance model analysis software by hand. Moreover, our approach supports reasoning at run-time by providing predictions within a few milliseconds. This can be considered to be significantly faster than solving white-box models using simulation or analytical solvers. Moreover, our approach is in favor when it comes to updates. Our prediction model can be automatically relearned on the availability of additional training data. Predictions can be used to automatically decide about the suitability of a given analysis method for given time-to-result requirements. Time-to-result awareness would allow to avoid running analyses that exceed time constraints. Integrated into solution approaches, our approach enables them to reason about their

time-to-result in a self-aware manner [Kou+17].

Methodology

The approach presented in the following enables the prediction of time-to-result for model-based analysis based on information about model characteristics and analysis parameters. First, we discuss the practical applicability of black-box and white-box prediction methodologies for our time-to-result prediction problem. Then, we describe and formalize the prediction methodology followed by the methodology to determine performance-relevant characteristics.

Discussion of Prediction Methodologies

The estimation of performance model analysis time requires to apply a prediction methodology. The prediction can be realized using either black-box machine-learning approaches or white-box performance modeling.

Applying white-box performance modeling requires to understand, quantify, and model control flow and parametric dependencies of the analysis method explicitly. However, performance modeling formalisms and their solution approaches come with many model and analysis characteristics that are interrelated in the prediction process. While several modeling approaches technically provide modeling capabilities [Kou+16], the requirements for understanding, formalization, and quantification make white-box approaches infeasible to apply for performance model analysis time prediction. We argue that explicit modeling is not worth the effort, as advantages of white-box models, like compositionality [SH12], are not exploited within the given problem statement.

In contrast to white-box approaches, machine-learning techniques automatically derive inter-relations and relieve the performance engineer from explicit modeling. Machine-learning approaches require several analysis runs to train a prediction model which is feasible for model-based analysis. As a consequence, we propose and evaluate machine-learning-based techniques for analysis time prediction of a model-based analysis approach.

Learning and Prediction Process

We formalize the problem, illustrated in Figure 6.6, as follows: For previously unmeasured configuration tuples x of performance model characteristics and a corresponding analysis parametrization, we want to know the time-to-result y for model analysis. Each variant can be represented as a selection of features,

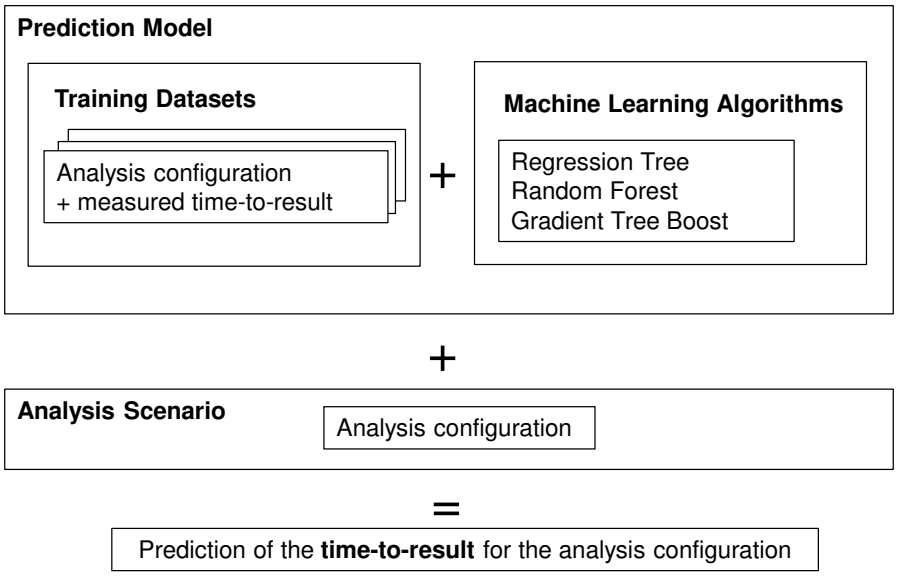


Figure 6.6: Time-to-result prediction approach.

called an *analysis configuration*. Based on the configuration input, our approach returns the time-to-result prediction (i.e., $f(x)$).

We train a prediction model using supervised learning. We learn a function $f : R^n \rightarrow R$, based on given training datasets $(x_1, f(x_1)), ..., (x_n, f(x_n))$. At this, the features are a combination of model and analysis characteristics x and the dependent variable time-to-result $f(x)$.

The main challenge in applying machine-learning to predict the time required for performance analysis is to determine the features impacting the analysis run-time. Potential features can be extracted from model description and analysis configuration. In contrast to many other machine-learning application scenarios [BL14; BE04; Did+15; Guo+13; Zha+15; ZT07], there is no intuitive feature candidate set known in advance. We argue that it can be selected based on fundamental comprehension of performance engineering. However, we cannot provide an automated methodology to efficiently discover promising candidate features yet. We apply a quick test to prune the potential feature set based on statistical hypothesis tests (refined in the next section). After the features impacting time-to-result have been determined, we perform measurements for a cross product of feature values to generate an initial training dataset.

Based on this problem description, different machine-learning techniques can be applied. For the evaluation, we performed a preselection of promising machine-learning algorithms. We consider regression-based approaches to be the most suitable machine-learning approach for our setting. Random forests are generally considered to provide relatively good accuracy, robustness, and ease of use [PJ17]. Therefore, we selected regression trees, random forests, and gradient tree boost for evaluation.

Greedy Feature Selection Methodology

To obtain a suitable prediction model, the effects of model characteristics and analysis parametrization have to be investigated to derive suitable features. The feature candidates have to be selected manually and feature value extraction routines have to be implemented. At this, it is important to mention that it is helpful to draw upon expertise in the analyzed model-based prediction approach.

The available modeling features and the analysis interface provide a superset of possible features for prediction. Obviously, different solution strategies may depend on different features. For example, analytic solvers may depend on other features than simulation. Therefore, the determination of features has to be repeated for each modeling formalism and analysis approach separately.

The process of identifying the most relevant features is called *feature selection*. The central premise applying feature selection is that the data contains features that are either redundant or irrelevant. An appropriate feature selection for time-to-result prediction pays off as many model features do not impact the time-to-result. For example, the naming of software components obviously has no impact on simulation time. Considering only a small set of features provides three benefits: First, models are more comfortable to interpret. Second, it leads to shorter training times and avoids the curse of dimensionality. Finally, it facilitates generalization and avoids overfitting by reduction of variance.

In contrast to many machine-learning applications which inherently provide a feature set, like the performance prediction for software product lines [Guo+13; Zha+15], the time-to-result prediction for model-based analysis has no predefined feature candidate set. There are no degrees of freedom and corresponding variation routines already defined by variation scenarios. It is important to note that there is no natural mapping of modeling and analysis constructs to features. There can be alternative mappings of model and analysis configuration to features. Potential features capturing deployment complexity could be, for example, the maximum or the average number of components per host, as well as the total number of deployed components.

Additionally, combinations of multiple primary metrics into a single feature could be considered. The context of time-to-result prediction for model-based analysis, classical feature selection approaches cannot be initially applied, as they assume a predefined feature candidate set.

In the following, we present a pragmatic statistical methodology that provides the advantage to evaluate the effect of features on time-to-result quickly. However, it comes with the disadvantage that we may overlook performance effects that occur only on the interplay of multiple variables. To determine performance-relevant characteristics, we formulate hypotheses on the impact of model characteristics and analysis configuration parameters on the time-to-result of an analysis. For each hypothesis, we generate training vectors varying the related model or analysis parameter which represents an independent variable. To obtain significant input data for statistical tests, we repeat twenty sequential execution time measurements for each configuration. Based on measurements, we can test the following hypotheses:

ANOVA Test For two time-to-result series, corresponding to two different analysis configurations, we formulate the null hypothesis that time-to-result means are equal. In case of rejecting the null hypothesis the analyzed characteristic has a statistically significant effect on the execution time.

The F-test decision is based on F and p measures. The *F-value* is a ratio of the variability between both series divided by the variability within one series. The p-value corresponds to the right area of the F-distribution and indicates the smallest level of significance that would allow the rejection of the null hypothesis [Mon08]. A large F ratio indicates a statistically significant variation between the samples of the groups. The null hypothesis assessment compares the p value to the significance level α. If the p-value is less than or equal to α, the null hypothesis is rejected indicating that not all of the population means are equal. If the probability p is greater than α, there is not enough evidence to reject the null hypothesis that the population means are equal [Mon08]. We set the ANOVA test significance level to 0.01 which corresponds to a risk of one percent in concluding that a difference in the means exists when there is no actual difference [Mon08].

HSD Test Based on the influencing factors deduced from ANOVA test results, we apply the *Honestly Significant Difference (HSD)* test to identify the degree of impact. HSD allows to compute a single value in order to determine the minimum difference between group means required to conclude that there

is a significant difference. The HSD test makes use of the distribution of the studentized range statistic

$$q = \frac{\overline{y}_{max} - \overline{y}_{min}}{\sqrt{\frac{MS_{within}}{n}}}$$

where $\overline{y}_{max}$ and $\overline{y}_{min}$ are respectively the largest and smallest group means out of a group of p equally-sized group means. MS_{within} is the mean square within groups and n is the group sample size. The HSD test considers two group means significantly different if the absolute value of their difference exceeds

$$HSD = q_\alpha(a, f)\sqrt{\frac{MS_{within}}{n}}$$

for a significance level α, where a is the number of groups and F the degrees of freedom associated with MS_{within}. We select a significance level α of 0.01 for the HSD test. The distribution of $q_\alpha(a, f)$ has been tabulated and can be consulted in fundamental statistic literature, e.g., [Mon08].

Assessment of Test Results Based on ANOVA and the HSD test, we can determine characteristics impacting the time-to-result. These can be applied as an input to generate analysis variations and train a prediction model. As a result, one obtains features that impact time-to-result. The methodology does not specify when to stop appending new features. In general, the model refinement process (i.e., adding new features) can be terminated if predictions reach acceptable accuracies. The statistical tests suit to substantiate that there is an impact of a model or analysis characteristic on time-to-result. If no impact can be shown this does not prove that a certain feature has no impact for additional models. In case of adaptations to meta-model or model analysis, features have to be investigated again. If the set of features has grown to be hardly handleable, which is unexpected due to manual efforts in the case under consideration, classical feature reduction approaches can be applied. For example, one could test for statistical independence of features and prune the feature set on negative test results.

6.4.2 Feature Engineering Example

While we consider our prediction methodology applicable to multiple performance modeling and solution approaches, we will demonstrate feature selection for the solution process of the DML [Kou+16; Hub+17] which we

apply throughout this thesis. First, we quickly refresh characteristics of DML and its declarative solution approach and discuss challenges in predicting time-to-result. Then, we introduce the used base models for training and evaluation. Finally, we determine the features impacting the time-to-result.

Subject Solution Approach

Investigated Modeling Language DML provides an architectural performance modeling language designed for run-time performance and resource management. This architectural performance modeling language has been applied for various purposes, in different domains, and at different development stages in online and offline scenarios. DML is designed to serve as a basis for self-aware systems management during operation, ensuring that system quality-of-service requirements are satisfied while infrastructure resources are utilized as efficiently as possible. DML has a modular structure and is provided as a set of meta-models for modeling the usage profile, resource landscape, and application architecture. The application architecture meta-model can be further divided into repository and deployment meta-models. The component specification defines the relation between service input parameters and the performance effect of a service call. The deployment specification maps component instances to available resources, e.g., servers. The usage profile defines a set of usage scenarios that describe the investigated user behaviors. The explicit specification of usage profiles allows to evaluate whether a system adheres to SLAs for a set of given user behaviors.

The various submodels of DML reference elements from each other. They are interconnected in many parts. For instance, the specification of a deployment creates pointers to the resource landscape and a component description. Linking between elements is common for architectural languages as it allows to reuse elements in several contexts. For example, a single component may be deployed on multiple servers and a single server may host several components. In summary, DML includes many modeling features and allows for cross-references between elements which may potentially impact the analysis performance. This makes DML a challenging candidate for time-to-result prediction.

Investigated Performance Evaluation Approach While architectural models allow for different analysis approaches, this work focuses on DML's commonly used solution approach [Bro+15; Hub+15]. The model solution applies a declarative paradigm [Wal+16a], which means that analysis is triggered based on a

performance query. The declarative approach reduces the analysis configuration complexity. Instead of specifying analysis parameters like simulation run length or accuracy explicitly, analysis constraints like `fast` or `accurate` are used to determine a suitable simulation configuration internally.

The solution of DML models applies a multi-step approach. It includes a model-to-model transformation followed by simulation and result filtering. The Descartes Query Language (DQL) framework [GBK14] triggers a transformation to QPNs [MKK11], followed by a simulation using the batch-means method implemented by the SimQPN simulation engine [Kou+16], and finalized by filtering of simulation results.

One challenge for time-to-result prediction is the complexity of the multi-step approach. Another challenge for the time-to-result prediction is that the simulation applies two termination criteria in parallel: maximum virtual time and sufficient accuracy.

Subject Models

We perform the determination of performance-relevant features based on the models presented in the following. All models have been taken from a public repository [1]. To allow for an extended set of models, the presented models serve as a basis to create several model *variations*, which we explain in corresponding experiments. We use the term model variation to describe a replication, in which one or more features (analysis configuration or model characteristics) have been modified. Model variations are necessary to study the impact of characteristics on the time-to-result of the performance analysis.

HelloWorldExample This is a minimal model to demonstrate DML usage. It contains the lowest architectural complexity of all presented models. It implements a closed workload with a population of 10 clients and a think time following an exponentially distributed interarrival time of 4.2 seconds calling a single interface. The repository contains a basic component providing a single functionality characterized by a single resource demand.

SPECjEnterprise2010 The model is based on the supplier domain of the SPECjEnterprise2010 standard benchmark[2] and builds upon a case study presented in [BHK14]. The model represents a scenario where users send requests to the supplier domain to order car parts from suppliers. The supplier domain

[1] https://se3.informatik.uni-wuerzburg.de/descartes/dml-examples
[2] http://www.spec.org/jEnterprise2010

then places a purchase order with a selected supplier offering the required parts. The usage profile contains a closed workload, with a population of 80 customers and an exponentially distributed think time. A single signature call represents a new order request. Two basic component objects represent a WebLogic server and a database server. When a new order request arrives, the WebLogic server processes it and executes an external call to the database server.

Petclinic The Petclinic model has been created by an automated extraction based on execution trace files using the approach presented in Chapter 5. The model represents the Petclinic application [3], a portal for vet appointments that has been deployed on a high-end server in a VM with 42 cores. The provided analysis setting contains an open workload with an exponentially distributed arrival rate of 780 customers per second. Each vet customer visits the front page, looks at the list of all available vets, searches pet owners, and displays pet owners. Due to the automated extraction, the model provides the highest level of detail of all models. The repository model contains eleven basic components, most of them providing multiple interface providing and requiring roles with complex control flows including multiple resource demands and external calls inside loop actions.

ThreeTierArchitecture In addition to models from case studies, we implemented a model generator for typical three-tier architecture models. In the generated models, each tier is deployed on a separate resource container. The default workload is closed with a population of 100 clients and a think time following an exponentially distributed interarrival time. Each client calls the three interfaces one after another. The first interface represents a request from the client tier to update a database entry. The client tier performs an external call to the business logic tier, which processes this information, and then forwards to the database tier to perform the corresponding write operation. The second interface represents the request from the client tier to process a piece of information. For this purpose, the client tier performs an external call to the business logic tier. The third interface represents the persistence of information in the client tier; it does not involve external calls to services of other tiers.

At the time we performed determination of features, the presented models were the only ones available and supported by solving.

[3] https://github.com/spring-projects/spring-petclinic

Table 6.1: ANOVA test results for variation of processing constraints.

Model	processing constraint	$\overline{x}$ (ms)	σ (ms)	F	p
SPECjEnterprise2010	unconstrained	7482.1	51.23	5.85	0.0048
	accurate	7467.35	17.84		
	fast	7444.45	27.34		
HelloWorld	unconstrained	17917.45	35.52	209.53	<.0001
	accurate	17755.65	31.79		
	fast	17696	38.55		
HelloWorld (modified)	unconstrained	26042.4	1195.102	15.6	<.0001
	accurate	24974.3	68.106		
	fast	24987.85	104.6031		

Determination of Features

As model complexity and analysis efforts do not necessarily correspond, the challenge is to find suitable indicators for analysis complexity. For example, the number of component definitions does not affect solution complexity as workload descriptions do not have to trigger executions at all components.

We performed a definition and preselection of features based on an investigation of model transformations, simulator implementations, and the experimental results presented in [Mül+16]. The selection of models capable to test a hypothesis is limited by their inherent features. For example, effects of open workload parametrization cannot be tested using a closed workload definition. Some modeling features, like open or closed workload definition, imply a natural transformation which allows for additional analyses when analyzing both configurations. Other features imply no natural modification rules.

We investigate only features supported by DML solving which is less than provided by the modeling language. Some modeling features are not yet supported by solving and have therefore not been used so far. In the following, we present our evaluation of several feature hypotheses.

Hypothesis 1 - Impact of processing constraints

The aim of the first hypothesis is to show that processing constraints in a performance query affect its execution time. To this end, we vary the processing constraints `accurate`, `fast`, and `unconstrained` which is the default. The analyzed models are SPECjEnterprise2010, HelloWorld, and a variation of HelloWorld where the usage profile includes an additional loop.

Table 6.1 shows that for the analyzed models the probability p is below the significance level. Thus the null hypothesis can be rejected. It can be concluded

Table 6.2: ANOVA test results for variation of compute nodes.

Model	# nodes	$\overline{x}$ (ms)	σ (ms)	F	p
SPECjEnterprise2010	2	7785.90	1279.38	2.13	0.1282
	4	7368.65	22.2149		
	8	7367.70	25.3234		
HelloWorld	3	14680.00	330.9	0.53	0.591477
	6	14780.45	309.82		
	9	14723.45	282.38		

that not all population means are equal, which means that the accuracy level of the query has a statistically significant effect on the query execution time.

The post hoc HSD test results of 33.64, 33.95, and 665.28 show that the impact of processing constraints depends on the considered models. Especially for the HelloWorld model all pairwise mean comparisons differ significantly. For the SPECjEnterprise2010 model, the `unconstrained` versus `fast` comparison shows a significant difference. The investigation of a modified HelloWorld model showed means to be significantly different except for the `accurate` versus `fast` comparison. The modification includes appending a loop to the usage profile, as well as setting the population to 15 and the rate of the exponentially distributed think time to 6.2. In conclusion, query constraints may impact the query execution time, but it can not be confirmed in all test cases. The constraint has no influence on time-to-result when models can be solved accurately and fast at the same time.

Hypothesis 2 - Impact of compute node count In this hypothesis, we analyze the effect of the number of compute nodes specified in the resource landscape model. In the test scenario, we measure execution times for the SPECjEnterprise2010 model and variations replicating compute nodes two and four times.

Table 6.2 shows that the probability p exceeds the significance level α for both models. Thus, there is not enough evidence to reject the null hypothesis. For the analyzed setting, the number of compute nodes in the resource landscape model has no statistically significant effect on the execution time. This conforms to expectations since our variations do not include allocation of software components to the replicated nodes. The number of nodes only impacts time-to-result when software is deployed on it. Deployment is represented by separate modeling elements. We conclude that to naively consider the number of compute nodes depicts no suitable feature. As no significant F value was yielded in both cases, the HSD post hoc test was not performed.

Table 6.3: ANOVA test results for different workload types

Model	workload	$\overline{x}$ (ms)	σ (ms)	F	p
HelloWorld	closed	991.1	1078.8	2.31	0.1368
	open	624.55	13.12		
SPECjEnterprise2010	closed	6867.4	176.298	2.32	0.136
	open	6928.15	27.24		

Hypothesis 3 - Impact of workload type Performance models can be specified using either an open or closed workload definition. We set up the null hypothesis that the type of workload specification impacts performance.

All models in this experiment have been initially specified using a closed workload denoted by a given population and think time. We transform them to generate an arrival rate equivalent open workload definition and compare both models. In order to obtain an inter-arrival time $\frac{1}{\lambda}$ equivalent to the closed workload definition, we derive the system arrival rate by applying Little's Law [MAD04]: $N = \lambda R$ where N is the average number of customers in the queue, λ is the arrival rate, and R is the average response time. N corresponds to the user population, while R can be obtained using simulation.

Table 6.3 shows the results of the ANOVA test performed on each test scenario. The probability p exceeds the significance level α for the analyzed models. The results of our analysis show that the workload type has no statistically significant effect on the performance query execution time. Thus, having no significant F value, the post hoc HSD is not necessary. We conclude that workload type has no significant impact on time-to-result.

Hypothesis 4 - Impact of population in closed workloads The objective is to analyze the effect of population count in a closed workload on the execution time. We set up the hypothesis that population count has no impact on time-to-result. For this test scenario, we measure execution times for the SPECjEnterprise2010 model with populations of 80, 160 and 320 customers and the HelloWorld model with populations of 10, 20 and 40 customers.

Table 6.4 shows that the null hypothesis can be rejected, since the p value is below the α significance level. It can thus be concluded for the analyzed setting, that the population in a closed workload has a statistically significant effect on the execution time.

The HSD post hoc test results for SPECjEnterprise2010 (76.47) and HelloWorld (336.58) further indicate that the number of users has a significant impact on time-to-result.

Table 6.4: ANOVA test results for different populations in a closed workload.

Model	population	$\overline{x}$ (ms)	σ (ms)	F	p
SPECjEnterprise2010	80	7418.95	66	56680.59	<0.0001
	160	14695.3	31.11		
	320	14845.85	117.26		
HelloWorld	10	16732.35	47.94	47.66	<0.0001
	20	17295.6	63.05		
	40	17815.65	602.72		

Table 6.5: ANOVA test results for think time variations.

Model	think time	$\overline{x}$ (ms)	σ (ms)	F	p
HelloWorld	0.2	10201.78	102.96	1772.37	<0.0001
	4.2	12645.21	39.21		
	8.2	12527.57	221.2		
SPECjEnterprise2010	0.005	6705.65	83.77	17056.44	<0.0001
	0.05	12998.05	81.2		
	0.5	12722.85	175.7		

Hypothesis 5 - Impact of think time in a closed workload For closed workload models, we formulate the null hypothesis that think time has no influence on time-to-result. Using HelloWorld and SPECjEnterprise2010, we modify the exponentially distributed user think times by varying the arrival rate parameter λ. We measure the query execution time for the HelloWorld model with think times 0.2, 4.2, and 8.2 and the SPECjEnterprise2010 model with think times of 0.005, 0.05, and 0.5.

Table 6.5 shows for both models that the p value is below the α significance level. This leads to the rejection of the null hypothesis and the conclusion that think time in closed workloads statistically impacts the execution time. The HSD values for HelloWorld (140.7) and SPECjEnterprise2010 (116.78) are rather low. More in-depth investigations show that there is an increase of time-to-result until a saturation point. Further increases of think time do not affect the externally visible time-to-result behavior. This can be explained by the convergence of steady-state simulation convergence behavior.

Hypothesis 6 - Impact of inter-arrival time We formulate the null hypothesis that the job inter-arrival time for exponentially distributed arrivals in open workloads has no influence on time-to-result. For this test scenario, the inter-arrival time of the HelloWorld model is set to 0.0005, 0.05, and 0.5. For the Petclinic model, the parameterization is 0.0005, 0.05 and 5. The null hypothesis

Table 6.6: ANOVA for inter-arrival time variations.

Model	inter arrival rate	$\overline{x}$ (ms)	σ (ms)	F	p
HelloWorld	0.0005	397.15	13.33	13854.59	<0.0001
	0.05	630.57	9.35		
	0.5	3179.26	97.63		
PetClinic	0.0005	3112.57	517.07	103187.32	<0.0001
	0.05	15271.05	1428.37		
	5	121241.94	156.87		

can be rejected based on the results depicted in Table 6.6. The HSD values for the HelloWorld (56.36) and Petclinic (869.47) show that means are significantly different for all pairwise comparisons. The variance test results show that the inter-arrival time parameter has a significant impact on the measured time-to-result. When varying within the reference models, we found that the time-to-result of the specified query increased up to several minutes, for example, when the inter-arrival time of the HelloWorld model was set to five. If the arrivals exceed the maximum system processing capabilities, arriving jobs are queued which slows down analysis and may even lead to overflow. This observation can be explained by the solution approach applied by DQL for query processing. The connector implementation specifies a fixed upper bound for the SimQPN virtual simulation time. When varying the parameters inside the DML model structure, the simulation of the QPN obtained in the model-to-model transformation might not reach a steady state in the given virtual time. Then the query execution engine aborts simulation.

Hypothesis 7 - Impact of assembly connections Hypothesis 7 assumes an impact of the number of assembly connections the solution process refers to on time-to-result In DML, assembly connectors connect two assembly contexts which themselves refer to a repository component to specify the software. They represent a connection between an interface providing role of the first component and an interface requiring role of the second component.

To test the hypothesis, we created two variations of the SPECjEnterprise2010 model where we doubled the number of components. The first model variation contains a single assembly connector referenced by a single system call user action from the usage scenario. This means that only two of the four repository components are referenced within the analysis. The second model variation consists of two assembly connectors. The usage scenario contains two

Table 6.7: ANOVA for number of used repository components.

Model	comp. used	$\overline{x}$ (ms)	σ (ms)	F	p
SPECjEnterprise2010 (2 components)	0	3909.55	147.34	194.37	<0.0001
	1	7340.2	1090.54		
SPECjEnterprise2010 (4 components)	0	5147.25	17.21	11798.62	<0.0001
	2	10036.25	200.55		

Table 6.8: ANOVA test results for variations branchings and branching depth.

Model	variation	$\overline{x}$ (ms)	σ (ms)	F	p
3-Tier-Architecture	1 BranchUserAction	11354.23	109.57	89.01	<0.0001
	2 BranchUserActions	11511.11	31.78		
	3 BranchUserActions	11675.76	42.36		
	branching depth = 1	8662.84	16.9	80.6	<0.0001
	branching depth = 2	8840.68	18.94		
	branching depth = 3	9152.68	206.97		

system call user actions making use of all four basic repository components interconnected in two pairs.

DML's solution process considers only assembly connectors directly related to the system call user actions specified within the workload. Further, assembly connectors connecting components for possible chained external calls in the signature behaviors from the repository models will not be considered. For the analyzed model variations, Table 6.7 shows that the number of interconnected and used repository components has a statistically significant effect on the query execution time since the p value is below the significance level α in both cases and the null hypothesis is therefore rejected. The HSD post hoc test yields a minimum absolute difference of 570.13 for the group means to be significantly different. All pairwise comparisons conclude a significant variation.

Hypothesis 8 - Impact of branching number and depth. Hypothesis 8 tests if: (i) the number of branching elements, and (ii) the branching depth of nested elements within the workload definition have a statistically significant effect on the execution time. For this purpose, we analyze the Three-Tier-Architecture model which naturally includes the branching modeling feature.

In a first test scenario, we analyze three variations of the Three-Tier-Architecture model: we split the arrivals of the system call user actions of the usage profile models into one, two, and three branching actions. In a second test scenario, we investigate additional branch actions nested inside the original branch action.

Table 6.9: ANOVA test results for different loop iteration counts.

Model	Iteration count	$\overline{x}$ (ms)	σ (ms)	F	p
3-Tier-Architecture	2	5694.5	213.11	51.98	<0.0001
	4	5379.16	22.78		
	8	5301.22	25.79		
	10	5334.44	39.15		
	20	5295.55	19.85		

Table 6.8 shows that the branching depth in the usage profile model is statistically significant for the time-to-result for the analyzed setting. The HSD values for the post hoc variance analysis are 73.75 and 118.72 respectively for the branch count and branching depth. This confirms the results obtained in the ANOVA test showing all pairwise mean comparisons to be significantly different.

Hypothesis 9 - Impact of workload loop iterations DML enables to specify loops within the usage profile model. Loop probabilities can be described in various ways, we focus on the probabilistic description present in training models. A probability mass function (PMF) specifies the number of loop iterations and their corresponding probability. For example, the PMF IntPMF[2;0.4)(5;0.6)] indicates that the loop body is executed 2 times with a probability of 0.4 and 5 times with a probability of 0.6. We set up the hypothesis that the mean loop iteration count within the usage profile model implies no difference in execution times. To analyze the hypothesis, use a version of the three-tier-architecture model containing a single loop in its usage profile. We analyze with 2, 4, 8, 10, and 20 loop iterations each with a probability of 1.

Based on the results shown in Table 6.9, the null hypothesis can be rejected. Thus, it can be concluded that for the analyzed setting, the loop iteration count has a statistically significant impact on the time-to-result. However, only the comparison of two loop iterations with other configurations shows a significant difference. The explanation for convergence of simulation time when increasing the mean number of probabilistic loop iterations is it decreases fluctuations within simulation. Simulations having less fluctuation tend to converge faster towards stable indices.

6.4.2.1 Summary of Feature Determination

To find features, we evaluated hypotheses on nine model and analysis characteristics using ANOVA and HSD tests from which we selected seven features.

We classify features as major influencing factors having an F value greater than 500. F values between 50 and 500 indicate a moderate impact.

Characteristics with a major influence are: (i) the population and think time for closed workloads, (ii) the inter-arrival time in open workload models, and (iii) the number of referenced assembly connectors. Characteristics with a moderate impact include: (i) the number of branches within user actions, (ii) the branching depth, (iii) the loop iteration count, and (iv) the query processing constraints. Characteristics with no significant impact denoted by a positive ANOVA test include number of compute nodes and workload type.

6.5 Concluding Remarks

Solution approaches and tools for performance evaluation come with different capabilities, strengths, and limitations. They allow for interchangeability, but there is lack of automated and extensible decision support. In this chapter, we presented a methodology for automated selection of a suitable performance engineering approach tailored a given system and user concerns. We decouple the complexity of the approach selection into a decision engine and solution approach capability models. The proposed capability meta-model provides a methodology for the comparison of solution strategies. The meta-model splits into sub models for functional capabilities, accuracy, limitations, and costs. The decision engine can filter applicable solution strategies and rank them according to the user concerns. Compared to tree-based approaches, our approach allows to easily append and modify solution strategies as the performance engineer does not rely on the knowledge of previously integrated approaches. Also, our approach enables to easily introduce new comparison attributes like new costs, metrics or statistics. Section 7.2 evaluates the framework by demonstrating decision support for model-based and measurement-based performance evaluation. It discusses how to cover of existing decision support and how to adapt decision support to evolution scenarios.

Comparisons can be specified using static or context-aware cost estimations which for their part could be based on explicit specifications or machine-learning. This section also presented a methodology that allows for a context-aware estimation of the time-to-result for model-based performance evaluation using machine-learning algorithms. It is based on information about model *and* analysis characteristics. This enables prediction approaches to get self-aware concerning time-to-result. We apply our methodology to a representative model-based analysis approach and deduced performance influencing factors. The evaluation results presented in Section 7.3 show that based on a small set

of training models, accurate time-to-result predictions can be achieved. The evaluation considers estimations for models not included in training data and estimations where alternative analysis configurations of the considered model have been used for training. The prediction for unknown models exhibits a mean deviation below half a second and a mean percentage error below 20%. The accuracy can be significantly improved when predicting time-to-result for analysis configurations of previously analyzed models.

Part III

Validation and Conclusions

Chapter 7

Validation

In this section, we present an evaluation of the main contributions of this thesis. For each contribution, we first describe the goals and research questions guiding the evaluation, before we discuss our validation results. In Section 7.1, we evaluate the framework for automated extraction of architectural performance models. In Section 7.2, we evaluate our approach for automated decision support. In Section 7.3, we evaluate the time-to-result prediction accuracy. In Section 7.4, we evaluate the framework for Declarative Performance Engineering (DPE).

7.1 Automated Creation of Architectural Performance Models

In this section, we evaluate our extraction framework for architectural performance models, called Performance Model eXtractor (PMX), presented in Chapter 5. The evaluation of model extraction targets the savings in model creation, the reusability of the generic framework for multiple modeling formalisms, and the prediction accuracy of derived models. We refine our evaluation goals by the following research questions:

RQ1 *What are savings of fully automated model extraction compared to application-specific modeling efforts?* (Section 7.1.2)

RQ2 *What is the prediction accuracy of performance models extracted by PMX?* (Section 7.1.3)

RQ3 *Are performance prediction results equivalent using builders implementations for different modeling formalisms so that they can be considered interchangeable?* (Section 7.1.3)

RQ4 *What are benefits of toolchain parallelism achieved by multiple builder implementations?* (Section 7.1.4)

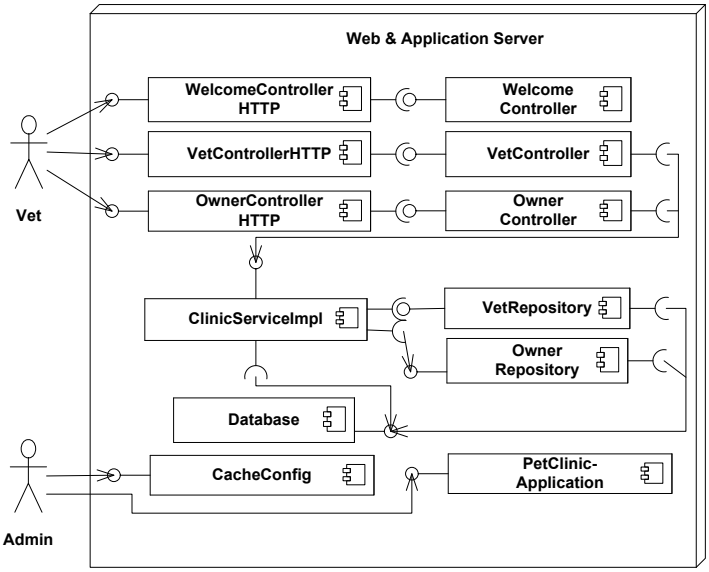


Figure 7.1: Petclinic application architecture.

RQ5 *What are the savings implementing a formalism specific builder (and reusing the PMX framework) compared to the implementation of formalism specific extraction software from scratch?* (Section 7.1.5)

The generic model extraction framework can be combined with a builder implementation for a concrete modeling formalism to extract models using that formalism. We implemented model builders for the PCM and the DML formalism as part or our evaluation. Based on these formalisms we use different downstream analysis tools to analyze resulting models. Moreover, we compare resulting models to measurements and between each other. Prior to answering the research questions we briefly introduce the case study systems.

7.1.1 Case Study Systems

Petclinic

To evaluate our approach, we selected the Petclinic application [1] representing a portal for vet appointments. It is a sample application to demonstrate the features of the Spring framework. Figure 7.1 presents the coarse-grained application architecture. The HTTP components render web pages based on the

[1] https://github.com/spring-projects/spring-petclinic

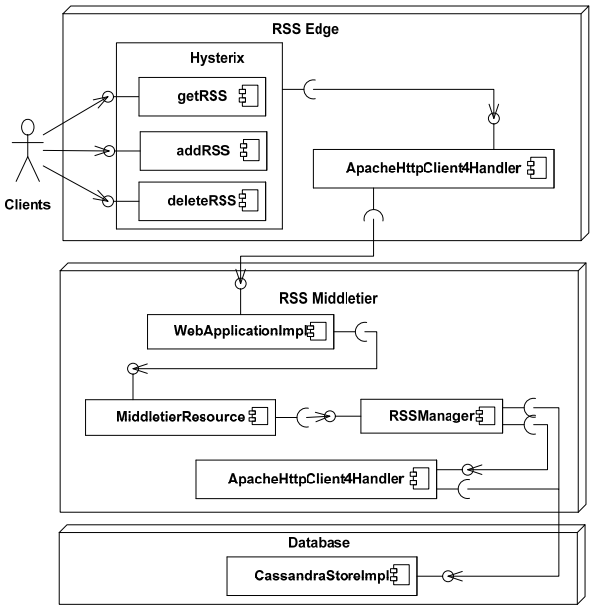


Figure 7.2: Netflix RSS reader application architecture.

information gathered from calls to the business components of the same name. The application communicates with an in memory H-SQL database.

We deployed the application on a Dell Power Edge R815 with 48 cores, each core equipped with an Opteron 6174 CPU 2.6 GHz. The application was running on an Ubuntu 14.04.5 VM equipped with 16 GB RAM (to be no bottleneck) and an assignment of 40 cores.

The portal can be started using a default content initialization which we applied for our experiments. We modified the “browse and edit" workload shipped with the application removing the “edit" operations to avoid database contention due to locking. The workload has an exponentially distributed arrival rate. Each vet customer calls the following sequence of interfaces: The vet visits the front page `welcomeGET`, looks at all vets `showVetListGet`, then searches pet owners `initFindFormGet`, `processFindFormGet` and displays all pet owners using `showOwnerGET`. Then the vet triggers two times a specific pet owner page `processFindFormGet`. Besides workload, we modified the application to cache the vet catalog once at startup to improve the application performance.

Netflix RSS Reader

Over the years, Netflix has open sourced many of its components to build microservice applications such as Hystrix, Eureka, Karyon, and Ribbon. The Netflix RSS reader is a microservice application that ties them up to showcase how Netflix implements microservice applications. Figure 7.2 presents the coarse-grained application architecture of the Netflix RSS reader application. The Eureka component is used to discover the middle tier instances that fetch the RSS feeds subscribed by the user. The Hystrix component is used to provide a greater tolerance of latency and failures when communicating with the middle tier service. Using Hystrix, the Netflix RSS reader application adopts its behavior dependent on the incoming traffic. It implements a circuit breaker which triggers a fallback behavior at higher loads. The base container for this edge service is also built on top of Karyon. A simplistic UI is provided by the edge service to add, delete and display RSS feeds. The RSS Middle tier fetches the contents of RSS feeds from external feed publishers, parsing the RSS feeds and returning the data via REST entry points. Ribbon's HTTP client is used to fetch the RSS feeds from external publishers. This tier is also responsible for persisting the user's RSS subscriptions into the database.

We deployed the application on a HP ProLiant DL360 Gen9 server. It features an Intel(R) Xeon(R) CPU E5-2640 v3 with 8 cores at 2.60GHz. The application was running on an Ubuntu 5.4.0, equipped with 32 GG RAM. We applied a Dockerized version instrumented with Kieker monitoring for our experiments. For the presented experiments, none of the Docker containers had resource constraints.

To realistically test an RSS reader, it has to be parameterized using RSS feeds. However, the application is not shipped with a default configuration of feeds. To parameterize the RSS reader in a way to ensure reproducible performance evaluations, we collected seven text-only RSS feeds from popular news sites at November 22^{th}, 2017. In particular, we included RSS feeds from ABC news, BBC news, CNN, Deutsche Welle, Forbes, Reuters, Tageschau, and Wall Street Journal. For our experiments, we stored the RSS feeds locally and provided them using an NGINX server to exclude side effects from external feed providers. To initialize the application, we added the named RSS feeds prior to load experiments which then repetitively trigger the `getRSS` interface.

7.1.2 Savings of Modeling Effort

To automatically generate performance models for the described systems, we collected monitoring data during a low load calibration workload execution.

Application	EMF elements		EMF references	
	DML	PCM	DML	PCM
Petclinic (Spring)	376	441	265	488
RSS Reader	98	72	97	116

Table 7.1: Complexity of resulting models for case studies.

We used resulting monitoring data to trigger performance model extraction applying the builder implementations for PCM and DML. We manually investigated the architectural properties of the learned models and compared them to results form the Kieker analysis framework. All measured system components have been detected. Learned infrastructure and deployment models comply with the real system setup. The model of the Netflix RSS reader had to be adjusted to include load-dependent behavior.

In [Hub+10], [SW01], and [LB16] the manual modeling of a system of similar complexity took between three weeks and one person-month (loosely distributed over three months). Professional software architects and performance engineers performed the case studies. For larger systems or less experienced users, this effort is expected to be significantly higher. The named case studies do not state how much time has been spent on modeling and how much time on the setup of measurements, the creation of application-specific measurement and model-creation scripts, and repetition of experiments.

To illustrate the effort of a manual creation, we counted the number of created EMF elements as well as generated cross-references between model elements. Table 7.1 compares the number of EMF elements and internal cross-references for case study models. To generate training data, we applied monitoring configurations that ignore system startup and monitor the system at a coarse-grained level. Otherwise, the size of resulting models can quickly grow up to thousands of EMF elements. These numbers indicate that model creation in an industrial setting requires automation. Deviating counts for different formalisms derive from differences in modeling control flow and resource landscape.

7.1.3 Evaluation of Model Prediction Accuracy

In this section, we evaluate the prediction accuracy of the architectural performance model extracted in our case studies under different transaction rates. In general, the prediction accuracy depends on many factors such as the system

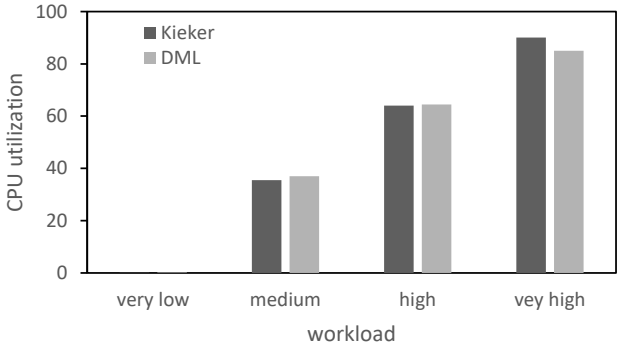
complexity, the granularity of monitoring data, or how much a system deviates from the assumptions (detailed in Section 5.4, e.g., CPU bound workloads).

The extracted models should provide sufficiently accurate prediction results compared to the actual system performance. Usually, deviations within 30% for response time and deviations within 5% for resource utilization are considered acceptable for capacity planning [MV00]. At high utilization, a 1-2% increase in utilization can more than double the response time [Wes13]. This makes response time predictions at high utilization brittle [Wan+12].

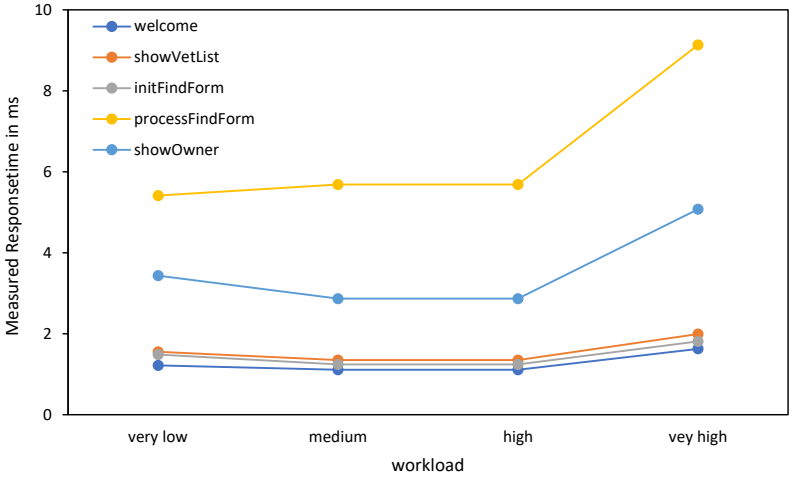
For the described setting, we measured a calibration workload at low utilization used to trigger performance model extraction applying the builder implementations for PCM and DML. The Netflix RSS reader application required subsequent adaptations as it changes its behavior on different load-levels. Then we performed benchmark measurements using the Kieker monitoring framework for different load levels and compare them to simulation results of the extracted models.

We derived performance metrics in the following way: Kieker log files were filtered using the declarative framework we presented in [Blo+16] to derive response times. Utilization has been measured using Kieker and custom Python scripts based on the psutil library, backed-up by the Linux top command. DML Analysis used a transformation to Queueing Petri Nets (QPNs) [MKK11] and simulation using SimQPN simulation engine. PCM analysis was performed using SimuLizar [BLB13] simulation engine.

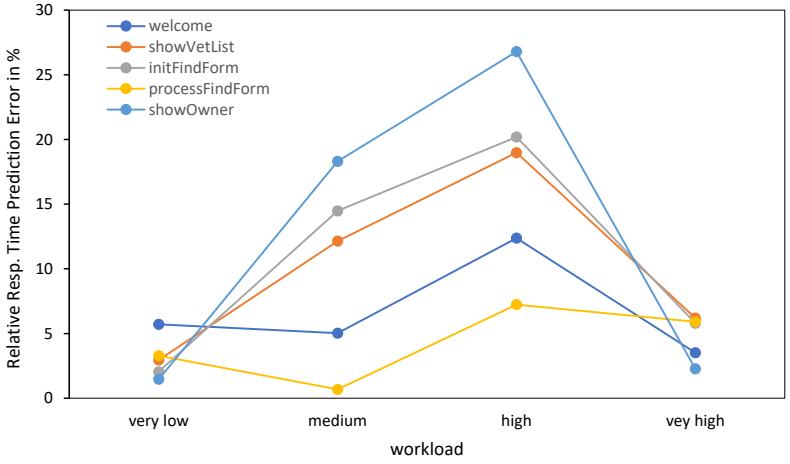
For different modeling formalisms, we investigate the accuracy of performance predictions of the derived models. For both evaluated scenarios we receive accurate model-based predictions. The Petclinic evaluation results are depicted in Figure 7.3. The deviation for utilization is below 3%. The deviation for response time is mostly below 20%. Figure 7.4 presents the evaluation results for the Netflix RSS reader case study. The absolute deviations for predicted utilizations are below 3%. While we underestimate utilization for the low load experiments (e.g., 50 requests per second), we overestimate already for medium load-levels (e.g., 300 requests per second). We conclude that the application has a load-dependent CPU demand which is not yet supported by existing modeling tools or solvers. Considering the fact of load-dependent CPU demands, the achievable prediction accuracy is limited, and the achieved accuracy is sound. The response time prediction error is less than 5% except for the highest load scenario where a fallback mechanism answers 96% of requests. The deviation of about 22% can be explained by a simplifying assumption of randomly distributed non-fallback executions which does not correspond to the circuit breaker behavior of the RSS reader system. In our experiments, the



(a) Measured and predicted utilization.

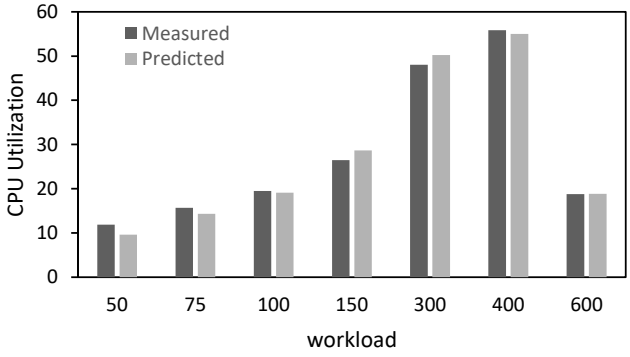


(b) Measured average response times.

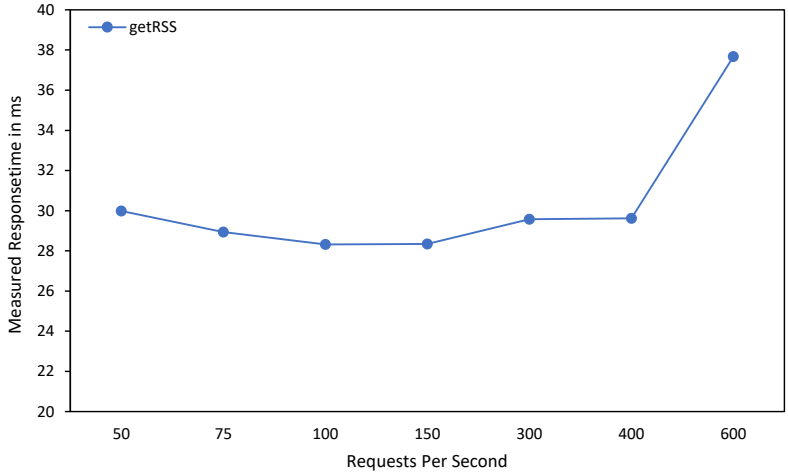


(c) Relative error of response time predictions.

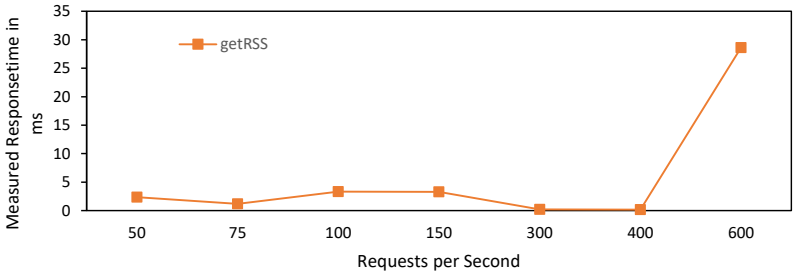
Figure 7.3: Measurements, prediction results, and prediction errors for the Petclinic scenario.



(a) Measured and predicted utilization.

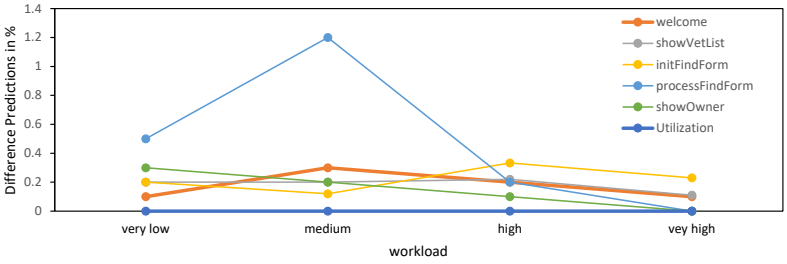


(b) Measured average response times.

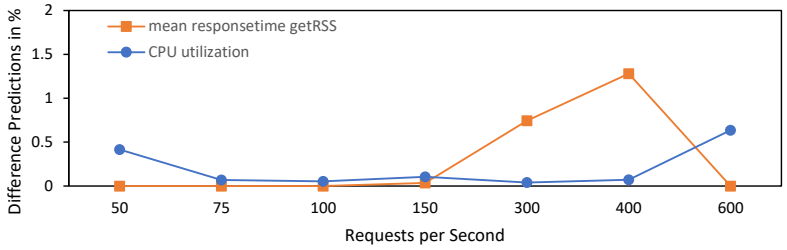


(c) Relative error of response time predictions.

Figure 7.4: Measurements, prediction results and prediction errors for the Netflix scenario.



(a) Petclinic



(b) Netflix

Figure 7.5: Comparison of prediction results using DML and PCM.

application answers a few requests correctly before it turns on its fallback mode for a time interval to not crash on overload.

Based on the accuracy derived by related performance prediction approaches, such as [Bre16; Guo+13; Hub+10; Hub+17; LB16; Rat+12], our prediction results can be considered very accurate. Comparison of predicted performance metrics to predictions of related model extraction approaches is not straightforward and unfeasible. Closely related approaches rely on commercial and closed source software. Machine-learning-based approaches train using measurements of multiple system configurations (cf. Section 3.3). The analyses show that our approach can yield models that can accurately predict performance.

Besides, comparison of measurements and model-based predictions, the performance evaluation shows that both modeling formalisms provide equivalent performance prediction results. It is important to confirm that formalisms and analysis toolchains are interchangeable. Small differences can be explained by rounding errors and slightly different processing of job scheduling within simulators.

7.1.4 Evaluation of the Space of Complementary Modeling and Analysis Features due to Multiple Builder Implementations

Thus far, we presented how to derive core performance metrics supported by all performance analysis toolchains. The performance evaluation showed that both modeling formalisms provide similar performance prediction results. Hence, we can assume that analysis results are transferable to the other formalisms as well. Based on this assumption, we demonstrate additional benefits of the complementary use of toolchain specific modeling and analysis approaches. A major benefit of the PMX approach is that findings can be transferred to all formalisms and toolchains that provide a builder implementation. Solely through accessing two formalism toolchains, a wide variety of complementary analysis approaches can be triggered. Exemplary, we will present some formalism specific features in the following:

Empirical variable characterization using DML While the prediction accuracy for utilization is entirely accurate, some initial experiments showed deviations for response time predictions at high load. For the prediction of response times the distribution but also ordering of job inter arrivals matters. Many short arrival times in a row lead to increased response times. The workload models for load driver and simulators assume a homogeneous distribution of arrivals, while an investigation of execution traces showed non-uniformly distributed inter-arrivals. For example, the inter-arrival times for about 940 customers per second showed a mean of 12 ms and a standard deviation of 15 ms. In regular time intervals, the load generator sends bulks of requests followed by an interruption to fulfill configured arrival rate.

The challenge is that such bulk behavior cannot be uncovered just investigating the arrival distributions. Our initial assumption that request arrivals are independent is a simplification. Neither constant values nor custom distributions, like exponential distribution or normal distribution, can accurately depict the arrivals generated by the load driver. Inhomogeneous arrivals may occur in testbeds but may also occur on production systems. Hence, it is worthwhile to start more in-depth investigations. In contrast to its competing modeling approaches, DML allows for empirical variable characterization. We extracted arrival inter-arrival times from a given measurement scenario to generate a workload model that preserves sessions and accurately replays arrivals. In this experiment, we included the time spent at the workload generator into the simulation. Figure 7.6 shows that accuracy of the performance prediction improves when comparing the same load scenario. Our experiments using

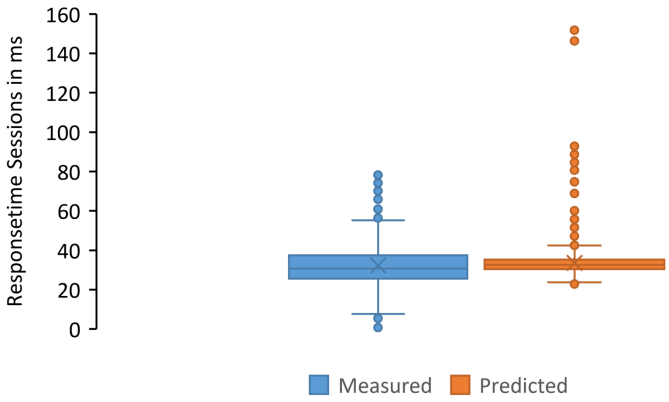


Figure 7.6: Response times box plots for measurement and replay of actual (non-uniformly distributed) arrivals at the model.

Workload in Requests per Second	Energy Consumption Measured	Predicted	Error
295	193.78	193.13	0.34%
715	217.00	221.37	2.01%
963	228.61	233.11	0.14%
1377	247.14	250.78	1.48%

Table 7.2: Energy consumption evaluation results for Petclinic case study. Consumption in Wh over an interval of 30 Minutes.

DML showed that the derived model can also predict individual requests and distributions accurately in addition to mean values.

Calculating energy consumption using PCM The Power Consumption Analyzer (PCA) enables software architects and system operators to reason the impact of design decisions and workload variations on power consumption. It integrates the design time power consumption analysis approach presented in [Sti+15] into the Palladio toolchain. At this, PCA leverages architectural performance models and combines them with power models to estimate power consumption. The accuracy of power consumption prediction highly depends on the accuracy of the used performance models. In order to evaluate feasibility of combining PMX and PCA for power prediction, we compare model-based predictions against power measurements for the Petclinic application. Table 7.2 lists the

predicted and measured total power consumption for different throughputs. The results show that the performance models learned by PMX can be applied for accurate prediction of power consumption.

7.1.5 Savings Through Framework Reuse

To evaluate savings of implementation effort, we compare the implementation effort of a builder to the effort of implementing model extraction for a specific formalism from scratch. Initially, PMX has been developed only for the PCM formalism. Despite expertise in software performance engineering and reuse of libraries where applicable, about twenty-four person-months of research and development went into the creation of stable software. Subsequently, we refactored our initial model extraction software to provide the model builder interface. Afterwards, we implemented a builder for the Descartes Modeling Language (DML). The implementation took about two person-months including testing.

We also started a builder implementation for the Performance Model Interchange Format (PMIF). However, we could not analyze resulting models due to the absence of a publicly available solver. In previous PMIF studies, the QNAP2 library has been applied for solving. However, the company developing QNAP2 has been bought up, and the closed-source solver is not publicly available anymore. For the PMIF builder implementation, the development time can be estimated upwards to a few days. As we could not analyze resulting models to verify correctness of the builder, we do not employ it to asses savings.

A builder implementation represents basically a model transformation that only considers a subset of operations that can be extracted from standard execution traces. The implementation effort depends on the similarity of models, the familiarity of developers with formalisms. Based on developer expertise, we quantify the effort ranging from two to four person-months which correspond to savings of 83% to 91% of developer time. The reusable PMX framework software has 9960 lines of Java code which corresponds to the code savings of applying our approach.

Besides a significant saving of time, we argue the application of the PMX framework also results in a reduced complexity. Developers only need to understand their modeling formalism instead of the logic behind all activities of the model extraction process.

7.1.6 Discussion

In contrast to application-specific model creation scripts, PMX generalizes the automated creation of architectural performance models while achieving accurate predictions. While model extraction tools without our approach have to be implemented from scratch, our approach lowers the implementation efforts for automated performance model creation tools by up to 91%. Using our framework, model extraction for a specific architectural performance modeling language requires only to implement the object creation routines of our builder interface. As an additional benefit, our approach simplifies the complementary use of multiple analysis toolchains for different performance modeling formalisms.

PMX is a generic library proving a core technology that enables a considerable amount of future research and industry applications. It can be applied to address manifold performance problems exceeding the presented evaluations. For example, [Rom17] proposes predefined templates of microservices architectural patterns and evaluates them based on a PCM performance model generated using PMX. His work investigates problems such as the service discovery for service registration and client-side load balancing for load distribution. Moreover, PMX has been integrated into the CASPA, which is a a platform for comparability of architecture-based software performance engineering approaches [Dül+17].

7.2 Decision Framework

In this section, we evaluate the framework for the automated selection of performance evaluation approaches. We demonstrate how to apply the methodology and tooling to provide automated decision support for performance evaluation approaches. Our evaluation orients at the reproducibility of state-of-the-art decision support for measurement-based and model-based evaluation approaches and the adaptability to the evolution of evaluation tools and comparison attributes. We refine our evaluation goals by the following research questions:

RQ1 Is our approach capable of depicting state-of-the-art automated decision support in the model-based analysis? (Section 7.2.1)

RQ2 Is our approach capable of providing automated decision support for the measurement-based analysis? (Section 7.2.2)

RQ3 Can our approach reflect the evolution of performance evaluation tools by adding, removing, or modifying performance evaluation strategies? (Section 7.2.1 and Section 7.2.2)

RQ4 Can our approach reflect the evolution of comparison attributes? (Section 7.2.1 and Section 7.2.2)

RQ5 Is our approach capable of depicting state-of-the-art automated decision support in the measurement-based analysis? (Section 7.2.3)

RQ6 What is the effort to come up and change capability models in comparison to tree-based decision support? (Section 7.2.4)

7.2.1 Decision Support for Model-based Performance Evaluation

We demonstrate the applicability of our approach for architectural performance models by comparing analysis approaches for the Palladio Component Model (PCM) formalism [BKR09]. We apply our methodology to architectural performance models as they use terminology similar to system architecture models. Supported solution strategies for PCM include SimuCOM, LQNS, SimQPN, and SimQPN Mean Value Analysis (MVA) [Bro+15]. SimuCOM transforms a PCM instance to a process-based discrete-event simulation supporting all PCM modeling concepts [BKR09]. A transformation to Layered Queueing Networks (LQNs) allows triggering the analytical LQNS solver [KR08]. A transformation to QPNs enables simulation and mean value analysis (MVA) using the SimQPN tool [MKK11].

Analysis approach	request class			accuracy	costs	limitations model	Case
	entity	metric	stat.		time-to-result		
SimuCOM	serv.	resp. time	sample	high	very high	-	tree
LQNS	serv.	resp. time	mean	high	very low	no loops, no fork-join, no param. dep., no blocking beh.	tree
SimQPN	serv.	resp. time	sample	high	medium	no loops, no fork-join, no param. dependencies	tree,3
SimQPN MVA	serv.	resp. time	mean	high	low	no loops, no fork-join, no param. dep.	tree,3
SimQPN parallel	serv.	resp. time	sample	high	very low	open workload, no loops, no fork-join, no param. dep.	2,3
JMT	serv.	resp. time	mean	high	medium	no loops, no fork-join, no param. dep., $\leq$64 job classes	1

Table 7.3: Capability models for model-based performance evaluation approaches.

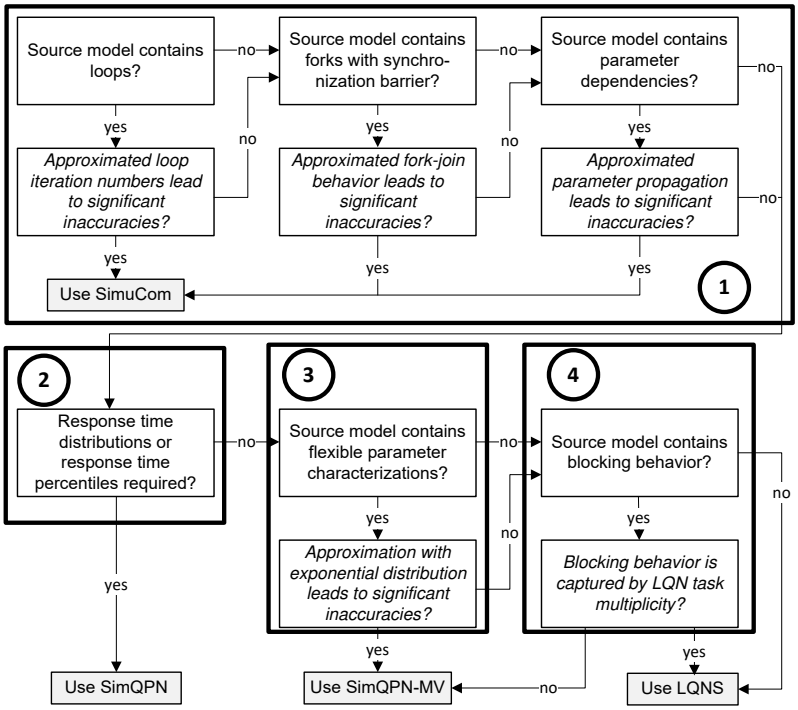


Figure 7.7: Solution strategy decision tree from [Bro+15].

Modeling In the following, we explain how we model capability models. Instead of performing separate experiments, we reverse engineer and decompose the decision tree presented in [Bro+15]. We identify four main steps as depicted in Figure 7.7. The results of reverse engineering the decision tree are shown in the capability models depicted in Table 7.3.

The first steps of the decision tree evaluate model capabilities that cannot be solved accurately by all investigated solution strategies that are based on transformations to stochastic formalisms. If the model contains loops, forks, or parametric dependencies that cannot be approximated, inaccuracies occur that make the approaches inapplicable. To model these limitations, constraints have to be integrated into the capability models of LQNS, SimQPN, and SimQPN MVA. We generalize limitations to apply for all occurrences of the respective modeling elements to enable fully automated decision support. In case the named limitations on modeling features do not apply for a given performance model, the decision tree proposes to use other analysis tools than SimuCOM. At this, the decision tree implicitly captures that SimuCOM performs poorly

regarding time-to-result. In contrast to the decision tree, our approach allows preserving this information within in the capability models explicitly.

Then, Step 2 of the decision tree states that when the distribution of response times is required, SimQPN shall be used. Accordingly, we append the statistic type sample to the capability model of the solution strategy SimQPN. SimQPN MVA and LQNS are only capable of deriving means, which can also be explicitly modeled.

Step 3 models a limitation of the transformations to LQNs which is not yet capable of carrying flexible parameter characterizations. Therefore, the containment of flexible parameter characterizations leads to significant inaccuracies when predicting performance indices using LQNS. Consequently, we define a constraint in the LQNS capability model.

Further, if the source model contains blocking behavior not captured by LQN task multiplicity (step 4), LQNS cannot be used, too. Consequently, we have to define an additional constraint for LQNS that checks if the source model contains blocking behavior not captured by the LQN task multiplicity.

Applying the decision engine presented in Section 6.3, the capability models depicted in Table 7.3 allow for the same selection of methods as in the decision tree depicted in Figure 7.7.

In extension to tree-based decision support, our approach allows separating applicability from cost and accuracy concerns. For example, when a system model contains loops, our approach proposes SimuCOM knowing that only SimuCOM is applicable. In the case of a scenario model that does not contain loops, forks or parameter dependencies, our approach states that SimuCOM and SimQPN are applicable and recommends SimQPN as it simulates faster. To fully automate the evaluation of limitations, we apply a simplifying generalization by restricting to *all* loops, branching actions, blocking behavior, and parametric dependencies, even though there might be analyses where respective entities have no significant effect.

Evolution Scenarios Providing capabilities to depict existing decision support, the primary benefit of our approach is that it provides flexibility for evolution scenarios, as demonstrated in the following use cases:

Case 1 *Adding a new solution approach.* While adding a solution approach to the decision tree would require to understand all approaches, our approach demands only to specify the capabilities of the new solution strategy. We added a transformation to Queueing Networks (QNs) and a subsequent simulation using JMT [BCS09]. It shares the limitations of the other model transformations such as concerning loop behavior. The original

development focus has been on manually created QN models instead of large-scale models deriving from model transformations from architectural performance models. Due to that, the solver implementation had a technical restriction to 64 job classes which prevented to simulate large-scale models (status as of January 2017). However, for small models, it performs better than SimQPN regarding time-to-result. The discussion triggered by the paper [WHK17] induced an JMT update that significantly increased the limit of job classes. We conclude that the naming of performance evaluation tool capabilities fosters competition and leads to improvements.

Case 2 *Adding a similar solver*. We reuse the transformation to QPNs and combine it with a parallel simulation using SimQPN [WSK15a]. Parallelization allows for significant speedup when the workload is specified as an open workload which is a prerequisite for efficient decomposition [WSK15a]. Capabilities for parallel and sequential simulation are very similar. This generally indicates hierarchical capability modeling which could be derived using refactoring.

Case 3 *Improvements of transformations*. Existing model transformations often do not support all features of the architectural source model. The restrictions are modeled as limitations. Extensions of transformations to support more source model features can be easily included in decision support by updating the respective capability models.
In general, there are transformation alternatives based on the same source model. For example, architectural models allow modeling deterministic loop counts. The corresponding model-transformations transform into probabilistic models for complexity reasons, but QPNs or multi-job-class QN would also allow for deterministic loop counts using multiple token colors or job classes.

Case 4 *Preselection*. Our approach enables easy pre-adjustments. Some application scenarios imply constraints known in advance that should not be evaluated on every request. A preselection of approaches for specific scenarios may be realized by registering only a subset of capability models. For example, knowing that only response time distribution will be requested, we can limit the registered approaches to SimuCOM and SimQPN. Having time constraints, one might select a subset of fast solvers.

7.2.2 Decision Support for Measurement-based Performance Evaluation

For measurement-based analysis, we could not find a popular decision tree to be refactored. In general, decision support for measurement-based analysis

Case				4	5	1,2,3,6
Analysis approach	request class			costs		limitations
	entity	metric	stat.	overhead	license	
DynaTrace	serv.	resp. time	sample	low	$	only Java, Node.js, .NET, MySQL, Python, Perl, Erlang, Ruby, iOS …
Instana	serv.	resp. time	sample	low	$	only .Net, Crystal, Go, Java, Node.js,PHP, Ruby, Scala
Kieker	serv.	resp. time	sample	medium	-	only Java, Perl, C#, VB6, Cobol, IEC61131-3
SPASS-meter	serv.	resp. time	sample	very low	-	sampling on high load, only Java

Table 7.4: Capability models for measurement-based performance evaluation.

approaches may use the same schema of accuracy, costs, and limitations as model-based analysis. However, comparison of measurement tools and approaches focuses on different features. While time-to-result is very relevant for model-based analysis, the time required for measurements is often similar for different measurement tools.

In contrast to model-based analysis, cost types describing financial expenses and system overhead predominate. Also, the validity period of decisions differs: the decision for a measurement infrastructure has a long-term character, as it is often a buy decision. For monitoring, system overhead plays a critical role. System monitoring approaches, like Magpie [BIN03] or X-Trace [Fon+07], are minimally invasive and target only network and operating system parameters. They come with the advantage of low system overhead but are not able to provide a view of internal application behavior. For Application Performance Management (APM) tools, the monitoring overhead is equal to the number of transactions times the number of measurement points divided by implementation overhead in seconds — depending on the actual monitoring tool implementation. Relative overhead of tools can be compared system independent; absolute monitoring overhead depends on transaction execution times of a concrete system and application scenario. Measurement-based approaches are usually not assessed or compared according to accuracy, as there is no ground truth like for model-based analysis. Therefore, we omitted a presentation of accuracy in Table 7.4 — which we would consider `very high` for all approaches.

Commercial monitoring frameworks often incur significant license costs, which makes them worthy to be included in the capability models of measurement-based approaches. In general, one could also model other cost types like hardware and system software required to operate, deployment and instrumentation costs, and ongoing maintenance costs. Monitoring tools are limited to the technology stacks for which they provide agents. A fundamental question is if a given monitoring tool can be applied to a specific system having specific technologies. The answer to this question can be modeled using limitations. Table 7.4 compares measurement-based analysis approaches according to the discussed capabilities.

We created the capability models based on information available in the referenced literature and performed no additional measurements. The selection has been made to demonstrate several concepts without making any claims to be exhaustive. The list contains commercial APM tools for use in production systems like DynaTrace and Instana. Research prototypes support fewer languages, libraries, and platforms but are usually open source. To reduce overhead,

sampling-based approaches have been proposed [BIN03; ES14; Sig+10]. SPASS-meter depicts an open-source implementation [ES14] while similar concepts run closed source at large providers like Microsoft [BIN03] or Google [Sig+10]. In the following, we discuss scenarios and explain how our decision engine answers or how to adjust to changes.

Case 1 *New Technology*: Consider a system or parts of it written in emerging technologies, like the new Go language. According to the modeled limitation, only Instana provides agents implemented for Go. Therefore, the decision engine proposes to use Instana.

Case 2 *Standard Technology*: The second use case considers a program written in a standard language, like Java. All investigated tools support Java. Therefore, the decision engine delivers the information that all measurement tools are applicable. Ordered according to license costs, it would recommend the open source Kieker framework or SPASS-meter.

Case 3 *Debugging*: Debugging requires accurate failure detection. This translates into denial of sampling. Detailed application traces are necessary to produce a comprehensive view of the monitored system. Applicable approaches include DynaTrace, Instana, and Kieker.

Case 4 *Managing overhead*: In a high load production system, the performance monitoring overhead is critical. In such a scenario, coarse-grained monitoring information on response times is sufficient. The decision support proposes an approach with low overhead, e.g., SPASS-meter.

Case 5 *Changes to pricing*: There is competition between APM vendors, which may result in changes to prices and pricing policy. The decision support can be updated by changing the cost specifications within capability models.

Case 6 *Expansion of supported technologies*: The competition between vendors also results in an increasing set of supported languages and technologies. Relevant changes could be reflected by adapting limitation definitions of capability models.

7.2.3 Holistic Decision Support for Measurement- and Model-based Performance Evaluation

Besides supporting separate decision support for model-based and measurement-based performance evaluation, the presented capability model approach can also provide combined decision support. The choice when to measure and when referring to model-based predictions is usually subject to a global “either-or" decision. To answer this choice flexibly has not been widely discussed due to the lack of an interoperability approach. Through our DPE approach,

we provide means to flexibility switch between both types of performance evaluation.

Decision support for model-based and measurement-based performance evaluation can be performed based on the capability model definitions presented in Table 7.3 and Table 7.4 with minor adaptations and extensions. The modeling of functional capabilities is the same for both analysis categories. While the comparison of most non-functional capabilities is straightforward, understanding of accuracy and time-to-result is different at measurement-based and model-based analysis disciplines.

The accuracy of model-based performance evaluation describes the deviation from the exact mathematically specified reference model solution. While measurements evaluate the system directly and can be assumed to be accurate, this perspective does not reflect the accuracy of how the system has been modeled. The accuracy of the model itself depends on how well model building and extraction techniques can capture the performance characteristics of a system. It depends, on the one hand, on the extraction mechanisms and, on the other hand, on the characteristics of the analyzed system. Also, for response time, deviations between measurements and model-based predictions at high system load are more perceptible than at low load. To reflect this within tool capability models, one has to set up a scenario specific cost evaluation function. Another option, more comfortable to implement, would be to adjust accuracy scores of model-based performance prediction approaches by subtraction or multiplication of a correction factor. Additional research has to be performed to select a suitable strategy for the integration of inaccuracies within models. However, our approach offers different ways to integrate such.

Another challenging comparison attribute is time-to-result. Several comparisons of time-to-result for model-based approaches have been performed (cf. Section 3.2.1). However, there is no natural link to the time required for measurement-based performance evaluation as performance analysts commonly configure manually. The choice is mostly like the following: In case additional measurements are required, model-based performance evaluation usually outperforms measurements regarding time to result. A qualitative comparison based on "very slow" for measurements can be sufficient for many application scenarios. However, this depends on the experiment setup.

The applied cost type depends on organizational and evaluation scenario specific needs. As an alternative to the time-to-result comparison, required `infrastructure costs` depict another suitable cost-type to compare measurement-based and model-based performance evaluation approaches.

Besides triggering measurement-based and model-based performance eval-

uation, approaches learning from on previous performance evaluations can be applied. If metrics of a system configuration have already been requested, historical monitoring logs could be queried which provides accurate values at no system overhead costs. Based on traces of similar configurations, also predictions based on transfer learning could provide desired metrics.

7.2.4 Efforts for Modeling

Initialization The most significant effort setting up decision support is to gain insights into the performance evaluation approaches by performing experiments, investigating performance evaluation code, and extensively searching literature. This effort includes the evaluation of how evaluation approaches behave on specific system characteristics not documented or even not intended by their developers (e.g., particular model sizes for model solvers). The effort for this challenging knowledge retrieval process, which highly benefits from an educated workforce, is comparable when creating static decision trees or applying our decomposed approach.

The effort to implement collected knowledge is low compared to the learning process. Compared to hardcoding a decision tree, we assume a similar technical effort when initializing decision support applying our presented framework. The implementation of evaluation functions is the same for tree-based and our decoupled decision support. In contrast to hardcoding a decision tree, our meta-model decoupling allows benefiting from a divide-and-conquer strategy. The capability modeling approach has the advantage that one may distribute modeling to multiple employees along capabilities and evaluation approaches. This may significantly reduce the complexity assigned to a single workforce. Besides facilitation to provide decision support of state-of-the-art complexity, this would enable for more comprehensive decision support containing an expanded set of evaluation approaches, additional comparison attributes, and more detailed limitations. Comprehension of framework interfaces and its technical integration can be learned within a *few days* which is the *only additional cost* of applying the presented approach.

Overall, the level of detail in decision support profoundly affects its setup efforts. For example, the initial setup effort depends if costs are quantified globally (which we did for the presented studies) or specific to system and analysis configuration (which we separately discuss in Section 7.3.3). For presented case studies, the time to create capability models for the architectural model case study took about two weeks and for measurement capability models took less than one week. We assume that for less-experienced workforce efforts can be higher.

Evolution	Minimum Number of perf. eval. approaches to investigate	
	decision tree	capability model
Update of analysis costs	[2 ... #approaches]	1
Add new metric	[#approaches supporting that metric]	1
Add new approach	[2 ... #approaches]	1
Remove approach	[2 ... #approaches]	1

Table 7.5: Comparison of potentially affected performance evaluation approaches on evolution scenarios for decision trees and decoupled approach.

Long-term Maintenance In general, we assume that there is an iterative refinement and update process when creating decision support systems for performance evaluation approaches. Commonly, performance evaluation tools target a specific application scenario where others do not perform well. Later, it might be discovered that such tools perform poorly in not intended application scenarios. Some limitations of performance evaluation tools appear when expanding their application space. This can be illustrated by an anecdote: In [WHK17], we named an upper bound of job classes limitation in the JMT implementation. Already during the paper review process, I was asked for an appropriate value for a new upper bound for job classes in JMT tool. Purely editor-based model creation limits the realistic number of job classes. The JMT developers could not imagine creating more than 64 job classes manually. Moreover, developers are mostly not interested in highlighting weaknesses, and external end users typically do not have insights into code-level. Due to the complexity of differences in performance evaluation approaches, we propose to pragmatically capture a sufficient level of detail and iteratively improve rather than aiming for the perfect modeling from the start. Even sparsely defined capability models may still provide benefits when allowing to exclude certain kinds of performance evaluation approaches.

To reflect adaptations to performance evaluation tooling in a static decision tree may require to inspect all of its nodes in the worst case. Table 7.5 compares the number of evaluation approaches to be inspected of our decomposed methodology to the tree-based approach. The assumption is that each node within a decision tree encapsulates the reason for the decision. Otherwise, maintaining tree-based decision support requires considerably more efforts. If

decision trees do not capture the reason for branching, there is no structured process (expect for traversing the tree) on node modification, addition, or removal. Moreover, there can be a collision of an update, appending, or removal of a performance evaluation method with a decision not captured in the tree. For example, removing a fast performance evaluation forces to decide which of the remaining approaches is the fastest for the unleashed non-answered application scenarios. Figuratively, the tree may shade some decisions which may recur on evolution which may even force to rebuild the whole tree.

7.2.5 Discussion

We demonstrated how to provide decision support for model-based, measurement-based, and holistic performance evaluation comparing both among each other. We can provide the same decision support as state-of-the-art tree-based solutions with comparable setup efforts. In contrast to tree-based decision support, we demonstrated how to efficiently cope with evolution scenarios like modifying characteristics, as well as adding or removing performance evaluation approaches.

7.3 Evaluation of Time-to-Result Estimation Accuracy

In this section, we evaluate accuracy and efforts for the machine-learning-based time-to-result prediction approach for model-based performance evaluation presented in Section 6.4. To this end, we apply the performance model and analysis features derived in Section 6.4.2 to feed a set of machine-learning algorithms. We refine our evaluation goals by the following research questions:

RQ1 How accurate are time-to-result predictions when applied to an unknown performance model? (Section 7.3.2.1)

RQ2 How many training models are required for accurate predictions? Can the prediction process be progressive? (Section 7.3.1)

RQ3 How accurate is time-to-result for different machine-learning algorithms? (Section 7.3.1)

RQ4 How accurate are predictions for analysis configurations where variations of performance analyses for the considered model have been used for training? (Section 7.3.2)

RQ5 What is the *initial* effort for prediction of analysis response times? (Section 7.3.3)

RQ6 What is the effort for the prediction of a single analysis time? (Section 7.3.3)

We conducted two case studies to evaluate the prediction accuracy of our approach. The first considers unseen performance models, while the second includes performance evaluations of the considered model into the training data.

Experimental Setup

At first, we evaluate the prediction accuracy for performance models not present in the training data. The presented experiment uses permutations of the three-tier architecture, Petclinic, and hello world models (introduced in Section 6.4) for training to predict four variations of SPECjEnterprise2010 analysis. Based on performance-relevant features, we analyzed model and analysis variations of the training models resulting in 345 training datasets including a cross product of variations. Table 7.6 summarizes the experiment results showing Mean Absolute Percentage Error (MAPE) and Root Mean Standard Error (RMSE). We present only a subset of prediction settings which we consider the most

Table 7.6: Time-to-result prediction errors for different training datasets.

Dataset		Mean time to result	Prediction error for time-to-result in seconds (RMSE) and mean absolute percentage error (MAPE)					
			RegressionTree		RandomForest		GradientTreeBoost	
Training dataset	Evaluation dataset		RMSE	MAPE	RMSE	MAPE	RMSE	MAPE
3Tier	SPECjEnterprise2010 Variation 1: unconstrained query	7.52	1.81	24.11	0.7	9.26	0.2	2.46
HelloWorld			9.29	123.56	7.92	105.43	9.16	121.85
PetClinic			114.05	1516.8	88.16	1172.48	113.51	1509.6
PetClinic,3Tier			1.57	64.65	0.47	6.29	1.08	14.45
PetClinic, HelloWorld			4.45	59.24	8.66	115.2	8.3	110.45
HelloWorld, 3Tier			1.81	24.11	5.24	69.76	1.47	19.57
PetClinic,HelloWorld,3Tier			0.91	12.10	4.75	63.27	3.38	45.06
Lowest error across dataset combinations			0.91	12.10	0.47	6.29	0.2	2.46
3Tier	SPECjEnterprise2010 Variation 2: constrained as fast	7.44	1.22	16.42	0.55	7.37	0.05	0.68
HelloWorld			9.4	126.2	8.25	110.85	9.16	123
PetClinic			113.86	1528.69	86.88	1166.47	113.62	1525.56
PetClinic, 3Tier			1.22	16.42	1.68	22.64	0.78	10.48
PetClinic, HelloWorld			4.52	60.76	8.38	112.54	8.37	112.43
HelloWorld, 3Tier			4.49	60.34	5.69	76.45	2.15	28.96
PetClinic,HelloWorld,3Tier			0.79	10.72	5.69	76.48	3.42	45.99
Lowest error across dataset combinations			0.79	10.72	0.55	7.37	0.05	0.68
3Tier	SPECjEnterprise2010 Variation 3: population set from 80 to 160	14.68	6.01	40.96	6.65	45.27	7.32	49.86
HelloWorld			2.12	14.44	0.86	5.9	2.02	13.77
PetClinic			106.88	727.68	81.71	556.31	105.93	721.19
PetClinic, 3Tier			6.01	40.96	4.14	28.20	6.02	41.05
PetClinic, HelloWorld			2.12	14.44	4.32	29.45	1.13	7.74
HelloWorld, 3Tier			2.74	18.69	2.33	15.91	5.91	40.25
PetClinic,HelloWorld,3Tier			2.74	18.69	0.51	3.47	3.89	26.49
Lowest error across dataset combinations			2.12	14.44	0.51	3.47	1.13	7.74
3Tier	SPECjEnterprise2010 Variation 4: four interconnected components	10.11	0.72	7.16	0.89	8.84	2.67	26.43
HelloWorld			5.76	56.94	5.34	52.84	6.54	64.65
PetClinic			111.08	1097.92	83.79	828.13	110.44	1091.53
PetClinic,3Tier			1.27	12.55	0.97	9.65	1.41	13.94
PetClinic, HelloWorld			1.85	18.34	10.8	107.14	5.71	56.49
HelloWorld, 3Tier			0.72	7.16	0.39	3.81	2.05	20.34
PetClinic,HelloWorld,3Tier			0.4	4	0.2	1.98	0.5	4.93
Lowest error across dataset combinations			0.4	4	0.2	1.98	0.5	4.93

challenging to predict. We omit additional presentation of permutations of prediction and training data, as the prediction accuracy does not change significantly. For scenarios not presented here, errors were below the ones we present.

7.3.1 Required Number of Training Models

Most statistical learning techniques can make more accurate predictions when more data is available [HTF09].

Hypothesis Prediction accuracy improves when a larger training sample is available (for RQ 2).

Methodology To evaluate the effect of a different amount of training data, we train separate prediction models using all subsets of training data separately.

As presented in Table 7.6, the prediction based on the input of three training models yields stable predictions for all scenarios. For all variations of training models, we obtained the worst prediction using a single training model. The prediction is outperformed by the worst score for two models, which is again outperformed by the scores for three training models. Exemplified by numbers from the prediction of `Variation 1` using a regression tree: the RMSE metric decreases from 114, to 4, to 0.9 for more training models. The MAPE decreases in the same manner from 1516, to 590, to 12. Thereby, our experiments confirm the assumption that adding more models improves the prediction accuracy in general. However, increasing the number of learned models does not always imply an improvement in the accuracy of predictions unless the new learned model has similar complexity to the unknown model. It is evident for training datasets containing a combination of two learned models since they do not in all cases outperform the training datasets consisting of only one model. For example, considering the `Variation 1` and 2 of the validation datasets, the training dataset obtaining the best accuracy consists only of the three-tier architecture dataset. Validating against more complex variations of the validation dataset (`Variation 3` and `Variation 4`), the training dataset consisting of all three learned models provides more accurate predictions than all partial training datasets.

7.3.2 Choice of Machine-Learning Algorithms

There are several machine-learning algorithms capable of solving the non-linear regression problem.

Hypothesis One of the selected learning algorithms outperforms others regarding prediction accuracy (RQ 3).

Methodology In this experiment, we evaluate the prediction accuracy of regression tree, random forest, and gradient tree boost.

Based on the results presented in Table 7.6, all prediction algorithms have fluctuating accuracies. For the investigated setting, there is no machine-learning algorithm always providing superior predictions. However, regression tree outperforms the other two algorithms providing mostly the lowest error rates and has fewer outliers. Based on all training models, regression tree (MAPE is between 4% and 19%) offers slightly improved error scores compared to random forest and gradient tree boost (MAPE between 2% and 77%).

7.3.2.1 Prediction Accuracy for Unknown Models

Based on the results from the previous two experiments, we identified the regression tree and the set of three training models to be the most appropriate prediction setting. A determination of the best setting allows testing the following hypothesis:

Hypothesis The presented approach is effective for predicting the time-to-result for performance models not present within the training data (RQ 1).

Methodology We evaluate the prediction accuracy based on a regression tree trained using a training dataset consisting of all experiment models except for the predicted SPECjEnterprise.

The evaluation results depicted in Table 7.6 show a MAPE for the prediction between 4% and 19%, and a RMSE between 0.4 and 2.74 seconds. We consider this to be sufficiently accurate based on the fact that simulation time may vary for different random seeds.

7.3.2.2 Prediction Accuracy for Known Models

The same base model can be analyzed using different analysis configurations. For the time-to-result prediction of a previously unmeasured analysis configuration, we consider a model to be known if the training dataset includes variations of it. While time-to-result prediction for unknown models poses the greater challenge, the prediction for known models depicts an everyday use case when updating the prediction model for new analyses.

Hypothesis The prediction accuracy improves for variations of a model already present in the training data (RQ 4).

Methodology The knowledge that a model has already been analyzed using different analysis configurations can be used for a reduction of the train-

Table 7.7: Prediction errors for time-to-result prediction of analyses on known models.

Dataset		Iteration	Prediction error for time-to-result in seconds (RMSE) and mean absolute percentage error (MAPE)					
			RegressionTree		RandomForest		GradientTreeBoost	
Training dataset	Evaluation dataset		RMSE	MAPE	RMSE	MAPE	RMSE	MAPE
HelloWorld, ThreeTierArchitecture	Data fold excluding the current iteration	1	0.06	1.38	1.13	45.59	2.32	83.8
		2	0.08	0.48	0.91	9.12	1.16	8.03
		3	0.07	1.56	0.99	57.79	1.72	81.46
		4	0.06	3.17	1.02	60.82	2.35	> 100
		5	0.26	2.86	1.17	39.53	2.23	75.58
		6	0.05	0.56	0.95	31.33	1.7	61.68
		7	0.11	1.39	0.78	46.87	1.61	73.69
		8	0.05	0.34	0.81	13.6	1.1	18.07
		9	0.11	1.81	1.27	24.19	1.99	47.32
		10	0.08	1.93	0.7	31.82	0.15	51.85
Mean error across iterations			0.09	1.54	0.97	36.06	0.68	60.91

ing dataset. Exemplary, we generated a set of analyses for HelloWorld and ThreeTierArchitecture. The model variations and time-to-result measurements sum up to a total of 230 analysis tuples, 115 per model. Then we partitioned the analysis tuples into ten folds and performed cross-validation to evaluate the prediction accuracy. Iteration one excludes the first fold from training; iteration two excludes the second fold, etc. Each iteration tries to predict response times for its corresponding analysis configurations.

Table 7.7 summarizes the prediction error scores obtained by the different machine-learning algorithms across the iterations of the cross-validation process. Compared to the prediction for unknown models, the error scores can be significantly improved. Regression tree outperforms both random forest and gradient tree boost with a mean RMSE of 0.09 seconds and a mean MAPE of 1.54%. Compared to the prediction of unknown models, there is no significantly outperforming machine-learning algorithm. In terms of RMSE, random forest presents a worse prediction accuracy with a mean deviation of 0.97 seconds. Considering MAPE, the gradient tree boost algorithm obtains the worst error score with a mean of 60.91%. This can be explained by the fact that RMSE is more sensitive to outliers than error functions based on the absolute error.

7.3.3 Effort

Prediction approaches can be evaluated according to prediction accuracy and overheads. While the previous sections investigated accuracy, we discuss initialization efforts and the effort for a single time-to-result prediction using our machine-learning-based prediction methodology in the following.

Initialization We first discuss the initialization effort to set up time-to-result predictions (RQ 5). Including implementations, setting up training data generation for the specific exemplary DML solution process took several weeks. Despite using machine-learning, our approach still requires insights into the solution approach to select appropriate features for prediction. Compared to architectural performance models, pure stochastic formalisms like, e.g., QNs, come with less potential features, which might speed up the setup process. Based on a single analysis approach, we cannot provide a meaningful quantification of efforts. This would require studies investigating additional solution strategies. However, compared to white-box modeling, machine-learning does not require to understand the internal behavior of the analysis approach completely. As a consequence, the human effort for creating a black-box machine-learning model is significantly lower compared to white-box modeling. Therefore, we

can appraise lower efforts even without creating a parallel white-box modeling study to compare the initialization effort.

Single Prediction The effort for the prediction of a single time-to-result (RQ 6) using our approach is very low. Predictions are always within few milliseconds. Moreover, regression-based algorithms provide good scalability if we would consider appending additional features. We argue that known white-box parametric model solution processes cannot outperform analysis times of regression trees Existing solutions are built on internal model transformations and worse scalability. The analysis time of white-box models is usually above a second. In contrast, each prediction using our approach can be provided within few milliseconds. Based on related research [ZT07] and several years of expertise in performance modeling, we can appraise superiority regarding execution times for our approach even without additional white-box modeling experiments.

7.3.4 Discussion

We apply the prediction methodology to a representative model-based performance prediction approach and evaluate the prediction accuracy for a selection of machine-learning algorithms based on previously deduced influencing factors. The evaluation considers predictions for models not included in training data and for models where alternative analysis configurations of the same model have been used for training. The evaluation results obtained in this study show that based on a small set of training models, accurate time-to-result predictions can be achieved. The prediction for unknown models exhibits a mean deviation below half a second and a mean percentage error below 20%. The accuracy can be significantly improved when predicting time-to-result for analysis configurations of previously analyzed models.

Internal Validity To increase internal validity, we use mean values of multiple analysis runs. Even using a limited number of base training models, accurate time-to-result predictions can be achieved. We expect that using more training models would provide even more accurate prediction results. In our experiments, regression tree performed significantly better compared to random forest and gradient tree boost. These claims are limited by the fact that we had only a small number of performance models for evaluation. The choice of training models influences the selection of performance influencing factors and the quality of the prediction model. We cannot state if analyzing new models would require to modify the features.

External validity Based on the experiments, claims can only be made for simulation-based solvers as we expect them to have similar performance behaviors. A generalization to analytical solvers has not been investigated in the presented experiments. To support claims about further model-based performance evaluation approaches, we plan to investigate more model-based analysis approaches in the future. Based on existing experiments, we cannot claim portability of prediction accuracy for other model-based analysis approaches yet.

7.4 Evaluation of Performance Concern Language and Processing Framework

This section evaluates the performance concern language and its corresponding processing framework presented in Chapter 4 which targets holistic performance management throughout the whole software engineering life-cycle using an integrated and light-weight approach. We evaluate based on the following requirements (introduced in Section 4.1):

- *"I would like to evaluate performance with different evaluation methods"* (Requirement 1)
- *"I would like to evaluate various kinds of performance concerns."* (Requirement 2)
- "*I would like to specify non-functional processing goals on time-to-result and accuracy which automatically trigger customized processing.*" (Requirement 3)
- "*I would like to evaluate performance throughout the whole software engineering life-cycle using an integrated approach that requires reasonable effort.*" (Requirement 4)

These requirements guide the evaluation refined by the goals to evaluate applicability, the efficiency in concern formulation, the flexibility of analyses, and the effort to perform such (which is level of automation). To assess the methodology and tooling, we refine the goals by the following research questions:

RQ1 *How can the declarative methods be applied to the software development life-cycle of real-world applications?* (Section 7.4.1)

- RQ1.1 *Can the declarative approach support the whole software development life-cycle?*
- RQ1.2 *What are differences of measurement-based and model-based performance evaluation?*
- RQ1.3 *What are reasonable guidelines to parameterize quality of service guarantees based on performance evaluation results?*

RQ2 *Can the declarative approach automatically translate non-functional tradeoffs into reasonable processing configurations?* (Section 7.4.2)

- RQ2.1 *Can we provide reasonable tradeoff decisions for model-based performance evaluation?*

RQ2.2 *What is the difference in time-to-result between model-based and measurement-based performance evaluation?*

RQ2.3 *How much savings of time-to-result can be achieved at generic concern processing? For which types of concerns can we derive savings on framework-level?*

RQ3 *How does the declarative approach facilitate and accelerate to achieve holistic performance awareness?* (Section 7.4.3)

RQ3.1 *What is the effort to switch between measurement-based and model-based performance evaluation?*

RQ3.2 *What is the effort to evaluate various kind of performance concerns?*

RQ4 *What are the savings when connecting a performance evaluation approach to the generic declarative language framework?* (Section 7.4.4

In Section 7.4.1, we demonstrate how the declarative performance engineering framework behaves in the field by presenting multiple kinds of analyses using different evaluations techniques based on two case study systems. Section 7.4.2 evaluates the implementation of tradeoff processing. Section 7.4.3 evaluates the implementation of tradeoffs. Section 7.4.4 reviews the potential for savings when integrating a performance evaluation technique. Finally, Section 7.4.5 discusses threats to validity cross-cutting for all research questions.

7.4.1 Demonstration of Software Life-Cycle Integration

We demonstrate the formulation of performance concern and their automated evaluations using measurement-based and model-based performance evaluation methods based on the case study systems applied for performance model extraction (presented in Section 7.1). To provide a concise evaluation, we aim to avoid repetitive presentation of redundant concerns. In the following, we illustrate applying our approach to solve performance concerns that typically appear within the software development life-cycle. In particular, we demonstrate comparative evaluations, SLA life-cycle, and optimization.

Performance evaluation artifacts change during software life-cycle which demands to alternate between different kinds of evaluation. Even without implemented software, software systems can be modeled and parameterized using estimated resource demands which has been proposed by the model-driven engineering community. Early model creation can be connected to significant efforts *and* a vague prediction capability of the model. To save efforts, we recommend performing model-based performance engineering activities when

at least parts of the system implementation are available. A runnable system allows triggering measurement-based performance evaluations which can be used to construct performance models automatically. Based on a single run, one may extract a performance model from application performance execution traces.

Comparative Evaluations

Performance evaluation results have to be compared in various decision processes during the software life-cycle. Measurements performed on software systems are subject to variations. Listing 7.1 shows how to compare response times for two measurement runs in a peak load scenario using our language and Figure 7.8 presents results.

Listing 7.1: Comparison of response times for two measurements runs of the Petclinic application.

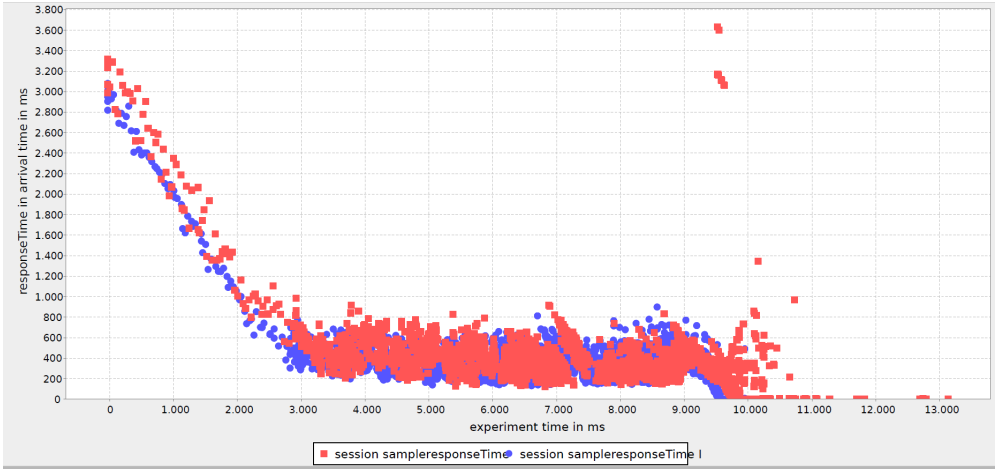
```
SELECT
   session.responseTime
VARYING 'iteration'<1,2>
FOR
   SERVICE 'session' AS session
USING kieker@'petclinic.properties';
```

The flexibility in changing visualizations provided by our declarative framework supports the understanding of performance behavior. While the number of data points in a scatter plot can make it challenging to compare multiple evaluation results and data points may mask each other, the reduced box plot visualization does not suffer from elusive comparison. However, the box plot visualization hides that there is a temporal progression of response times. Equipped with comprehension of deviations of individual requests helps to classify performance regression and accuracy of model-based predictions later.

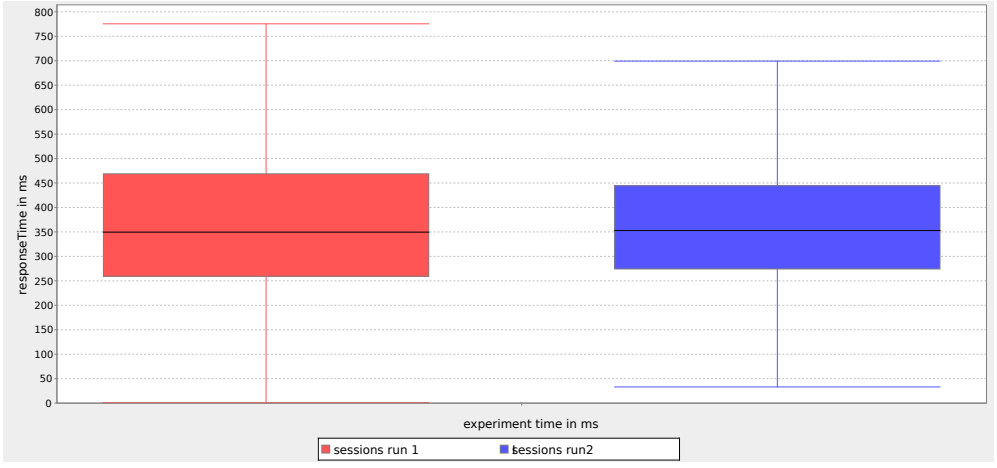
Listing 7.2: Comparison of response times for different arrival rates.

```
SELECT
   getRSS.responseTime
VARYING 'arrival rate' <50, 75, 100, 150, 300, 400, 600>
USING kieker@'netflix.properties';
```

Listing 7.2 presents a concern comparing the response times for different arrival rates, and Figure 7.9 presents the corresponding result visualization for measurement-based and model-based performance evaluation. It shows that

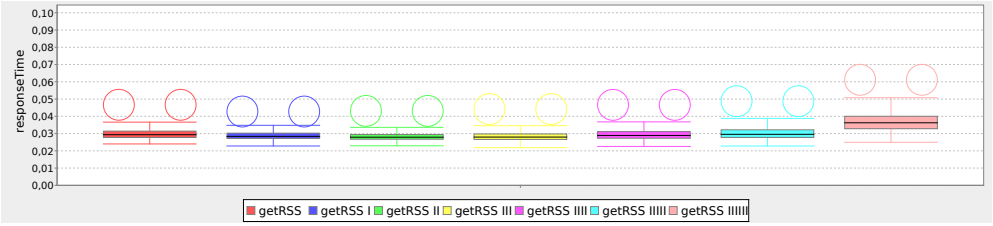


(a) Scatter plot

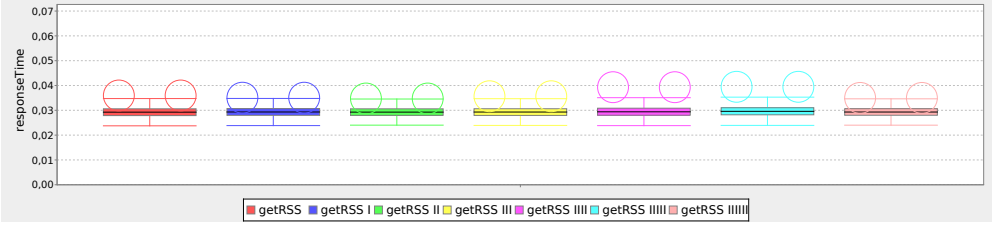


(b) Box plot

Figure 7.8: Variation of result visualization for two measurement runs with the same peak load on Petclinic system for concern presented in Listing 7.1.



(a) Measurements



(b) Model-based predictions

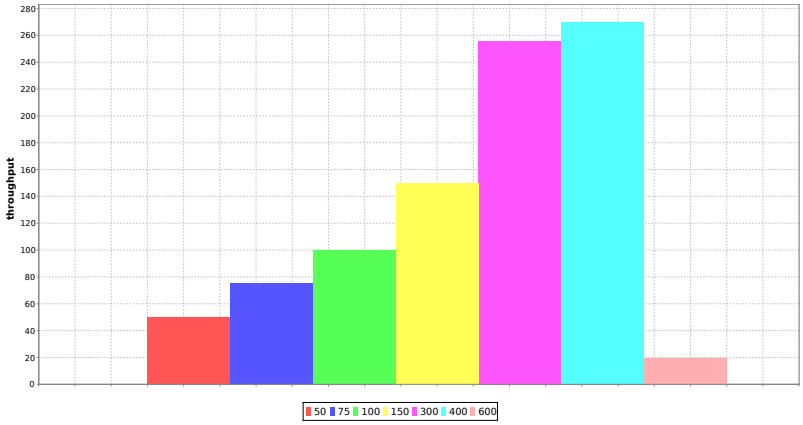
Figure 7.9: Measured a and predicted b and response times for 50, 75, 100, 150, 300, 400, 600 requests per second for the Netflix RSS reader as box plots.

the performance model well captures not only means but also the upper end of the response time distribution. The reproduction of upper-end behavior is very relevant for SLAs evaluation mostly focusing on the upward spike behavior.

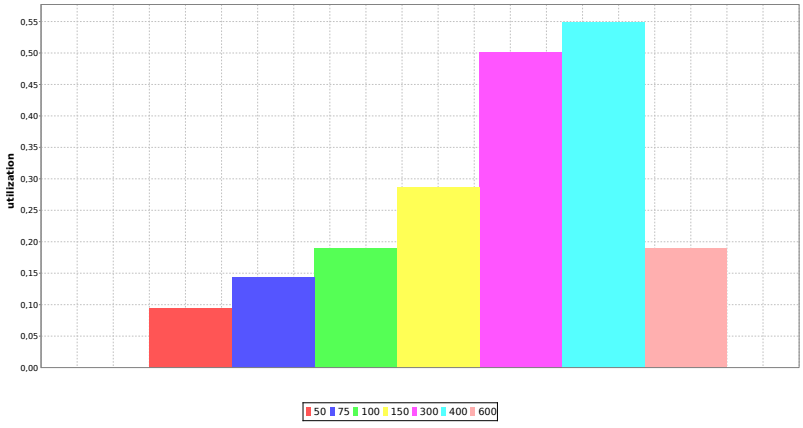
Our declarative framework allows comparing arbitrary changes to system and workload easily. For example, one can compare different revisions of a software system based on commit hash value. The only requirement is to implement adaptation operations in the adapter once. Primarily, the adapter for model-based performance evaluation provides a notable set of degrees of freedom based on the modeling language so that we can also evaluate system variation using language support. Without our approach, the system configuration would have to be changed manually at a very technical level. Therefore, our integrated and automated approach saves time and efforts compared to manual modification of system or model description.

SLA Life-Cycle Support

SLAs define guarantees for an upper usage behavior when having not less than a minimum amount of available resources. During software system life-cycle, SLAs have to be evaluated for different system settings and may require adaptation on system evolution.



(a) Throughput



(b) Utilization

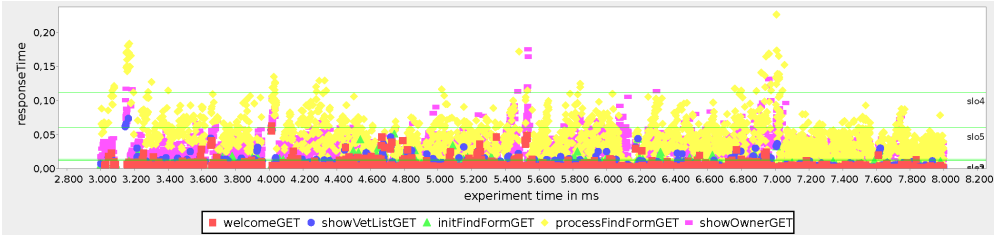
Figure 7.10: Predicted throughput and utilization for 50, 75, 100, 150, 300, 400, 600 requests per second for the Netflix RSS reader. Both decrease at high load. The overprotective dropping behavior can be well reflected in the model.

Listing 7.3: Concern to generate an SLA specification for the Petclinic system for 40 cores and a workload of 130 continuously session-firing threads.

```
GENERATE <MULTIPLY 1.2> SATISFACTION-LEVEL 95.0 PERCENT
VARYING 'cores'<40>, 'workload threads'<130>
FOR
   SERVICE 'WelcomeControllerHTTP.welcomeGET()' AS welcomeGET,
   SERVICE 'VetControllerHTTP.showVetListGET(java.util.Map)' AS
       showVetListGET,
   SERVICE 'OwnerControllerHTTP.initFindFormGET(java.util.Map)'
       AS initFindFormGET,
   SERVICE 'OwnerControllerHTTP.processFindFormGET(
       org.springframework.samples.petclinic.model.Owner,
       org.springframework.validation.BindingResult,
       java.util.Map)' AS processFindFormGET,
   SERVICE 'OwnerControllerHTTP.showOwnerGET(int)' AS showOwnerGET
USING kieker@'petclinic.properties'
```

Listing 7.4: Generated SLA Concern for the Petclinic system.

```
EVALUATE
AGREEMENTS
   sla1 CONTAINS (slo1,slo2,slo3,slo4,slo5)
GOALS
   slo1: welcomeGET.responseTime<0.012 SATISFACTION-LEVEL 95.0
       PERCENT,
   slo2: showVetListGET.responseTime<0.014 SATISFACTION-LEVEL
       95.0 PERCENT,
   slo3: initFindFormGET.responseTime<0.012 SATISFACTION-LEVEL
       95.0 PERCENT,
   slo4: processFindFormGET.responseTime<0.11 SATISFACTION-LEVEL
       95.0 PERCENT,
   slo5: showOwnerGET.responseTime<0.06 SATISFACTION-LEVEL 95.0
       PERCENT
FOR
   SERVICE 'WelcomeControllerHTTP.welcomeGET()' AS welcomeGET,
   SERVICE 'VetControllerHTTP.showVetListGET(java.util.Map)' AS
       showVetListGET,
   SERVICE 'OwnerControllerHTTP.initFindFormGET(java.util.Map)'
       AS initFindFormGET,
   SERVICE 'OwnerControllerHTTP.processFindFormGET(
       org.springframework.samples.petclinic.model.Owner,
       org.springframework.validation.BindingResult,
       java.util.Map)' AS processFindFormGET,
   SERVICE 'OwnerControllerHTTP.showOwnerGET(int)' AS showOwnerGET
USING kieker@'petclinic.properties'
```



(a) Scatter plot visualization enriched by SLO thresholds.

SLA Performance Summary

Execution comlies to sla1

Execution comlies to slo3 3.803921568627451% violations (97 in total) for 5.0% violations allowed
Execution comlies to slo4 1.5760441292356187% violations (40 in total) for 5.0% violations allowed
Execution comlies to slo1 3.549141965678627% violations (91 in total) for 5.0% violations allowed
Execution comlies to slo2 3.868698710433763% violations (99 in total) for 5.0% violations allowed
Execution comlies to slo5 2.5019857029388404% violations (126 in total) for 5.0% violations allowed

(b) SLA Evaluation Report.

Figure 7.11: SLA evaluation results using a generated SLA for the Petclinic system.

Model-based analysis and measurements can be used to parameterize SLA thresholds. Experiments triggering measurements can be time intensive and require an expensive setup. Deriving SLAs for different settings and usage behaviors can be cumbersome when only based on measurements. For analyzing substantial parameter spaces, the model-based analysis allows for faster performance evaluation.

The concern in Listing 7.3 generates an SLA definition for the Petclinic system for 40 cores and a workload of 130 continuously session-firing threads. The variation clause can also be used to generate multiple SLAs, e.g., for different maximum arrival counts 100, 200, 400, 1000. It considers all services trigger by the workload description. The SLA generation specifies no penalties as they do not relate to the system and the business department and can add them later. Thus, this definition can be very concise. Listing 7.4 shows the generated SLA concern and Figure 7.11 its evaluation results.

Listing 7.5: Concern to generate an SLA specification for 100 arrivals per second for the Netflix RSS reader.

```
GENERATE <MULTIPLY 1.2> SATISFACTION-LEVEL 95.0 PERCENT
VARYING 'arrival rate' <100>
FOR
   SERVICE 'com.netflix.recipes.rss.hystrix.GetRSSCommand.run()'
       AS getRSS
USING kieker@'netflix.properties';
```

Listing 7.6: Generated SLA specification for the Netflix RSS reader based on the concern in Listing 7.5.

```
EVALUATE
AGREEMENTS
  sla1 CONTAINS (slo1)
GOALS
  slo1: getRSS.responseTime<0.039 SATISFACTION-LEVEL 95.0 PERCENT
VARYING 'arrival rate' <100>
FOR
  SERVICE 'com.netflix.recipes.rss.hystrix.GetRSSCommand.run()'
      AS getRSS
USING kieker@'netflix.properties';
```

For the Netflix system, we experimented with SLA parametrization and evaluation. Listing 7.5 shows SLA generation based on measurements for 100 arrivals per second to which we apply an offset of twenty percent. The generated SLA concern, specifying an upper limit of 0.039 seconds for response times, is depicted in Listing 7.6. We derived *the same* threshold applying the model-based predictions for SLA generation. In general, such a match cannot be assumed and highly depends on how well the model fits the real system. However, it illustrates that the declarative method can be successfully applied to parameterize SLAs.

The SLA evaluation results, depicted in Figure 7.11 and Figure 7.12, illustrate insights and discoveries we made for SLA parametrization based on performance evaluation results:

NO 100 % Guarantees The SLA evaluation of a considerable measurement interval can still be subject to small variations. In general, extrema evaluation is more sensitive than evaluation of mean values. Therefore, a guaranteed satisfaction level of hundred percent is not appropriate and exceeds actual response times by far.

Rounding Performance evaluation mostly can be more accurate when investing

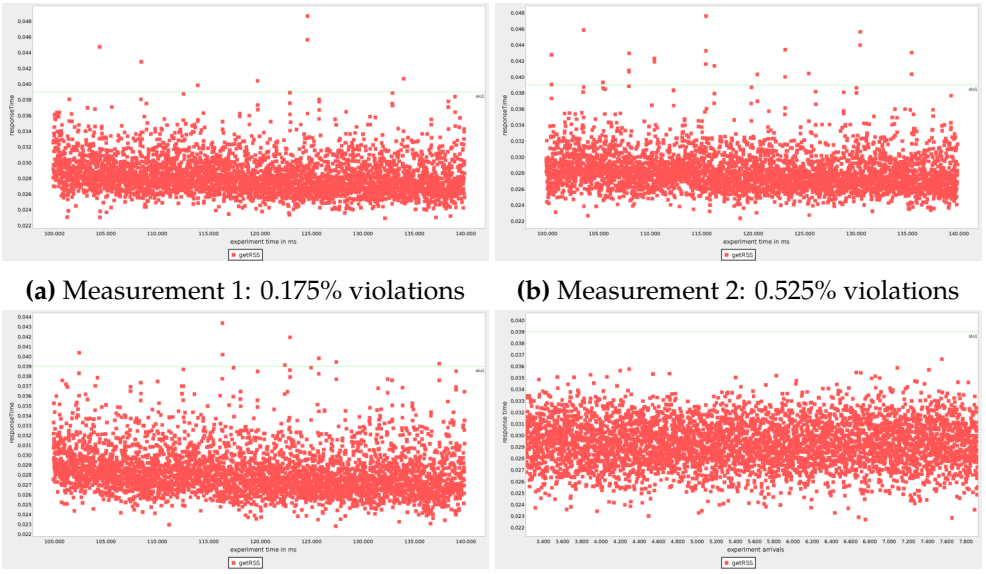
in additional efforts. However, there is no benefit of high accuracy as thresholds at the end slightly overrate actual response times.

Load-Level Bursty workloads and higher load levels increase variations in response times which may justify higher offsets (cf. Figure 7.11).

Characteristics of Evaluation Method Compared to measurements, outliers occur less frequent in model-based predictions. Performance models abstract the real system by applying statistical approximations instead of, for example, fine-grained modeling background processes. Therefore, they can predict outliers less accurately especially if they rarely occur in measurements. Therefore, we again conclude that the desired `SATISFACTION-LEVEL` should not be set to 100%, but to a lower value, such as 95%. Then performance models can be reasonably applied within the SLA negotiation process.

Knowing the impacting factors, we provide —based on our studies— concrete recommendations for SLA negotiation based on performance evaluations results. What is an appropriate offset for SLA generation? — we decided on 20%. How parameterize efficiently? – Fast analyses are sufficient as we perform rounding. Model-based predictions can be applied when providing sufficient accuracy. An offset of twenty percent followed by rounding up by two decimal places depicts a feasible strategy to parameterize SLAs within our studies. Rounding to more decimal numbers would impede a stable and reproducible SLA parametrization. Adding more decimal numbers would introduce an illusion of precision which is neither provided by measurements nor model-based predictions while providing a feasible time-to-result. While this does not provide customer value, also a significantly higher performance evaluation period would be required. An overestimation of twenty percent is not always required but can be assumed justifiable within the scope SLA parametrization.

We demonstrated the essential SLA life-cycle features, which can be applied versatilely using degrees of freedom. The SLA evaluation concern depicted in Figure 7.7 evaluates the Petclinic system for two scenarios. Scenarios one has an arrival rate of 790 users per second. The second scenario has 1000 user sessions per second, causing a higher resource contention resulting in increased response times and SLA violations. The concern depicted in Figure 7.8 evaluates the compliance to an SLA for a different number of cores.



(a) Measurement 1: 0.175% violations

(b) Measurement 2: 0.525% violations

(c) Measurement 3: 0.200% violations

(d) Model-based prediction: 0% violations

Figure 7.12: SLA evaluations using generated SLO thresholds (cf. Listing 7.5) for 100 requests per second for the Netflix RSS reader.

Listing 7.7: Concern to evaluate the SLA compliance for the Petclinic system for different arrival rates.

```
EVALUATE
AGREEMENTS
  sla CONTAINS slo1,slo2,slo3,slo4,slo5
GOALS
  slo1:welcome.responseTime<1.3 ms
  PENALTY 1.0 EUR PER 10% VIOLATIONS
  slo2:showVetList.responseTime<1.7 ms
  PENALTY 1.0 EUR PER 10% VIOLATIONS
  slo3:initFindForm.responseTime<1.3 ms
  PENALTY 1.0 EUR PER 10% VIOLATIONS
  slo4:processFindForm.responseTime<4 ms
  PENALTY 1.0 EUR PER 10% VIOLATIONS
  slo5:showOwner.responseTime<2.6 ms
  PENALTY 1.0 EUR PER 10% VIOLATIONS
VARYING 'arrival rate' <790, 1000>
USING dml@'petclinic.properties';
```

Listing 7.8: Concern to evaluate the SLA compliance for the welcome service of Petclinic system equipped with different amount of CPU cores.

```
EVALUATE
AGREEMENTS
  sla CONTAINS slo1
GOALS
  slo1:welcome.responseTime<1.3 ms
VARYING 'processing units cpu' AS cores <2,4,8>
USING dml@'petclinic.properties';
```

Listing 7.9: Concern to determine the minimum number of CPU cores required to ensure given quality of service requirements for Petclinic system.

```
EVALUATE MIN
VARYING 'processing units cpu'
  AS cores <1 .. 40 BY 1>
SATISFYING AGREEMENTS
sla CONTAINS slo1
GOALS
  slo1:welcomeGET.responseTime<0.0013
USING dml@'model.properties';
```

Optimization Concerns

An integral part of performance concerns traces back to optimization problems. These can be constrained or unconstrained. Quality of service guarantees usually constrains optimization of the system configuration or provisioned resources. In contrast, finding upper bounds for usage is not necessarily constrained.

The declarative language can be used to determine minimized resource configurations that still satisfy given quality of service guarantees. It allows for automated parameter tuning, which usually relies on the experience of a performance engineer. We can use our approach to determine reconfiguration rules for a proactive setting using workload forecasting. If the expected number of customers exceeds a threshold, an adaptation mechanism reconfigures to a derived number of cores. Exemplary, the concern in Listing 7.9 varies the number of cores and checks for SLA conformance. The configuration having the lowest number of cores that satisfies the given SLA has seven cores, which has been determined by model-based evaluation and confirmed by measurements.

Our framework also allows to automatically determine the maximum request rate the system can satisfy ensuring a given SLA. The user requirements on formulating optimization concerns can be different. Besides slightly different meanings of the following concerns, this flexibility can help to pinpoint and reduce the number of required evaluations. For example, the concern in Listing 7.10 determines the maximum arrival rate from a user-configured set of alternatives for the maximum resource configuration. The system configuration with 48 cores can cope with up to 3500 users per second. We determined the maximum user count model-based as the JMeter workload generator for the measurements was not able to steadily provide such high loads. The concern in Listing 7.11 requests the Pareto front for resources and maximum arrivals by describing a multi-objective optimization problem. The concern in Listing 7.12 controls the variation at the resource dimension and optimizes for the maximum arrival rate which the Petclinic system can serve for a given SLA. The concern in Listing 7.13 considers different dimensions for optimization and variation.

Listing 7.10: Query to determine the maximum request rate the Petclinic system can satisfy ensuring an SLA from a set of arrival rates.

```
EVALUATE MAX
VARYING 'arrival rate'
  AS rps <700 .. 3000 BY 500>
AGREEMENTS
  sla CONTAINS (slo1)
GOALS
  slo1:welcomeGET.responseTime<0.0013
USING dml@'model.properties';
```

Listing 7.11: Query to determine the maximum request rate the Petclinic system can satisfy ensuring an SLA.

```
OPTIMIZE MAX 'arrival rate'
AGREEMENTS
  sla CONTAINS (slo1)
GOALS
  slo1:welcomeGET.responseTime<0.0013
USING dml@'model.properties';
```

Listing 7.12: Query to determine the maximum request rate the Petclinic system can satisfy ensuring an SLA Find Pareto front.

```
OPTIMIZE MAX 'arrival rate', MIN 'processing units cpu'
AGREEMENTS
  sla CONTAINS (slo1)
GOALS
  slo1:welcomeGET.responseTime<0.0013
USING dml@'model.properties';
```

Listing 7.13: Query to determine the maximum request rate the system can satisfy ensuring an SLA for different number of CPUs.

```
OPTIMIZE MAX 'arrival rate'
VARYING 'processing units cpu'
  AS cores <1 .. 40 BY 5>
AGREEMENTS
  sla CONTAINS (slo1)
GOALS
  slo1:welcomeGET.responseTime<0.0013
USING dml@'model.properties';
```

The analyses presented so far target mainly one system configuration snapshot only varying provisioned resources. However, software system evolution incorporates many more aspects, for example, software refactoring (exchange of database, integration of new libraries) or feature improvements due to requirements changes. Feature improvements of software systems often come at the cost of performance degradation. For example, adding a protection or encryption component to a software system may decrease its performance [Iff+18b]. If a system evolves, contracts and configurations tailored to a previous version of the system need to be updated. Therefore, the presented storyline repeats in every iteration of the software life-cycle.

7.4.2 Optimized Processing and Interpretation of Tradeoffs

The processing of concerns can be optimized according to non-functional goals. For a balanced evaluation, we assume the existence of a Pareto front where the optimization of a single attribute does not affect others. A configuration should always be from or close to the Pareto set. To further optimize, one has to tradeoff optimization criteria. Therefore, the generic DPE language allows specifying tradeoff decisions using analysis constraints. In the following, we focus on optimization of time-to-result. The time-to-result can be optimized by

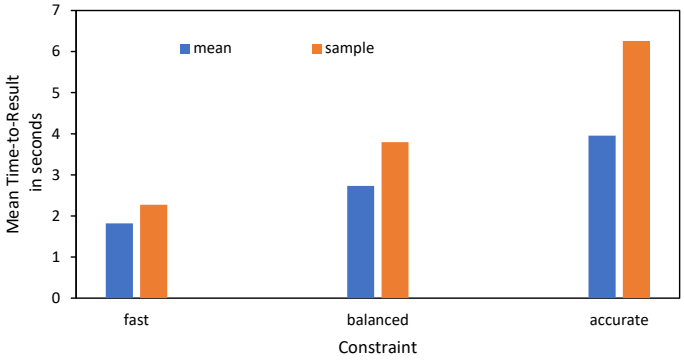


Figure 7.13: Average time-to-result for different processing constraints for an evaluation of a Petclinic scenario with an arrival rate of 170 and 6 CPU cores.

evaluation approach selection, by tailoring the parametrization of performance evaluations, or by avoiding performance evaluation runs.

Tailored Parametrization of Performance Evaluation Approaches In general, the applied evaluation technique has a significant impact on the time-to-result. The measurement experiments took 400 seconds per evaluation run excluding system startup and ramp-up time which accounts about 30 seconds for the initialization of the application for both systems. The time for the model-based evaluation of a single system state is below 10 seconds in our experiments. For constrained analyses this can be significantly faster. Also, the processing of particular performance evaluation techniques can be optimized. The realization is specific to the performance evaluation methodology. Optimizations have to be implemented within the corresponding connector.

Model-based performance evaluation often allows for tuning several analysis parameters according to non-functional goals. To illustrate the tradeoff for accuracy and time-to-result, we constrain the the processing of concerns as as `fast`, `balanced`, and `accurate`. The model-based performance evaluation adapter modifies the simulation configuration according to the constraint. Constrained as fast reduces the processing time, at the cost of assured accuracy. This provides an increased significance, but values remained for the particular experiment equivalent. For the Petclinic system and the Netflix RSS reader application at medium load. We requested response times once as mean and once as sample. Figure 7.13 depicts time-to-result for all constraints for the Petclinic use case. Figure 7.14 depicts the same for an evaluation for the Netflix RSS reader use case.

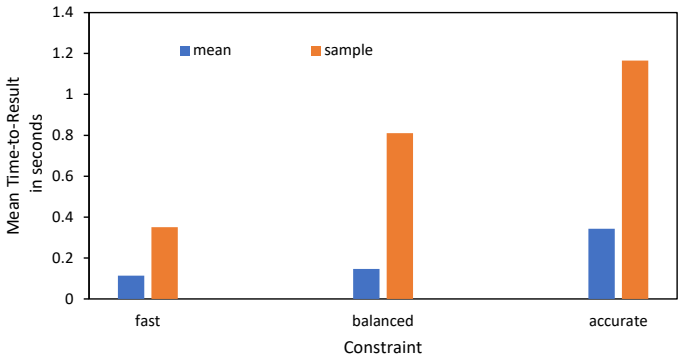


Figure 7.14: Mean time-to-result for different processing constraints for an evaluation of a Netflix RSS reader scenario with 100 arrivals per second and no dropping of requests.

The measurement-based analysis uses tailored measurement configurations as presented in [Blo+16]. Like for model-based analysis, experiment parameters for measurement-based analysis like ramp-up time and run length can be optimized. Moreover, measurement instrumentation tailored to SLA metrics can be applied to reduce system overhead, as we performed in [Blo+16]. Additional constraints may target a reduced measurement overhead, e.g., to apply sampling [Sig+10].

Avoiding Evaluations at Generic Concern Processing At framework level, non-functional constraints can be interpreted to reduce the number of evaluations triggered by the connector. Every non-executed evaluation run saves execution time and resources. Wherever a variation clause demands fast analysis, interpolation libraries can be used. Instead of evaluating all variations, only a subset is evaluated using performance evaluations, and missing values are interpolated. Moreover, all concerns that search a configuration with particular characteristics can benefit from an efficient search. The parametrization of genetic algorithms, which is the number of iterations and population size, can be fit to non-functional optimization goals. The current framework implementation does utilize the full potential of optimization yet. However, we sketched how to implement time-to-result and accuracy tradeoff.

7.4.3 Efforts and Savings

The previously presented case studies and evaluations show the benefits of applying the declarative performance engineering approach. In the following, we describe effort and savings of applying it for different evaluation scenarios.

7.4.3.1 Applying Disparate Performance Evaluation Methods

While the benefits of changing back and forth between measurement-based and model-based evaluation of concerns during the software life-cycle are evident (illustrated by the previous two sections), expenses without our integrated approach are significant. In contrast to using our declarative approach and the presented tooling, one can switch between performance evaluation approaches replacing the `ContextAccess` statement. For example, switching from model-based analysis using DML to measurement-based analysis using Kieker happens by merely replacing `dml@'system.properties'` with `'kieker@system.properties'`. Thereby, the same interface can trigger analyses using distinct kinds of evaluations.

The declarative approach reduces the tool interfaces to be learned to one! The straightforward language interface hides the complexity by not requiring any of the different performance evaluation parameters to provided a balanced evaluation. If non-functional requirements on processing are different, the adapter can derive a tailored parameterization based on a single constraint or optimization goal. Thus, the declarative approach tremendously reduces the efforts of applying the performance evaluation approaches by an automated answering process. The monitoring connector connects and automates multiple steps, such as the generation of a monitoring configuration, and tools, measurement framework and load driver, for performance testing. The declarative approach abstracts from its underlying performance evaluation technologies. Changing to another monitoring framework for which a connector is provided merely requires only to change the connector specifier. The integration of another load driver happens by an adaptation of the connector implementation. The connector for model-based analysis executes five model transformations before simulating a QPN and back feeding the results.

The quality of performance evaluation results is equivalent to non-automated solutions as we trigger existing tools. As the connector implementations store intermediate artifacts, such as monitoring configurations or model transformation results, a performance analyst could still intervene and manually customize processing if required. However, manual customizations were not required in the presented case studies.

7.4.3.2 Answering Multiple Types of Concerns

From the performance analysis tools providing multiple types of analysis, the PCM toolchain is most close regarding comprehensiveness. It provides an evaluation of metrics and SLAs as well as optimization. However, its analysis approaches have different interfaces which makes analysis cumbersome. The majority of tools for performance analysis supports a small set of analyses. Most tools even support only a single kind of analysis. Apparently, an approach integrating multiple kinds of analyses saves efforts. The integrated approach does not force to set up and familiarize with multiple tools and interfaces. As the single interface supports all kinds of performance analyses, users do not have to search for functionality or integrate it.

The effort of answering a concern is applying the respective language features to formulate it. As our editor supports auto-completion, the language can be learned quickly. We propose a Domain Specific Language (DSL), as it is efficient in writing and comfortable to read and maintain, while still being machine-readable. To specify concerns, we chose not to use XML, which is machine-readable but can be difficult for humans [Par01; Kos+08]. We provide a languages editor that supports the formulation of concerns using auto-completion and automatic syntax highlighting.

7.4.4 Savings of Framework Reuse

There are manifold advantages to adopting the generic declarative performance engineering framework. Developers implementing an adapter for their performance evaluation approach benefit from the following aspects:

First, there can be significant reuse of code which deals with generic parts of performance analyses and result presentation. Reusable parts of the framework include result interpretation and visualization, and generic analysis algorithms (cf. Section 4.4). As constituted on 26th June 2018, the declarative performance engineering framework has more than 160.000 lines of Java code. Table 7.8 compares complexity metrics of the framework to its connector interface. The presented numbers indicate significant reuse implementing only a small set of methods from the interface. The framework has been developed over more than five years supported by student assistants. As we had no fulltime developer to conduct implementations, we conservatively estimate 36 person-months fulltime development for the framework. The savings correspond to framework efforts subtracting the time required to connect the performance evaluation approach. As we did not connect the performance evaluation approaches in a controlled experiment, we rely upon an upward estimation of six devel-

	Framework	Interface
Package declarations	177	1
Class declarations	571	3
Method declarations	6635	30

Table 7.8: Complexity metrics for the declarative performance engineering framework and its connector interface.

oper months to implement the three interfaces of the framework. As a result, savings when connecting to the current framework version can be up to 30 person-months which correspond to more than 83% compared to the implementation of superimposing analyses and interpretations from scratch. While the framework already includes manifold features, we still see a vast potential for improvements. Developers may benefit from intended future improvements such as initialization bias detection, interpolation, or root cause analysis. Expansions would increase savings when connecting performance evaluation approaches to the framework. Second, comparability to related performance evaluation approaches comes at negligible costs. This facilitates to evaluate the correctness of the implementation and supports error detection by proving benchmark results. Third, efforts for the adoption of a performance evaluation approach by external users can be assumed low as they reuse the same interfaces. Finally, there is a clear architecture. The presented framework encapsulates generic parts of answering performance concerns. This separation of concerns improves software maintainability.

7.4.5 Discussion

The presented evaluation discusses all research questions concerning automation, applicability, and usability. We evaluate the declarative framework against the requirements for holistic performance management using two case study systems. The presented evaluations show that applying model-based performance prediction can speed up performance analyses significantly. The value of performance evaluation method interchangeability depends on the evaluation accuracy of respective methods. The accuracy of higher-level evaluations themselves depends on the accuracy of the applied performance evaluation technique. To obtain accurate results can be challenging for measurement-based approaches [Myt+09] but even more for model-based performance evaluation. The evaluation shows that model-based and measurement-based performance

evaluation methods can reasonably complement each other to answer multiple kinds of concerns within the presented case studies. However, we can only make limited claims on the accuracy of performance models in general. The accuracy depends on the system under consideration, input data for model extraction, and the extraction mechanism. At design time, there are only estimates of resource demands or service times by human experts. Later, performance models generated from traces usually provide more accurate predictions. The required level of accuracy depends on the application scenario and stage. In early stages, a higher degree of uncertainty still might be acceptable.

The requirement for an integrated approach implicitly assumes savings of effort. The evaluation questions "*What is the effort using different evaluation techniques? What are efforts saved using the declarative approach?*" cannot be quantified straightforward, as their answers depend on several factors including the number and kinds of applied performance evaluation approaches, the kinds of analyses, the experience and preferences of performance engineers, or the complexity of application and application scenario. We argue that even when only applying a single performance evaluation mechanism, the declarative approach still provides savings by automating multiple kinds of analyses.

In addition to the presented evaluation, we could have demonstrated savings and ease of use of the declarative performance engineering framework in a user study. However, we decided against this for several reasons. Experiences from related study objects show essentially no insights other than apparent tendencies we already gained from the case studies. The anticipated results of a user study would only allow for a trend but no absolute quantification of improvements. A user study would compare the time required and the evaluation accuracy when solving performance concerns with and without the declarative approach. To quantify the improvements using the declarative approach, a comparison group would have to solve performance concerns without the declarative approach. A substantial problem preventing a comparative user study is that performance evaluation studies require qualified participants at least for the comparison group. The setup and learning of performance evaluation tools, to which studies would compare, would require a significant amount of time. The challenges to realize performance evaluation using state-of-the-art approaches and tools motivate the declarative performance engineering approach.

Chapter 8

Conclusions

This chapter concludes the thesis. First, we provide a summary of its contributions. Then, we discuss the benefits of the presented work and give an overview of potential future work in the area of Software Performance Engineering (SPE).

8.1 Summary

This thesis pursues the vision of Declarative Performance Engineering (DPE) to support stakeholders in the software life-cycle, such as system developers and administrators, in performance-relevant decision making. It decouples the description of user concerns to be solved (performance questions and goals) from the task of (automatically) selecting and applying a specific solution strategy to answer a given concern. Thereby, it bridges the semantic gap between the level on which performance concerns are formulated and the technical level on which performance evaluations are conducted.

The contributions presented in this thesis span the following areas: *specification of performance concerns*, automated *reasoning* using model-based and measurement-based performance evaluation, automated performance *model extraction*, and automated *decision support* for performance evaluation approaches.

A Framework to Answer Performance Related Concerns

To formulate performance concerns, we provide a Domain Specific Language (DSL) that allows specifying several types of performance analyses independent of a concrete performance evaluation approach. Besides the specification of functional aspects, the presented language allows to customize concerns using processing constraints. The latter impact non-functional processing attributes such as accuracy, time-to-result, or system overhead. We provide a framework architecture and a reference implementation to process performance concerns automatically. The framework contains generic algorithms to solve multiple types of performance concerns coupled with a generic results visualization

component. It allows to plug in arbitrary performance evaluation approaches. We implement exemplary adapters for model-based and measurement-based performance evaluation.

We published the vision of DPE in [Wal+16a]. An initial version of the connector to measurement-based analysis has been published in [Blo+16], the visualization in [Wal+16b], and parts of the SLA evaluation in [WOK17]. The integrated approach has been presented in [Wal+18]. All implementations were realized as extensions of the Descartes Query Language (DQL) framework. The work on DPE was partially supported by the German Research Foundation (DFG) under grant No. KO 3445/15-1.

Automated Extraction of Architectural Performance Models from Execution Traces

We provide the first open source model extraction framework for architectural performance models based on execution traces provided by monitoring tools such as APM tools or profilers. The framework, called Performance Model eXtractor (PMX), realizes our proposed method for model extraction generalizing over the targeted architectural modeling language. Our framework integrates state-of-the-art extraction and estimation techniques with object creation routines that apply to many architectural performance models.

The framework separates model extraction from object creation routines specified in a generic model builder interface. To create model extraction routines for a new performance modeling formalism, developers only need to write object creation routines instead of developing model extraction software from scratch. The reuse of the framework lowers the effort to implement performance model extraction tools by up to 91% through a high level of reuse. Further, our approach enables for complementary use of different analysis toolchains accessible through builder implementations for different architectural performance modeling formalisms.

Alternative methodologies for performance model creation have been described in [Wal+17b]. The PMX tool, as well as the idea of the generic builder interface, have been published in [Wal+17c]. We explained providing model extraction as a web service for two architectural performance modeling formalisms in [Wal+17a]. The tool PMX has been applied in cooperation with Forschungszentrum Informatik (FZI) research transfer institute in [Wal+17c] and also independently by researchers from FZI [Rom17]. The development of PMX was partially supported by the DFG under the grant KO 3445/11-1 and in cooperation with ABB as part of an Academic Research Award ($80,000)

awarded in the area of industrial software systems. The ABB Research Grant Program is a highly competitive international program.

Automated and Extensible Decision Support for Performance Evaluation Approaches

We present a methodology and tooling for the automatic selection of performance evaluation approaches tailored to the user concerns *and* application scenario. We propose to decouple the complexity of selecting a performance evaluation strategy for a given scenario. To this end, we separate the modeling of solution approach capabilities from providing a generic decision engine.

We provide an automated decision framework and corresponding capability meta-model to describe performance evaluation techniques. The framework recommends an evaluation approach from a set of registered performance evaluation techniques tailored to specific evaluation scenario needs. To this end, the framework receives as input a user concern and the characteristics of the system under consideration. The proposed tool capability meta-model allows to describe functional and non-functional capabilities of performance evaluation approaches and tools at different granularities. In contrast to existing tree-based approaches, our approach explicitly supports modifying solution approaches and rating criteria and thus remains at the cutting edge of performance evaluation and system development. Our evaluation demonstrates applicability, reproducibility of tree-based decision support, and the adaptability to the evolution of performance evaluation tooling and comparison attributes.

We propose an approach to predict the time-to-result for model-based analysis based on model characteristics and analysis parametrization. We determine performance-relevant features using statistical tests. We implement the approach and demonstrate its practicability by applying it to analyze a simulation-based multi-step analysis approach for an architectural performance model. The presented methodology allows evaluating the suitability of a model solver to match time-to-result constraints for a given prediction scenario.

The decision framework and the corresponding capability meta-model have been published in [WHK17]. The work on decision support was partially supported by the German Research Foundation (DFG) under grant No. KO 3445/15-1.

8.2 Benefits

The benefits of the presented contributions can be summarized as follows:

- The proposed performance concern language in conjunction with the processing framework significantly reduces the complexity of applying performance analyses for all stakeholders and thereby enables performance awareness throughout the software engineering life-cycle. The proposed performance concern language bridges the semantic gap between the level on which performance concerns are formulated and the technical level on which performance evaluations are conducted. The corresponding declarative performance engineering framework provides a high degree of automation in answering performance concerns using measurement-based and model-based performance evaluation approaches.

- Using the automatic performance model extraction framework, performance analysts do not need to create performance models of applications manually. Moreover, its generalization over modeling formalisms (based on a shared model creation interface) allows for significant savings when reusing the core framework.

- The decision support framework enables an automatic, system-aware, and concern-aware choice of suitable performance evaluation approach ranging from model-based to measurement-based approaches. In contrast to static decision trees, the decoupling into a generic decision engine and pluggable capability models enables adaptation to the evolution of performance evaluation approaches.

- The machine-learning-based time-to-result estimation enables assessing the suitability of model-based performance evaluation for individual analysis settings.

8.3 Future Work

The results of this thesis provide a basis for several topics of future work. In the following, we provide an overview of research topics extending the presented work.

- *Feedback systems*: Currently, we assume that a performance engineer learns the usage of our performance concern language and its innovative features

by examples or reading its language specification. In addition, performance engineers can be supported by interactive and application-oriented hints pointing them to its innovative features. Helpful hints comprise, for example, "Did you try to use processing constraints?" when a response takes too long, or "Did you know you can automatically parameterize SLA thresholds?" when evaluating an SLA.

- *Tailored result visualization*: Our declarative approach allows to generalize performance analysis visualization and reuse it for multiple performance evaluation techniques. In addition to providing metrics as plain data, there can be concern-aware reports and visualizations helping to understand problems when reporting to stakeholders. In this thesis, we introduced an essential visualization component that can be expanded for further visualizations such as Kiviat graphs, physical maps overlaid with performance metrics, or (differential) flame graphs.

- *Benchmarking model creation*: Manifold performance model creation approaches have been proposed. However, there are no commonly accepted benchmarking methods concerning comparison attributes and quality metrics. The accuracy of performance metrics can be investigated for manifold system configurations *and* on different aggregation levels, such as means, distributions, or individual requests. Moreover, the performance engineering community has to agree on common benchmarking scenarios and corresponding training datasets. For benchmark development, the declarative approach presented in this thesis can be applied to abstract from different modeling formalisms and facilitate the comparison of model-based predictions to measurements. To generate training datasets and to improve comparability with the system under consideration, simulators should be expanded to be able to return execution traces like provided by application performance monitoring tools.

- *Incremental model updates*: Instead of repeatedly generating performance models from scratch, models can be incrementally updated. Model updates can be due to different triggers affecting different parts of the model which should be reflected by an agent-based update architecture [SWK16] and adaptive selection of fitting methods [GEK18]. Triggers to update a model can be reconfigurations at system run-time as well as source code changes. Concepts for updates at system run-time were sketched, for example in [SWK16] or [GEK18]. However, there is no methodology for an automated and systematic transfer of source code changes to architectural performance models yet [MK18]. Automating such a transfer requires to

identify commits impacting performance and performing a subsequent root cause analysis. Ideally, only the minimally affected subset of a system has to be tested by tailored performance experiments.

- *Parametric dependencies*: Performance model variables may depend on the value of input parameters. To predict the impact of variations in the input parameters on the application performance, performance models have to capture parametric dependencies explicitly. While [Kro12; Bro14], and [Eis+18] propose abstractions to model these parametric dependencies in architectural performance models, they require a user to specify them manually. Techniques to automatically identify these dependencies based on execution traces could help to further improve the predictive power and accuracy of the extracted performance models.

- *Automated decision support:* The decision framework presented in this thesis enables to specify costs of performance evaluation approaches either statically or context-aware. Future work may implement context-aware cost predictions by applying the presented time-to-result prediction methodology to additional solution techniques. Furthermore, additional cost types could be investigated quantitatively, e.g., system overhead for monitoring. Besides application to architectural performance models, the decision methodology can be applied to evaluation software for stochastic performance models, such as ORIS [Buc+10], which supports the analysis of timed and stochastic Petri nets, or Java Modelling Tools (JMT) [BCS09] supporting Queueing Network (QN) and Queueing Petri Net (QPN) analyses.

Appendix

Listing 8.1: Syntax of the declarative performance engineering language using Extended Bacus Naur Form (EBNF)

```
1  DescartesQL ::=
2     ((( PerformanceMetricsConcern | WhatIfConcern | OptimizationConcern | SLAConcern)
          (EntityReferenceClause)? (ConstraintClause)? ) | StructureConcern)
          ContextAccess ';'
3
4  SLAConcern ::= EvaluationClause Contract
5
6  OptimizationConcern ::= ('OPTIMIZE' (ObjectiveClause (',' ObjectiveClause)*)
       VariationClause? ('SATISFYING' Contract)?)
7
8  ScaleFactor ::= (DOUBLE | INT) '*'
9
10 ObjectiveClause ::= (('MIN' | 'MAX' | ) (MetricReferenceClause | VariationPoint) |
       ScalarizedObjectiveClause)
11
12 ScalarizedObjectiveClause ::= ScaleFactor? ObjectiveClause ('+' ScaleFactor?
       ObjectiveClause)('+' ScaleFactor? ObjectiveClause)*
13
14 MultiObjectiveClauseXXX ::= (ObjectiveClause ',' ObjectiveClause (','
       ObjectiveClause)*)
15
16 EvaluationClause ::= (('EVALUATE' ( |'QUANTITATIVELY'|'QUALITATIVELY')) | ('GENREATE
       THRESHOLDS' OffsetClause (',' OffsetClause)* )) VariationClause?
17
18 Contract ::= 'AGREEMENTS' Sla (',' Sla)* 'GOALS' Slo (',' Slo)*
19
20 PerformanceMetricsConcern ::=
21    'SELECT' (MetricReferenceClause | ('*')) VariationClause?
22
23 WhatIfConcern ::=
24    ('WHAT_IF'|'WHAT_DIFFERS_IF') (VariationReferenceClause| AdaptationClause) (','
          (VariationReferenceClause| AdaptationClause))* (MetricReferenceClause |
          ContextTypeID | MetricID |('*')) FilteringClause?
25
26 FilteringClause ::= (FilteringRule | FilteringStrategy)
27
28 AdaptationClause ::=  ID
```

```
29
30 MetricReferenceClause ::=
31    MetricReference (',' MetricReference)*
32
33 MetricReference ::=
34    (ContextElementIDOrAlias ('.' MetricID ('.' StatTypeID)?)?)
35
36 SlaClause::=
37    'AGREEMENTS' Sla (',' Sla)*
38
39 SloClause::=
40    'GOALS' Slo (',' Slo)*
41
42 Sla::=
43    ID 'CONTAINS' weightedSlo (',' weighteadSlo)*
44
45 Slo::= ID ':' (MetricReference | Slo ('*' Slo)*) Comparator Threshold (Fuzziness)?
       (Penalty)?
46
47 Comparator::=
48    '<' | '<=' | '>' | '>=' | '=='
49
50 AggregateFunction ::=
51    'MIN' | 'MAX' | 'GEOMEAN' | 'N' | 'SUM' | 'SUMOFSQUARES' | 'STDDEV' |
52    'VAR' | 'PERCENTILE' | 'MEAN'
53
54 EntityReferenceClause ::=
55    'FOR' EntityReference (',' EntityReference)*
56
57 EntityReference ::=
58    ('RESOURCE' | 'SERVICE') EntityID (AliasClause)?
59
60 AliasClause ::=
61    'AS' ID
62
63 ContextAccess ::=
64    'USING' Connector '@' Scenario
65
66 Connector ::=
67    ID
68
69 ModelLocation ::=
70    STRING
71
72 ConstraintClause ::=
73       'CONSTRAINED AS' ConstraintID
74
75 ConstraintID ::=
76       STRING
```

```
77
78 StructureConcern ::=
79    'LIST' ('ENTITIES' | 'RESOURCES' | 'SERVICES' | 'METRICS' '(' EntityReferenceClause
          ')' | 'Variation' )
80
81
82 AggregateFunction ::=
83    'MIN' | 'MAX' | 'GEOMEAN' | 'N' | 'SUM' | 'SUMOFSQUARES' | 'STDDEV' |
84    'VAR' | 'PERCENTILE' | 'MEAN'
85
86 EntityReferenceClause ::=
87    EntityReference (',' EntityReference)*
88
89 EntityReference ::=
90    ('RESOURCE' | 'SERVICE') EntityID (AliasClause)?
91
92 AliasClause ::=
93    'AS' ID
94
95 VariationClause ::=
96    ((VariationPoint ValueVariationClause) | AdaptationStrategy) (',' ((VariationPoint
          ValueVariationClause) | AdaptationStrategy))*
97
98 AdaptationStrategy::= (VariationPoint ((String | ContextElement)*))
99
100
101 PerformanceIssueConcern ::=
102    'DETECT' (DetectBottlenecksConcern)
103
104 DetectBottlenecksConcern ::=
105    'BOTTLENECKS' VariationClause
106
107 VariationConfigurationPropertyClause ::=
108    '[' VariationConfigurationProperty (',' VariationConfigurationProperty)* ']'
109
110 ConfigurationProperty ::=
111    ID ('.' ID)* '=' STRING
112
113 VariationConfigurationProperty ::=
114    VariationIDOrAlias '.' PropertyName '=' STRING
115
116 ValueVariationClause ::=
117    '<' (((DOUBLE | INT) (',' (DOUBLE | INT))*) | (INT '..' INT 'BY' INT)) '>'
118
119 IntervalVariationClause ::=
120    INT '..' INT 'BY' INT
121
122 VaryingClause ::=
123    'VARYING' VariationReferenceClause
```

```
124
125 VariationReferenceClause ::=
126     VariationReference (',' VariationReference)*
127
128 VariationReference ::=
129    VariationID (AliasClause)? (ValueVariationClause)?
130
131 ObserveClause ::=
132    'OBSERVE' (ObservationClause) (SampleClause)?
133
134 ObservationClause ::=
135    ConnectorTimeUnitClause | ConnectorInstanceReferenceClause | ObserveRelativeClause |
         ObserveBetweenClause
136
137 SampleClause ::=
138    'SAMPLED BY' RelativeTimeDurationClause+
139
140 ConnectorTimeUnitClause ::=
141    INT ID
142
143 ConnectorInstanceReferenceClause ::=
144    ID STRING
145
146 ObserveRelativeClause ::=
147    ObserveRelativeDirectionType (AbsoluteTimeClause | RelativeTimeClause)
148
149 ObserveBetweenClause ::=
150    'BETWEEN' (AbsoluteTimeClause | RelativeTimeWithSignClause)
151    'AND' (AbsoluteTimeClause | RelativeTimeWithSignClause)
152
153 RelativeTimeWithSignClause ::=
154    RelativeTimeSignType RelativeTimeClause
155
156 AbsoluteTimeClause ::=
157    STRING
158
159 RelativeTimeClause ::=
160    RelativeTimeDurationClause+
161
162 RelativeTimeDurationClause ::=
163    INT TimeModifierType
164
165 ObserveRelativeDirectionType ::=
166    'SINCE' | 'NEXT'
167
168 RelativeTimeSignType ::=
169    '-' | '+'
170
171 TimeModifierType ::=
```

```
172     'y' | 'm' | 'w' | 'd' | 'h' | 'M' | 's' | 'S'
```

List of Figures

List of Algorithms

List of Tables

Listings

Bibliography

[Ahm+16] Tarek M. Ahmed, Cor-Paul Bezemer, Tse-Hsun Chen, Ahmed E. Hassan, and Weiyi Shang. "Studying the Effectiveness of Application Performance Management (APM) Tools for Detecting Performance Regressions for Web Applications: An Experience Report". In: *Proceedings of the 13th International Conference on Mining Software Repositories*. MSR '16. Austin, Texas: ACM, 2016, pp. 1–12 (see page 47).

[And+13] Yuki Ando, Seiya Shibata, Shinya Honda, Hiroyuki Tomiyama, and Hiroaki Takada. "Automated Identification of Performance Bottleneck on Embedded Systems for Design Space Exploration". In: *Embedded Systems: Design, Analysis and Verification*. Ed. by Gunar Schirner, Marcelo Götz, Achim Rettberg, Mauro C. Zanella, and Franz J. Rammig. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 171–180 (see pages 37, 61).

[App10] AppDynamics. *The Real Cost of Application Performance Management (APM) Ownership*. Tech. rep. AppDynamics, 2010 (see page 40).

[Ard+14] Giuliano Ardagna Danilo and Casale, Michele Ciavotta, Juan F. Pérez, and Weikun Wang. "Quality-of-service in cloud computing: modeling techniques and their applications". In: *Journal of Internet Services and Applications* 5.1 (2014) (see page 38).

[Bal+04] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. "Model-Based Performance Prediction in Software Development: A Survey". In: *IEEE Transactions on Software Engineering (TSE)* 30.5 (2004), pp. 295–310 (see pages 20, 39, 40).

[Ban+00] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. 3rd. Prentice Hall, 2000 (see page 39).

[Bar+04] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. "Using Magpie for request extraction and workload modelling". In: *Proceedings of the 6th Symposium on Operating Systems Design & Implementation (OSDI'04)*. San Francisco, CA: USENIX Association, 2004, pp. 18–18 (see pages 18, 44).

[BIN03] Paul Barham, Rebecca Isaacs, and Dushyanth Narayanan. "Magpie: online modelling and performance-aware systems". In: *9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*. Lihue, Hawaii: USENIX, May 2003, pp. 85–90 (see pages 18, 178, 179).

[BWZ15] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015 (see page 58).

[BK02] F. Bause and F. Kritzinger. *Stochastic Petri Nets˜- An Introduction to the Theory*. 2nd. Vieweg Verlag, 2002 (see pages 28, 29).

[Bau93] Falco Bause. "Queueing Petri Nets˜- A formalism for the combined qualitative and quantitative analysis of systems". In: *Proceedings of the 5th International Workshop on Petri Nets and Performance Models, Toulouse, France*. Oct. 1993 (see page 29).

[BBM17] Falko Bause, Peter Buchholz, and Johannes May. "A Tool Supporting the Analytical Evaluation of Service Level Agreements". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ICPE '17. 2017, pp. 233–244 (see page 38).

[BLB13] Matthias Becker, Markus Luckey, and Steffen Becker. "Performance Analysis of Self-adaptive Systems for Requirements Validation at Design-time". In: *Conference on Quality of Software Architectures*. QoSA '13. Vancouver, British Columbia, Canada: ACM, 2013, pp. 43–52 (see pages 33, 164).

[Bec+10] Steffen Becker, Michael Hauck, Mircea Trifu, Klaus Krogmann, and Jan Kofron. "Reverse Engineering Component Models for Quality Predictions". In: *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR '10)*. IEEE, 2010, pp. 199–202 (see page 42).

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "The Palladio Component Model for Model-driven Performance Prediction". In: *Journal of Systems and Software* 82.1 (Jan. 2009), pp. 3–22. ISSN: 0164-1212 (see pages 22, 49, 101, 106, 117, 172).

[Bel79] Bell Laboratories. *Unix Programmer's Manual*. 7th. Vol. 1. Murray Hill, NJ, USA, 1979 (see page 18).

[Ben18] Project Benchflow. *Benchflow Project on GitHub*. Website. Online available at `https://github.com/benchflow/benchflow`. Last accessed: 2018-02-08. 2018 (see page 36).

[BCS09] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. "JMT: performance engineering tools for system modeling". In: *SIGMETRICS Perform. Eval. Rev.* 36.4 (2009), pp. 10–15. ISSN: 0163-5999 (see pages 30, 175, 218).

[BPG15] C. P. Bezemer, J. Pouwelse, and B. Gregg. "Understanding software performance regressions using differential flame graphs". In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Mar. 2015, pp. 535–539 (see page 92).

[Bez+19] Cor-Paul Bezemer et al. "How is Performance Addressed in DevOps?" In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, ICPE 2019, Mumbai, India, April 7-11, 2019*. 2019, pp. 45–50 (see page xiv).

[Blo+16] Matthias Blohm, Maksim Pahlberg, Sebastian Vogel, Jürgen Walter, and Dusan Okanovic. " Kieker4DQL: Declarative Performance Measurement". In: *Proceedings of the 2016 Symposium on Software Performance (SSP 2016)*. Kiel, Germany, Nov. 2016 (see pages 7, 98, 164, 207, 214).

[BR03] Christian Blum and Andrea Roli. "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". In: *ACM Comput. Surv.* 35.3 (Sept. 2003), pp. 268–308. ISSN: 0360-0300 (see pages 86, 87).

[Bol+06] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley and Sons, 2006 (see pages 5, 27, 30, 31, 39, 40, 123–125, 131, 138).

[Bon+05] E. Bondarev, P. de With, M. Chaudron, and J. Muskens. "Modelling of input-parameter dependency for performance predictions of component-based embedded systems". In: *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. Vienna, Austria: EUROMICRO, Aug. 2005, pp. 36–43 (see page 97).

[Bon+04] Egor Bondarev, Johan Muskens, Peter de With, Michel Chaudron, and Johan Lukkien. "Predicting Real-Time Properties of Component Assemblies: A Scenario-Simulation Approach". In: *EUROMICRO*. 2004, pp. 40–47 (see page 22).

[BL14] Luca Bortolussi and Roberta Lanciani. "Stochastic Approximation of Global Reachability Probabilities of Markov Population Models". In: *11th European Workshop on Computer Performance Engineering - EPEW 2014*. Vol. 8721. Springer. Florence, Italy: Springer, 2014, 224–239 (see page 140).

[Bre16] Paul Charles Brebner. "Automatic Performance Modelling from Application Performance Management (APM) Data: An Experience Report". In: *Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering, ICPE 2016, Delft, The Netherlands, March 12-16, 2016*. 2016, pp. 55–61 (see pages 46, 102, 114, 115, 167).

[BLL06] Lionel C. Briand, Yvan Labiche, and Johanne Leduc. "Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software". In: *Transactions on Software Engineering (TSE)* 32.9 (Sept. 2006), pp. 642–663. ISSN: 0098-5589 (see page 44).

[Bro14] Fabian Brosig. "Architecture-Level Software Performance Models for Online Performance Prediction". Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, July 2014 (see pages 20, 21, 28, 96, 218).

[BHK11] Fabian Brosig, Nikolaus Huber, and Samuel Kounev. "Automated Extraction of Architecture-Level Performance Models of Distributed Component-Based Systems". In: *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. Oread, Lawrence, Kansas, Nov. 2011, pp. 183–192 (see pages 19, 110).

[BHK14] Fabian Brosig, Nikolaus Huber, and Samuel Kounev. "Architecture-Level Software Performance Abstractions for Online Performance Prediction". In: *Elsevier Science of Computer Programming Journal (SciCo)* Vol. 90, Part B (2014), pp. 71–92 (see pages 19, 145).

[BKK09] Fabian Brosig, Samuel Kounev, and Klaus Krogmann. "Automated Extraction of Palladio Component Models from Running Enterprise Java Applications". In: *Proceedings of the 1st International Workshop on Run-time mOdels for Self-managing Systems and Applications (ROSSA 2009)*. ACM, Oct. 2009 (see page 19).

[Bro+15] Fabian Brosig, Philipp Meier, Steffen Becker, Anne Koziolek, Heiko Koziolek, and Samuel Kounev. "Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-based

Architectures". In: *Transactions on Software Engineering (TSE)* 41.2 (2015), pp. 157–175 (see pages 5, 34, 39, 40, 123–125, 131, 137, 138, 144, 172, 174).

[BE04] Yuriy Brun and Michael D. Ernst. "Finding Latent Code Errors via Machine Learning over Program Executions". In: *Proceedings of the 26th International Conference on Software Engineering*. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 480–490 (see page 140).

[BDK15] Andreas Brunnert, Alexandru Danciu, and Helmut Krcmar. "Towards a Performance Model Management Repository for Component-based Enterprise Applications". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ICPE '15. Austin, Texas, USA: ACM, 2015, pp. 321–324 (see page 46).

[BK17] Andreas Brunnert and Helmut Krcmar. "Continuous performance evaluation and capacity planning using resource profiles for enterprise applications". In: *Journal of Systems and Software (JSS)* 123 (2017), pp. 239–262 (see pages 19, 46, 102, 114, 115, 120, 123).

[BVK13] Andreas Brunnert, Christian Vögele, and Helmut Krcmar. "Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications". In: *Computer Performance Engineering: 10th European Workshop, EPEW 2013, Venice, Italy, September 16-17, 2013. Proceedings*. Ed. by Maria Simonetta Balsamo, William J. Knottenbelt, and Andrea Marin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 74–88 (see pages 19, 43, 44, 46).

[Bru+15] Andreas Brunnert et al. *Performance-oriented DevOps: A Research Agenda*. Tech. rep. SPEC-RG-2015-01. SPEC Research Group — DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), Aug. 2015 (see pages xvi, 1, 19, 48, 101, 123).

[Buc+10] Giacomo Bucci, Laura Carnevali, Lorenzo Ridi, and Enrico Vicario. "Oris: a tool for modeling, verification and evaluation of real-time systems". In: *International Journal on Software Tools for Technology Transfer* 12.5 (Sept. 2010), pp. 391–403. ISSN: 1433-2787 (see pages 30, 218).

[BV17] Peter Buchholz and Sebastian Vastag. "Toward an analytical method for SLA validation". In: *Software and Systems Modeling (SoSyM)* (June 2017). ISSN: 1619-1374 (see page 37).

[But13] Brandon Butler. *Google, Microsoft play catch up to Amazon, add load balancing, auto-scaling to their clouds*. Accessed: 2017-03-01. 2013 (see page 38).

[16] *CACTOS Toolkit Version 2*. Tech. rep. Last visited on 17.01.2017. CACTOS Consortium, 2016 (see pages 22, 101, 106).

[Cam15] Federico De Silva Cameron Haight Will Cappelli. *Gartner's Magic Quadrant for Application Performance Monitoring*. Dec. 2015 (see pages 6, 18, 40).

[CZS08] Giuliano Casale, Eddy Z. Zhang, and Evgenia Smirni. "KPC-toolbox: Simple yet effective trace fitting using markovian arrival processes". In: *2008 Fifth International Conference on Quantitative Evaluation of Systems (QUEST)*. Sept. 2008, pp. 83–92 (see pages 33, 45).

[Cay13] E. Cayirci. "Modeling and simulation as a cloud service: A survey". In: *2013 Winter Simulations Conference (WSC)*. Dec. 2013, pp. 389–400 (see page 115).

[CMB15] Stephen Chen, James Montgomery, and Antonio Bolufé-Röhler. "Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution". In: *Applied Intelligence* 42.3 (Apr. 2015), pp. 514–526. ISSN: 1573-7497 (see page 88).

[CDJ10] Carlos A. Coello Coello, Clarisse Dhaenens, and Laetitia Jourdan. "Multi-Objective Combinatorial Optimization: Problematic and Context". In: *Advances in Multi-Objective Nature Inspired Computing*. Ed. by Carlos A. Coello Coello, Clarisse Dhaenens, and Laetitia Jourdan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–21 (see page 87).

[Coe02] Carlos A Coello Coello. "Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art". In: *Computer Methods in Applied Mechanics and Engineering* 191.11-12 (2002), pp. 1245–1287 (see page 87).

[CDI11] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-Based Software Performance Analysis*. 1st. Springer Publishing Company, Incorporated, 2011 (see pages 34, 44).

[CM00] Vittorio Cortellessa and Raffaela Mirandola. "Deriving a Queueing Network Based Performance Model from UML Diagrams". In: *Proceedings of the 2Nd International Workshop on Software and Performance*. WOSP 2000. Ottawa, Ontario, Canada: ACM, 2000, pp. 58–70 (see page 19).

[Deb01] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001 (see page 87).

[Det12] Markus von Detten. "Archimetrix: A Tool for Deficiency-Aware Software Architecture Reconstruction". In: *Proceedings of the 2012 19th Working Conference on Reverse Engineering (WCRE '12)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 503–504 (see page 42).

[DM06] Antinisca Di Marco and Raffaela Mirandola. "Model Transformation in Software Performance Engineering". In: *Proceedings of the 2nd International Conference on the Quality of Software Architectures (QoSA '06)*. Ed. by Christine Hofmeister, Ivica Crnkovic, and Ralf Reussner. Vol. 4214. LNCS. Springer, 2006, pp. 95–110 (see page 34).

[Did+15] Diego Didona, Francesco Quaglia, Paolo Romano, and Ennio Torre. "Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ICPE '15. Austin, Texas, USA: ACM, 2015, pp. 145–156 (see pages 21, 140).

[DM01] Lei Ding and Nenad Medvidovic. "Focus: A light-weight, incremental approach to software architecture recovery and evolution". In: *In Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA '01)*. IEEE. 2001, pp. 191–200 (see page 42).

[Dül+17] Thomas F. Düllmann, Robert Heinrich, Andre van Hoorn, Teerat Pitakrat, Jürgen Walter, and Felix Willnecker. "CASPA: A Platform for Comparability of Architecture-based Software Performance Engineering Approaches". (Demo Paper). In: *Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA 2017)*. IEEE, 2017 (see pages xv, 171).

[Eat18] Kit Eaton. *How one second could cost amazon $1.6 billion in sales*. Website. Online available at `https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales`. Last accessed: 2018-01-31. 2018 (see page 50).

[ES14] Holger Eichelberger and Klaus Schmid. "Flexible resource monitoring of Java programs". In: *Journal of Systems and Software (JSS)* 93 (2014), pp. 163–186 (see pages 18, 40, 41, 179).

[Eis+19] Simon Eismann, Johannes Grohmann, Jürgen Walter, Jóakim von Kistowski, and Samuel Kounev. "Integrating Statistical Response Time Models in Architectural Performance Models". In: *2019 IEEE International Conference on Software Architecture (ICSA)*. Acceptance Rate: 21,9% (21/96). Hamburg, Germany, Mar. 2019, pp. 71–80 (see page xiii).

[Eis+18] Simon Eismann, Jürgen Walter, Jóakim von Kistowski, and Samuel Kounev. "Modeling of Parametric Dependencies for Performance Prediction of Component-based Software Systems at Run-time". In: *2018 IEEE International Conference on Software Architecture (ICSA 2018)*. Acceptance Rate: 25,6%. May 2018 (see pages xiii, 54, 97, 218).

[FP17] Vincenzo Ferme and Cesare Pautasso. "Towards Holistic Continuous Software Performance Assessment". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ICPE '17 Companion. L'Aquila, Italy: ACM, 2017, pp. 159–164 (see page 47).

[FRH15] Florian Fittkau, Sascha Roth, and Wilhelm Hasselbring. "ExplorViz: Visual Runtime Behavior Analysis of Enterprise Application Landscapes". In: *Proceedings of the 23rd European Conference on Information Systems (ECIS)*. 2015 (see page 92).

[Fon+07] *X-Trace: A Pervasive Network Tracing Framework*. USENIX Association, Apr. 2007, pp. 271–284 (see pages 18, 178).

[Fow+14] J. Joseph Fowler, Thienne Johnson, Paolo Simonetto, Michael Schneider, Carlos Acedo, Stephen Kobourov, and Loukas Lazos. "IMap: Visualizing Network Activity over Internet Maps". In: *Proceedings of the Eleventh Workshop on Visualization for Cyber Security*. VizSec '14. Paris, France: ACM, 2014, pp. 80–87 (see page 92).

[Fra+11] Greg Franks, Peter Maly, Murray Woodside, Dorina C. Petriu, Alex Hubbard, and Martin Mroz. *Layered Queueing Network Solver (LQNS) software package*. `http://www.sce.carleton.ca/rads/lqns/`. Last visit: 2014-03-20. 2011 (see page 30).

[Fra+09] Greg Franks, Tariq Omari, C. Murray Woodside, Olivia Das, and Salem Derisavi. "Enhanced Modeling and Solution of Layered Queueing Networks". In: *Transactions on Software Engineering (TSE)* 35.2 (2009), pp. 148–161 (see page 27).

[Fre+12] Sören Frey, Andre van Hoorn, Reiner Jung, Benjamin Kiel, and Wilhelm Hasselbring. "MAMBA: Model-Based Software Analysis Utilizing OMG's SMM". In: *Proceedings of the 14. Workshop Software-Reengineering (WSR '12)*. Also appeared in Softwaretechnik-Trends 32(2) (May 2012) 49-50. May 2012, pp. 37–38 (see page 37).

[FK98] Svend Frølund and Jari Koistinen. "Quality of Services Specification in Distributed Object Systems Design". In: *Proc. 4th Conference on USENIX Conference on Object-Oriented Technologies and Systems*. COOTS'98. 1998, pp. 1–1 (see pages 36, 37).

[Gam+95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995 (see pages 107, 121).

[GMW00] David Garlan, Robert T. Monroe, and David Wile. "Foundations of Component-based Systems". In: ed. by Gary T. Leavens and Murali Sitaraman. New York, NY, USA: Cambridge University Press, 2000. Chap. Acme: Architectural Description of Component-based Systems, pp. 47–67 (see page 106).

[Gil02] David Gilbert. "The jfreechart class library". In: *Developer Guide. Object Refinery* 7 (2002) (see page 91).

[Goo+12] Thijmen de Gooijer, Anton Jansen, Heiko Koziolek, and Anne Koziolek. "An Industrial Case Study of Performance and Cost Design Space Exploration". In: *Proc. of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE*. Boston, Massachusetts, USA: ACM, 2012, pp. 205–216 (see pages 6, 40, 76, 101, 114).

[GBK14] Fabian Gorsler, Fabian Brosig, and Samuel Kounev. "Performance Queries for Architecture-Level Performance Models". In: *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. Dublin, Ireland: ACM, Mar. 2014 (see pages 34, 37, 71, 77, 94, 145).

[GKM82] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. "Gprof: A Call Graph Execution Profiler". In: *SIGPLAN Not.* 17.6 (June 1982), pp. 120–126. ISSN: 0362-1340 (see pages 18, 43).

[GMS07] Vincenzo Grassi, Raffaela Mirandola, and Antonino Sabetta. "Filling the gap between design and performance reliability models of component-based systems: A model-driven approach". In: *Journal of Systems and Software* 80.4 (2007), pp. 528–558 (see pages 22, 34).

[GEK18] Johannes Grohmann, Simon Eismann, and Samuel Kounev. "The Vision of Self-Aware Performance Models". In: *Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA 2018)*. Seattle, USA, 2018 (see page 217).

[Gro+17] Johannes Grohmann, Simon Eismann, Jürgen Walter, and Samuel Kounev. *Descartes Modeling Language - Quick Start Guide*. University of Würzburg. Am Hubland, Informatikgebäude, 97074 Würzburg, Germany, Apr. 2017 (see page xvi).

[Guo+13] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski. "Variability-aware performance prediction: A statistical learning approach". In: *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Nov. 2013, pp. 301–311 (see pages 22, 140, 141, 167).

[Hal92] Robert J. Hall. "Call Path Profiling". In: *Proceedings of the 14th International Conference on Software Engineering (ICSE '92)*. Melbourne, Australia: ACM, 1992, pp. 296–306 (see page 43).

[Ham09] Dick Hamlet. "Tools and experiments supporting a testing-based theory of component composition". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 18.3 (2009), p. 12 (see page 97).

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009 (see pages 22, 186).

[HHF13] Christoph Heger, Jens Happe, and Roozbeh Farahbod. "Automated Root Cause Isolation of Performance Regressions During Software Development". In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ICPE '13. Prague, Czech Republic: ACM, 2013, pp. 27–38 (see pages 46, 47, 50, 61).

[Heg+17] Christoph Heger, André van Hoorn, Mario Mann, and Dušan Okanović. "Application Performance Management: State of the Art and Challenges for the Future". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017*. ACM, 2017, pp. 429–432 (see pages 5, 49).

[HKR13] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. "Elasticity in Cloud Computing: What it is, and What it is Not". In: *Proc. 10th International Conference on Autonomic Computing (ICAC 2013)*. 2013, pp. 23–27 (see page 38).

[HRD10] K. Hoad, S. Robinson, and R. Davies. "Automating warm-up length estimation". In: *Journal of the Operational Research Society* 61.9 (Sept. 2010), pp. 1389–1403. ISSN: 1476-9360 (see page 33).

[Hoo14] André van Hoorn. *Model-Driven Online Capacity Management for Component-Based Software Systems*. Kiel Computer Science Series 2014/6. Dissertation, Faculty of Engineering, Kiel University. Kiel, Germany: Department of Computer Science, Kiel University, 2014 (see pages 18, 106, 107, 109, 115).

[HWH12] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ICPE '12. Boston, Massachusetts, USA: ACM, 2012, pp. 247–248 (see pages 12, 18, 94, 98, 106–108, 115, 116).

[HJ08] R. Hopkins and K. Jenkins. *Eating the IT Elephant: Moving from Greenfield Development to Brownfield*. IBM Press. Pearson Education, 2008 (see page 12).

[Hri+99] Curtis E. Hrischuk, C. Murray Woodside, Jerome A. Rolia, and Rod Iversen. "Trace-Based Load Characterization for Generating Performance Software Models". In: *Transactions on Software Engineering (TSE)* 25.1 (Jan. 1999), pp. 122–135. ISSN: 0098-5589 (see page 44).

[Hua+14] Peng Huang, Xiao Ma, Dongcai Shen, and Yuanyuan Zhou. "Performance regression testing target prioritization via performance risk analysis". In: *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. 2014, pp. 60–71 (see page 47).

[Hub+17] N. Huber, F. Brosig, S. Spinner, S. Kounev, and M. Bähr. "Model-Based Self-Aware Performance and Resource Management Using the Descartes Modeling Language". In: *Transactions on Software Engineering (TSE)* 43.5 (May 2017), pp. 432–452. ISSN: 0098-5589 (see pages 12, 38, 94, 117, 143, 167).

[Hub14] Nikolaus Huber. "Autonomic Performance-Aware Resource Management in Dynamic IT Service Infrastructures". Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, July 2014 (see page 137).

[Hub+10] Nikolaus Huber, Steffen Becker, Christoph Rathfelder, Jochen Schweflinghaus, and Ralf Reussner. "Performance Modeling in Industry: A Case Study on Storage Virtualization". In: *ACM/IEEE 32nd International Conference on Software Engineering*. Cape Town, South Africa: ACM, May 2010, pp. 1–10 (see pages 46, 101, 114, 118, 163, 167).

[Hub+12] Nikolaus Huber, André van Hoorn, Anne Koziolek, Fabian Brosig, and Samuel Kounev. "S/T/A: Meta-Modeling Run-Time Adaptation in Component-Based System Architectures". In: *Proceedings of the 9th IEEE International Conference on e-Business Engineering (ICEBE 2012)*. Hangzhou, China: IEEE, Sept. 2012, pp. 70–77 (see pages 25, 37).

[Hub+15] Nikolaus Huber, Jürgen Walter, Manuel Bähr, and Samuel Kounev. "Model-based Autonomic and Performance-aware System Adaptation in Heterogeneous Resource Environments: A Case Study". In: *Proceedings of the 2015 IEEE International Conference on Cloud and Autonomic Computing (ICCAC 2015)*. Cambridge, MA, USA: IEEE, Sept. 2015 (see pages xiii, 19, 34, 37, 45, 97, 137, 144).

[HSE10] Marianne Huchard, A. Djamel Seriai, and Alae-Eddine El Hamdouni. "Component-based architecture recovery from object-oriented systems via relational concept analysis". In: *Proceedings of the 7th International Conference on Concept Lattices and Their Applications (CLA 2010)*. Ed. by University of Sevilla. Sevilla, 2010, pp. 259–270 (see page 42).

[HF10] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. 1st. Addison-Wesley Professional, 2010 (see page 47).

[Hyp18] Ltd. Hyperic. Website. Online available at `https://github.com/hyperic/`. Last accessed: 2018-01-25. 2018 (see page 42).

[IBM03] IBM. *Web Service Level Agreement (WSLA) Language Specification, Version 1.0, Revision: wsla-2003/01/28*. `http://www.research.ibm.com/wsla/`. 2003 (see page 36).

[Iff+18a] Lukas Iffländer, Stefan Geißler, Jürgen Walter, Lukas Beierlieb, and Samuel Kounev. "Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking". In: *Proceedings of the 9th Symposium on Software Performance 2018 (SSP'18)*. Hildesheim, Germany, Nov. 2018 (see page xiv).

[Iff+18b] Lukas Iffländer, Jürgen Walter, Simon Eismann, and Samuel Kounev. "The Vision of Self-aware Reordering of Security Network Function Chains". In: *Proceedings of the 9th ACM/SPEC International Conference on Performance Engineering (ICPE 2018)*. Berlin, Germany, Apr. 2018 (see pages xiv, 33, 62, 205).

[ISO11] ISO/IEC/IEEE. "ISO/IEC/IEEE Systems and software engineering – Architecture description". In: *ISO/IEC/IEEE 42010:2011(E)* (Dec. 2011), pp. 1–46 (see page 1).

[IWF07] Tauseef Israr, Murray Woodside, and Greg Franks. "Interaction Tree Algorithms to Extract Effective Architecture and Layered Performance Models from Traces". In: *Journal of Systems and Software* 80.4 (Apr. 2007), pp. 474–492. ISSN: 0164-1212 (see page 44).

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley Professional Computing. Wiley, 1991 (see pages 27, 30–32, 37, 61).

[Jen81] Kurt Jensen. "Coloured petri nets and the invariant-method". In: *Theoretical Computer Science* 14.3 (1981), pp. 317–336. ISSN: 0304-3975 (see page 28).

[Joh98] Mark W. Johnson. "Monitoring and Diagnosing Application Response Time with ARM". In: *Proceedings of the IEEE 3rd International Workshop on Systems Management (SMW '98)*. IEEE, 1998, pp. 4–13 (see page 18).

[KTK10] Keven T. Kearney, Francesco Torelli, and Constantinos Kotsokalis. "SLA$\star$: An abstract syntax for Service Level Agreements". In: *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID '10)*. IEEE, 2010, pp. 217–224 (see pages 36, 38).

[KL03] Alexander Keller and Heiko Ludwig. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". In: *Journal of Network and Systems Management* 11.1 (2003), pp. 57–81 (see pages 36, 38).

[Ken53] David G. Kendall. "Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain". In: *The Annals of Mathematical Statistics* 24.3 (1953), pp. 338–354. ISSN: 00034851 (see page 26).

[Ker01] Lennard Kerber. "Scenario-Based Performance Evaluation of SDL/MSC-Specified Systems". In: *Performance Engineering, State of the Art and Current Trends*. London, UK, UK: Springer-Verlag, 2001, pp. 185–201 (see page 19).

[Kis+15] Jóakim von Kistowski, Nikolas Roman Herbst, Daniel Zöller, Samuel Kounev, and Andreas Hotho. "Modeling and Extracting Load Intensity Profiles". In: *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*. 2015, pp. 109–119 (see pages 33, 45).

[Kla+11] Benjamin Klatt, Franz Brosch, Zoya Durdik, and Christoph Rathfelder. "Quality Prediction in Service Composition Frameworks". In: *5th Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC 2011)*. Paphos, Cyprus, Dec. 2011 (see page 38).

[Kos+08] Tomaž Kosar, Pablo E. Martínez López, Pablo A. Barrientos, and Marjan Mernik. "A Preliminary Study on Various Implementation Approaches of Domain-Specific Language". In: *Information and Software Technology* 50.5 (2008), pp. 390–405 (see pages 55, 209).

[KL12] Yousri Kouki and Thomas Ledoux. "CSLA: A Language for improving Cloud SLA Management". In: *Int. Conf. on Cloud Computing and Services Science, CLOSER 2012*. 2012, pp. 586–591 (see pages 36, 74).

[Kou05] Samuel Kounev. *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction*. Shaker Verlag, Ph.D. Thesis, Technische Universität Darmstadt, Germany, Dec. 2005 (see pages 29, 46).

[KB03] Samuel Kounev and Alejandro Buchmann. "Performance Modeling of Distributed E-Business Applications using Queueing Petri Nets". In: *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2003), Austin, Texas, USA, March 6-8, 2003*. Best-Paper-Award. Washington, DC, USA: IEEE Computer Society, Mar. 2003, pp. 143–155 (see page 27).

[KB06] Samuel Kounev and Alejandro Buchmann. "SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation". In: *Performance Evaluation* 63.4-5 (May 2006), pp. 364–394. ISSN: 0166-5316 (see pages 30, 95).

[Kou+16] Samuel Kounev, Nikolaus Huber, Fabian Brosig, and Xiaoyun Zhu. "A Model-Based Approach to Designing Self-Aware IT Systems and Infrastructures". In: *IEEE Computer* 49.7 (July 2016), pp. 53–61 (see pages 22, 34, 101, 106, 108, 139, 143, 145).

[Kou+17] Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu, eds. *Self-Aware Computing Systems*. Berlin Heidelberg, Germany: Springer Verlag, 2017 (see pages 20, 139).

[KC14] Jonah Kowall and Will Cappelli. *Gartner's Magic Quadrant for Application Performance Monitoring*. 2014 (see pages 6, 18, 40).

[Koz11] Anne Koziolek. "Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes". PhD thesis. Karlsruhe, Germany: Institut für Programmstrukturen und Datenorganisation (IPD), Karlsruher Institut für Technologie, 2011 (see pages 19, 25, 37, 38, 62, 76, 77, 87).

[KKR11] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. "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*. QoSA-ISARCS '11. Boulder, Colorado, USA: ACM, 2011, pp. 33–42 (see page 87).

[KNR11] Anne Koziolek, Qais Noorshams, and Ralf Reussner. "Focussing Multi-Objective Software Architecture Optimization Using Quality of Service Bounds". In: *Models in Software Engineering*. Ed. by Juergen Dingel and Arnor Solberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 384–399 (see pages 37, 38, 87).

[Koz08] Heiko Koziolek. "Parameter dependencies for reusable performance specifications of software components". Universität Oldenburg, 2008 (see page 97).

[Koz10] Heiko Koziolek. "Performance evaluation of component-based software systems: A survey". In: *Elsevier Performance Evaluation* 67.8 (2010), pp. 634–658 (see pages 20, 39, 40, 46, 104).

[KR08] Heiko Koziolek and Ralf Reussner. "A Model Transformation from the Palladio Component Model to Layered Queueing Networks". In: *Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08)*. Ed. by Samuel Kounev, Ian Gorton, and

Kai Sachs. Vol. 5119. LNCS. Heidelberg: Springer, 2008, pp. 58–78 (see pages 34, 40, 172).

[Kri+13] Kyriakos Kritikos et al. "A Survey on Service Quality Description". In: *ACM Comput. Surv.* 46.1 (July 2013), 1:1–1:58. ISSN: 0360-0300 (see pages 38, 48).

[Kro12] K. Krogmann. *Reconstruction of Software Component Architectures and Behaviour Models Using Static and Dynamic Analysis*. The Karlsruhe series on software design and quality. KIT Scientific Publ., 2012 (see pages 45, 218).

[Kro+16] Johannes Kroß, Felix Willnecker, Thomas Zwickl, and Helmut Krcmar. "PET: Continuous Performance Evaluation Tool". In: *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*. QUDOS 2016. Saarbrücken, Germany: ACM, 2016, pp. 42–43 (see pages 47, 88).

[KKR08] Michael Kuperberg, Martin Krogmann, and Ralf Reussner. "ByCounter: Portable Runtime Counting of Bytecode Instructions and Method Invocations". In: *Proceedings of the 3rd International Workshop on Bytecode Semantics, Verification, Analysis and Transformation*. 2008 (see page 43).

[KKR09] Michael Kuperberg, Martin Krogmann, and Ralf Reussner. "TimerMeter: Quantifying Accuracy of Software Times for System Analysis". In: *Proceedings of the 6th International Conference on Quantitative Evaluation of SysTems (QEST) 2009*. Budapest, Hungary, Sept. 2009 (see page 43).

[LaM08] Martin LaMonica. *Amazon Web Services adds 'resiliency' to EC2 compute service*. `https://www.cnet.com/news/amazon-web-services-adds-resiliency-to-ec2-compute-service/`. Accessed: 2017-03-01. 2008 (see page 38).

[Lar01] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001 (see page 60).

[LR01] Meir M. Lehman and Juan F. Ramil. "Rules and Tools for Software Evolution Planning and Management". In: *Ann. Softw. Eng.* 11.1 (Nov. 2001), pp. 15–44. ISSN: 1022-7091 (see page 1).

[LB16] Sebastian Lehrig and Steffen Becker. "Using Performance Models for Planning the Redeployment to Infrastructure-as-a-Service Environments: A Case Study". In: *Conference on Quality of Software Architectures*. 2016, pp. 11–20 (see pages 45, 46, 101, 114, 163, 167).

[LB17] Philipp Leitner and Cor-Paul Bezemer. "An Exploratory Study of the State of Practice of Performance Testing in Java-Based Open Source Projects". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ICPE '17. L'Aquila, Italy: ACM, 2017, pp. 373–384 (see pages 5, 47).

[Lud+15] Heiko Ludwig et al. "rSLA: Monitoring SLAs in Dynamic Service Environments". In: *Proceedings of the 13th International Conference on Service-Oriented Computing (ICSOC)*. Ed. by Alistair Barros, Daniela Grigori, Nanjangud C. Narendra, and Hoa Khanh Dam. 2015, pp. 139–153 (see page 37).

[Luk+05] Chi-Keung Luk et al. "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation". In: *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05)*. Chicago, IL, USA: ACM, 2005, pp. 190–200 (see page 18).

[MAD09] Frederic Mallet, Charles Andre, and Julien DeAntoni. "Executing AADL Models with UML/MARTE". In: *2009 14th IEEE International Conference on Engineering of Complex Computer Systems*. June 2009, pp. 371–376 (see pages 22, 101, 106).

[Mar+10] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. "Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms". In: *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*. WOSP/SIPEW '10. San Jose, California, USA: ACM, 2010, pp. 105–116 (see pages 19, 84).

[MK18] Manar Mazkatli and Anne Koziolek. "Continuous Integration of Performance Model". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ICPE '18 Companion. Berlin, Germany: ACM, 2018 (see page 217).

[MKK11] P. Meier, S. Kounev, and H. Koziolek. "Automated Transformation of Component-Based Software Architecture Models to Queueing Petri Nets". In: *Proceedings of the 19th International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommuni-*

cation Systems. July 2011, pp. 339–348 (see pages 34, 95, 145, 164, 172).

[Men02a] Daniel A. Menascé. "Load Testing, Benchmarking, and Application Performance Management for the Web". In: *Proceedings of the 2002 International Computer Measurement Group (CMG) Conference*. Computer Measurement Group, 2002, pp. 271–282 (see page 18).

[Men02b] Daniel A. Menascé. "Software, Performance, or Engineering?" In: *Proceedings of the 3rd International Workshop on Software and Performance*. WOSP '02. Rome, Italy: ACM, 2002, pp. 239–242 (see page 48).

[MAD94] Daniel A. Menascé, Virgilio A. F. Almeida, and Larry W. Dowdy. *Capacity Planning and Performance Modeling - From Mainframes to Client-Server Systems*. Prentice Hall, Englewood Cliffs, NG, 1994 (see pages 27, 31).

[MAD04] Daniel A. Menascé, Virgilio A. F. Almeida, and Lawrence W. Dowdy. *Performance by Design: Computer Capacity Planning By Example*. Upper Saddle River, New Jersey, USA: Prentice Hall, 2004 (see page 149).

[Men+99] Daniel A. Menascé, Virgilio Almeida, Rodrigo C. Fonseca, and Marco A. Mendes. "A methodology for workload characterization of E-commerce sites". In: *Proceedings of the 1st ACM Conference on Electronic Commerce*. 1999, pp. 119–128 (see page 45).

[MG00] Daniel A. Menascé and Hassan Gomaa. "A Method for Design and Performance Modeling of Client/Server Systems". In: *IEEE Trans. Software Eng.* 26.11 (2000), pp. 1066–1085 (see page 44).

[MV00] Daniel A. Menasce and A. F. Almeida Virgilio. *Scaling for E Business: Technologies, Models, Performance, and Capacity Planning*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000 (see page 164).

[Mil+95] B. P. Miller et al. "The Paradyn parallel performance measurement tool". In: *Computer* 28.11 (Nov. 1995), pp. 37–46. ISSN: 0018-9162 (see page 92).

[MR09] Cyriel Minkenberg and Germán Rodriguez. "Trace-driven Co-simulation of High-performance Computing Systems Using OMNeT++". In: *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*. Simutools '09. Rome, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, 65:1–65:8 (see pages 33, 45).

[Mon08] Douglas C Montgomery. *Design and analysis of experiments*. John Wiley and Sons, 2008 (see pages 142, 143).

[Mor+09] B. Mora, F. Garcia, Francisco Ruiz, and M. Piattini. "Model-Driven Software Measurement Framework: A Case Study". In: *Proceedings of the 9th International Conference on Quality Software (QSIC '09)*. Aug. 2009, pp. 239–248 (see page 36).

[Mot+14] G. Motta, L. You, N. Sfondrini, D. Sacco, and T. Ma. "Service Level Management (SLM) in Cloud Computing - Third Party SLM Framework". In: *2014 IEEE 23rd International WETICE Conference*. 2014, pp. 353–358 (see page 38).

[Mül+16] Christoph Müller, Piotr Rygielski, Simon Spinner, and Samuel Kounev. "Enabling Fluid Analysis for Queueing Petri Nets via Model Transformation". In: *Proc. of International Workshop on Practical Applications of Stochastic Modelling (PASM)*. Münster, Germany: Elsevier, Apr. 2016 (see pages 6, 34, 40, 124, 125, 137, 147).

[Myt+09] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. "Producing Wrong Data Without Doing Anything Obviously Wrong". In: *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XIV. Washington, DC, USA: ACM, 2009, pp. 265–276 (see page 210).

[Ngu+14] Thanh H. D. Nguyen, Meiyappan Nagappan, Ahmed E. Hassan, Mohamed N. Nasser, and Parminder Flora. "An industrial case study of automatically identifying performance regression-causes". In: *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*. 2014, pp. 232–241 (see pages 50, 92).

[Obj08] Object Management Group, Inc. *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Version 1.1*. `http://www.omg.org/spec/QFTP/1.1/`. 2008 (see page 36).

[Obj12] Object Management Group, Inc. *Architecture-Driven Modernization (ADM): Structured Metrics Meta-Model (SMM), Version 1.0*. `http://www.omg.org/spec/SMM/1.0/`. 2012 (see page 36).

[Obj05] Object Management Group (OMG). *UML-SPT: UML Profile for Schedulability, Performance, and Time, v1.1*. Jan. 2005 (see page 22).

[Oka+16] Dušan Okanović, André van Hoorn, Christoph Heger, Alexander Wert, and Stefan Siegl. "Towards Performance Tooling Interoperability: An Open Format for Representing Execution Traces". In: *Computer Performance Engineering: Proceedings of the 13th European Workshop on Performance Engineering (EPEW '16)*. LNCS. Springer, 2016, pp. 94–108 (see pages 40, 120, 123).

[Ope11] Open Grid Forum. *Web Services Agreement Specification (WS-Agreement)*. `http://ogf.org/documents/GFD.192.pdf`. 2011 (see page 36).

[Par01] Terence Parr. *Humans should not have to grok XML*. `http://www.ibm.com/developerworks/library/x-sbxml/x-sbxml-pdf.pdf`. 2001 (see pages 37, 55, 209).

[PJ17] Akash Patel and Brigitte Jaumard. "Design and Implementation of a Smart Quotation System". In: *Advances in Artificial Intelligence: 30th Canadian Conference on Artificial Intelligence, Canadian AI 2017, Edmonton, AB, Canada, May 16-19, 2017, Proceedings*. Ed. by Malek Mouhoub and Philippe Langlais. Cham: Springer International Publishing, 2017, pp. 191–202 (see page 141).

[PC17] J. F. Pérez and G. Casale. "Line: Evaluating Software Applications in Unreliable Environments". In: *IEEE Transactions on Reliability* PP.99 (2017), pp. 1–17. ISSN: 0018-9529 (see pages 30, 40, 124).

[PC13] Juan F. Pérez and Giuliano Casale. "Assessing SLA Compliance from Palladio Component Models". In: *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2013, Timisoara, Romania, September 23-26, 2013*. 2013, pp. 409–416 (see page 30).

[Per18] Ltd. Performance Assurance. Website. Online available at `http://www.performance-assurance.com.au/`. Last accessed: 2018-01-23. 2018 (see page 6).

[Pet62] C.A. Petri. *Kommunikation mit Automaten*. Schriften des Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universität Bonn. Rhein.-Westfäl. Inst. f. Instrumentelle Mathematik an der Univ. Bonn, 1962 (see page 27).

[PW03] D. C. Petriu and C. M. Woodside. "Performance Analysis with UML". In: *UML for Real: Design of Embedded Real-Time Systems*. Ed. by Luciano Lavagno and Bran Martin Grantand Selic. Boston, MA: Springer US, 2003, pp. 221–240 (see page 19).

[PW07] Dorin Bogdan Petriu and C. Murray Woodside. "An intermediate metamodel with scenarios and resources for generating performance models from UML designs". In: *Springer Software and System Modeling (SoSym)* 6.2 (2007), pp. 163–184 (see page 34).

[PW02] Dorin C. Petriu and C. Murray Woodside. "Software Performance Models from System Scenarios in Use Case Maps". In: *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS '02)*. London, UK, UK: Springer, 2002, pp. 141–158 (see pages 19, 34, 44).

[PS02] Dorina C. Petriu and Hui Shen. "Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications". In: *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS '02)*. Vol. 2324. LNCS. Springer, 2002, pp. 159–177 (see pages 19, 34).

[08] *Q-ImPrESS Project Deliverable D2.1 - Service Architecture Meta-Model (SAMM)*. Tech. rep. Project deliverable. Last visited on 17.01.2017. Q-ImPrESS Consortium, 2008 (see pages 22, 101, 106).

[Rat+12] Christoph Rathfelder, Stefan Becker, Klaus Krogmann, and Ralf Reussner. "Workload-aware System Monitoring Using Performance Predictions Applied to a Large-scale E-Mail System". In: *Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture (WICSA) & 6th European Conference on Software Architecture (ECSA)*. Helsinki, Finland: IEEE Computer Society, Aug. 2012, pp. 31–40 (see page 167).

[RET18] GmbH RETIT. Website. Online available at `https://www.retit.de`. Last accessed: 2018-01-23. 2018 (see pages 6, 53).

[Rom17] Matthias Rombach. "Enabling Architectural Performability Analyses for Microservices via Design Pattern Completions". Master Thesis. Department of Informatics Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology, July 2017 (see pages 9, 171, 214).

[RKT15] Piotr Rygielski, Samuel Kounev, and Phuoc Tran-Gia. "Flexible Performance Prediction of Data Center Networks using Automatically Generated Simulation Models". In: *EAI International Conference on Simulation Tools and Techniques (SIMUTools)*. Athens, Greece, Aug. 2015 (see page 40).

[SHD16] K. A. R. Sagbo, Y. P. E. Hounguè, and E. Damiani. "SLA Negotiation and Monitoring from Simulation Data". In: *2016 12th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*. 2016, pp. 766–772 (see page 37).

[SWK17] Klaus Schilling, Jürgen Walter, and Samuel Kounev. "Spacecraft Autonomous Reaction Capabilities, Control Approaches and Self-Aware Computing". In: *Self-Aware Computing Systems*. Ed. by Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu. Berlin Heidelberg, Germany: Springer Verlag, 2017 (see page xvi).

[SWH06a] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. "Open Versus Closed: A Cautionary Tale". In: *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*. NSDI'06. San Jose, CA: USENIX Association, 2006, pp. 18–18 (see page 20).

[SWH06b] Bianca Schroeder, Adam Wierman, and Mor Harchol-balter. "Closed versus open system models and their impact on performance". In: *in Symposium on Networked Systems Design and Implementation*. 2006 (see page 20).

[SSB93] P. J. Schweitzer, G. Serazzi, and M. Broglia. "A survey of bottleneck analysis in closed networks of queues". In: *Performance Evaluation of Computer and Communication Systems: Joint Tutorial Papers of Performance '93 and Sigmetrics '93*. Ed. by Lorenzo Donatiello and Randolph Nelson. Berlin, Heidelberg: Springer, 1993, pp. 491–508 (see pages 31, 123).

[She+16] Shashank Shekhar, Hamzah Abdel-Aziz, Michael Walker, Faruk Caglar, Aniruddha Gokhale, and Xenofon Koutsoukos. "A simulation as a service cloud middleware". In: *Annals of Telecommunications* 71.3 (Apr. 2016), pp. 93–108. ISSN: 1958-9395 (see page 115).

[Sig+10] Benjamin H. Sigelman et al. *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*. Tech. rep. Google, Inc., 2010 (see pages 18, 40, 41, 179, 207).

[Sin+13] Rahul Singh, Prashant Shenoy, Maitreya Natu, Vaishali Sadaphal, and Harrick Vin. "Analytical Modeling for What-if Analysis in Complex Cloud Computing Applications". In: *SIGMETRICS Perform. Eval. Rev.* 40.4 (Apr. 2013), pp. 53–62. ISSN: 0163-5999 (see page 37).

[Ske07] James Skene. "Language support for service-level agreements for application-service provision". PhD thesis. University College London, 2007 (see page 36).

[SRE10] James Skene, Franco Raimondi, and Wolfgang Emmerich. "Service-Level Agreements for Electronic Services". In: *IEEE Transactions on Software Engineering (TSE)* 36.2 (2010), pp. 288–304 (see pages 36, 38).

[Smi81] Connie U. Smith. "Increasing Information Systems Productivity by Software Performance Engineering". In: *Proceedings of the International CMG Conference*. Dec. 1981, pp. 5–24 (see pages 17, 48).

[Smi+05] Connie U. Smith, Catalina M. Lladó, Vittorio Cortellessa, Antinisca Di Marco, and Lloyd G. Williams. "From UML models to software performance results: An SPE process based on XML interchange formats". In: *Proceedings of the 5th International Workshop on Software and Performance (WOSP '05)*. Palma, Illes Balears, Spain: ACM, 2005, pp. 87–98 (see page 34).

[SW01] Connie U. Smith and L.G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley object technology series. Addison-Wesley, 2001 (see pages 44, 46, 101, 118, 163).

[SW97] Connie U. Smith and Lloyd G. Williams. "Performance Engineering Evaluation of Object-Oriented Systems with SPE*ED". In: *Computer Performance Evaluation*. 1997, pp. 135–154 (see page 30).

[Spi17] Simon Spinner. "Self-Aware Resource Management in Virtualized Data Centers". PhD thesis. University of Würzburg, Germany, July 2017 (see pages 23, 46, 86, 103).

[Spi+15] Simon Spinner, Giuliano Casale, Fabian Brosig, and Samuel Kounev. "Evaluating Approaches to Resource Demand Estimation". In: *Performance Evaluation* 92 (Oct. 2015), pp. 51–71. ISSN: 0166-5316 (see pages 44, 106, 107, 110, 115, 117).

[Spi+14] Simon Spinner, Giuliano Casale, Xiaoyun Zhu, and Samuel Kounev. "LibReDE: A Library for Resource Demand Estimation". (Demo Paper). In: *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. Dublin, Ireland: ACM Press, 2014, pp. 227–228 (see page 44).

[Spi+17a] Simon Spinner, Antonio Filieri, Samuel Kounev, Martina Maggio, and Anders Robertsson. "Run-Time Models for Online Performance and Resource Management in Data Centers". In: *Self-Aware Computing Systems*. Springer International Publishing, 2017, pp. 485–505 (see page 49).

[Spi+17b] Simon Spinner, Antonio Filieri, Samuel Kounev, Martina Maggio, and Anders Robertsson. "Run-time Models for Online Performance and Resource Management in Data Centers". In: *Self-Aware Computing Systems*. Springer, 2017 (see page 39).

[SKM12] Simon Spinner, Samuel Kounev, and Philipp Meier. "Stochastic Modeling and Analysis using QPME: Queueing Petri Net Modeling Environment v2.0". In: *Proceedings of the 33rd International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2012)*. Ed. by Serge Haddad and Lucia Pomello. Vol. 7347. Lecture Notes in Computer Science (LNCS). Hamburg, Germany: Springer-Verlag, June 2012, pp. 388–397 (see page 30).

[SWK16] Simon Spinner, Jürgen Walter, and Samuel Kounev. "A Reference Architecture for Online Performance Model Extraction in Virtualized Environments". In: *Proceedings of the 2016 Workshop on Challenges in Performance Methods for Software Development (WOSP-C 2016) co-located with 7th ACM/SPEC International Conference on Performance Engineering (ICPE 2016)*. Delft, the Netherlands, Mar. 2016 (see pages xv, 95, 114, 217).

[SG98] Bridget Spitznagel and David Garlan. "Architecture-Based Performance Analysis". In: *Proceedings of the 1998 Conference on Software Engineering and Knowledge Engineering*. 1998, pp. 146–151 (see pages 22, 101, 106).

[SE94] Amitabh Srivastava and Alan Eustace. "ATOM: A System for Building Customized Program Analysis Tools". In: *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation (PLDI '94)*. Orlando, Florida, USA: ACM, 1994, pp. 196–205 (see page 18).

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, Wien, 1973 (see page 20).

[Ste+17] Petr Stefan, Vojtech Horky, Lubomir Bulej, and Petr Tuma. "Unit Testing Performance in Java Projects: Are We There Yet?" In: *Proceedings of the 8th ACM/SPEC on International Conference on Perfor-*

mance Engineering. ICPE '17. L'Aquila, Italy: ACM, 2017, pp. 401–412 (see page 47).

[Ste+09] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. 2nd. Addison-Wesley Professional, 2009 (see pages 128, 129, 131).

[Sti+15] Christian Stier, Anne Koziolek, Henning Groenda, and Ralf Reussner. "Model-Based Energy Efficiency Analysis of Software Architectures". In: *European Conference on Software Architecture*. Springer International Publishing. 2015, pp. 221–238 (see page 169).

[Sto96] R. Storn. "On the usage of differential evolution for function optimization". In: *Proceedings of North American Fuzzy Information Processing*. June 1996, pp. 519–523 (see page 88).

[SH12] Misha Strittmatter and Lucia Happe. "Compositional performance abstractions of software connectors". In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE)*. Boston, Massachusetts, USA: ACM, 2012, pp. 275–278 (see page 139).

[Str+16] Misha Strittmatter, Georg Hinkel, Michael Langhammer, Reiner Jung, and Robert Heinrich. "Challenges in the Evolution of Metamodels: Smells and Anti-Patterns of a Historically-Grown Metamodel". In: *10th International Workshop on Models and Evolution (ME)*. CEUR Vol-1706, Oct. 2016 (see pages 101, 103).

[Sum18] Ltd. Sumerian Europe. Website. Online available at `http://www.sumerian.com/`. Last accessed: 2018-01-23. 2018 (see page 6).

[Swo16] Ferdinand Swoboda. "Metamodel-Independent Automated Optimisation of Software Architecture Models". Bachelor Thesis. Department of Informatics Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology, Mar. 2016 (see page 87).

[TA06] Badis Tebbani and Issam Aib. "GXLA a Language for the Specification of Service Level Agreements". In: *First International IFIP TC6 Conference on Autonomic Networking*. 2006, pp. 201–214 (see pages 36, 38).

[The+10] Eno Thereska, Bjoern Doebel, Alice X. Zheng, and Peter Nobel. "Practical Performance Models for Complex, Popular Applications". In: *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMET-

RICS '10. New York, New York, USA: ACM, 2010, pp. 1–12 (see page 72).

[TNG06] Eno Thereska, Dushyanth Narayanan, and Gregory R. Ganger. "Towards self-predicting systems: What if you could ask 'what-if'?" In: *The Knowledge Engineering Review* 21.3 (2006), pp. 261–267 (see page 72).

[Thü+14] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. "A Classification and Survey of Analysis Strategies for Software Product Lines". In: *ACM Comput. Surv.* 47.1 (June 2014), 6:1–6:45. ISSN: 0360-0300 (see pages 21, 47).

[Tos04] Vladimir Tosic. "Service offerings for XML web services and their management applications". PhD thesis. Carleton University, Ottawa, Ontario, Canada, 2004 (see page 36).

[TPP02] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. "WSOL—Web Service Offerings Language". In: *Revised Papers from the CAiSE International Workshop on Web Services, E-Business, and the Semantic Web (WES '02)*. Vol. 2512. LNCS. Springer, 2002, pp. 57–67 (see page 36).

[Tri02] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. 2nd. John Wiley & Sons, Inc., 2002 (see pages 27, 31).

[Vad+15] Renata Vaderna, Željko Vuković, Dušan Okanović, and Igor Dejanović. "A Domain-Specific Language for Service Level Agreement Specification". In: *7th Int. Conf. on Inf. Tech.* 2015, pp. 693–697 (see page 37).

[Vis18] Ltd. Visualize IT. Website. Online available at `http://www.visualize-it.com/`. Last accessed: 2018-01-23. 2018 (see page 6).

[Vog+13] Christian Vogel, Heiko Koziolek, Thomas Goldschmidt, and Erik Burger. "Rapid Performance Modeling by Transforming Use Case Maps to Palladio Component Models". In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13)*. Prague, Czech Republic: ACM, 2013, pp. 101–112 (see page 34).

[VHK15] Christian Vögele, André van Hoorn, and Helmut Krcmar. "Automatic Extraction of Session-Based Workload Specifications for Architecture-Level Performance Models". In: *Proceedings of the 4th International Workshop on Large-Scale Testing, LT'15, Austin, TX, USA, February 1, 2015*. 2015, pp. 5–8 (see page 45).

[Vög+18] Christian Vögele, André van Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. "WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems". In: *Software & Systems Modeling* 17.2 (May 2018), pp. 443–477. ISSN: 1619-1374 (see page 45).

[Wal14] Jan Waller. "Performance Benchmarking of Application Monitoring Frameworks". PhD Thesis. Faculty of Engineering, Kiel University, Dec. 2014 (see pages 40, 41).

[Wal+18] Jürgen Walter, Simon Eismann, Johannes Grohmann, Dušan Okanović, and Samuel Kounev. "Tools for Declarative Performance Engineering". In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE 2018. Berlin, Germany: ACM, Apr. 2018, pp. 53–56 (see pages xiv, 24, 94, 214).

[WEH15] Jürgen Walter, Simon Eismann, and Adrian Hildebrandt. "Automated Transformation of Descartes Modeling Language to Palladio Component Model". In: *Proceedings of the 2015 Symposium on Software Performance (SSP 2015)*. Munich, Germany, Nov. 2015 (see page xv).

[Wal+17a] Jürgen Walter, Simon Eismann, Nikolai Reed, and Samuel Kounev. "Providing Model-Extraction-as-a-Service for Architectural Performance Models". In: *Proceedings of the 2017 Symposium on Software Performance (SSP 2017)*. Karlsruhe, Germany, Nov. 2017 (see pages xv, 9, 214).

[WHK17] Jürgen Walter, Andre van Hoorn, and Samuel Kounev. "Automated and Adaptable Decision Support for Software Performance Engineering". In: *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2017)*. Venice, Italy, Dec. 2017 (see pages xiii, 10, 176, 182, 215).

[Wal+16a] Jürgen Walter, Andre van Hoorn, Heiko Koziolek, Dušan Okanović, and Samuel Kounev. "Asking "What?", Automating the "How?": The Vision of Declarative Performance Engineering". In: *Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering (ICPE 2016)*. Delft, the Netherlands, Mar. 2016 (see pages xiv, 7, 54, 57, 91, 123, 144, 214).

[Wal+16b] Jürgen Walter, Maximilian König, Simon Eismann, and Samuel Kounev. "PAVO: A Framework for the Visualization of Performance Analyses Results". In: *Proceedings of the 2016 Symposium on Software Performance (SSP 2016)*. Kiel, Germany, Nov. 2016 (see pages xv, 7, 88, 214).

[Wal+17b] Jürgen Walter, Antinisca Di Marco, Simon Spinner, Paola Inverardi, and Samuel Kounev. "Online Learning of Run-time Models for Performance and Resource Management in Data Centers". In: *Self-Aware Computing Systems*. Ed. by Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu. Berlin Heidelberg, Germany: Springer Verlag, 2017 (see pages xvi, 9, 41, 108, 214).

[WOK17] Jürgen Walter, Dusan Okanovic, and Samuel Kounev. "Mapping of Service Level Objectives to Performance Queries". In: *Proceedings of the 2017 Workshop on Challenges in Performance Methods for Software Development (WOSP-C'17) co-located with 8th ACM/SPEC International Conference on Performance Engineering (ICPE 2017)*. l'Aquila, Italy: ACM, Apr. 2017 (see pages xv, 7, 214).

[WSK15a] Jürgen Walter, Simon Spinner, and Samuel Kounev. "Parallel Simulation of Queueing Petri Nets". In: *Proceedings of the Eighth EAI International Conference on Simulation Tools and Techniques (SIMUTools 2015)*. Athens, Greece, Aug. 2015 (see pages xiv, 32, 124, 129, 176).

[WSK15b] Jürgen Walter, Simon Spinner, and Samuel Kounev. "Parallel Simulation of Queueing Petri Nets". In: *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems* 16.8 (Aug. 2015) (see page xiii).

[Wal+17c] Jürgen Walter, Christian Stier, Heiko Koziolek, and Samuel Kounev. "An Expandable Extraction Framework for Architectural Performance Models". In: *Proceedings of the 3rd International Workshop on Quality-Aware DevOps (QUDOS 2017)*. l'Aquila, Italy: ACM, Apr. 2017 (see pages xv, 9, 19, 95, 114–116, 123, 214).

[Wan+12] Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Deepal Jayasinghe, Motoyuki Kawaba, and Calton Pu. "Response time reliability in cloud environments: an empirical study of n-tier applications at high resource utilization". In: *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*. IEEE. 2012, pp. 378–383 (see page 164).

[Wat17] Matt Watson. *Comparison of 18 Application Performance Management Tools*. Tech. rep. Stackify, July 2017 (see pages 40, 123).

[Wes13] Bob Wescott. *Every Computer Performance Book: How to Avoid and Solve Performance Problems on The Computers You Work With*. 1st. USA: CreateSpace Independent Publishing Platform, 2013 (see page 164).

[Wes+12] Dennis Westermann, Jens Happe, Rouven Krebs, and Roozbeh Farahbod. "Automated Inference of Goal-oriented Performance Prediction Functions". In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ASE 2012. Essen, Germany: ACM, 2012, pp. 190–199 (see page 21).

[WKH11] Dennis Westermann, Rouven Krebs, and Jens Happe. "Efficient Experiment Selection in Automated Software Performance Evaluations". In: *Proceedings of the 8th European Conference on Computer Performance Engineering*. EPEW'11. Borrowdale, UK: Springer-Verlag, 2011, pp. 325–339 (see page 47).

[WK16] F. Willnecker and H. Krcmar. "Optimization of Deployment Topologies for Distributed Enterprise Applications". In: *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*. Apr. 2016, pp. 106–115 (see pages 46, 102, 114, 115).

[Wil+15] Felix Willnecker, Andreas Brunnert, Bernhard Koch-Kemper, and Helmut Krcmar. "Full-Stack Performance Model Evaluation using Probabilistic Garbage Collection Simulation". In: *Proceedings of the 2015 Symposium on Software Performance (SSP 2015)*. 2015 (see page 120).

[WK18] Felix Willnecker and Helmut Krcmar. "Multi-Objective Optimization of Deployment Topologies for Distributed Applications". In: *ACM Trans. Internet Technol.* 18.2 (Jan. 2018), 21:1–21:21. ISSN: 1533-5399 (see page 76).

[WFP07] Murray Woodside, Greg Franks, and Dorina C. Petriu. "The Future of Software Performance Engineering". In: *2007 Future of Software Engineering*. FOSE '07. 2007, pp. 171–187 (see pages 2, 17, 35, 46, 47, 54).

[WPS02] Murray Woodside, Dorin Petriu, and Khalid Siddiqui. "Performance-related completions for software specifications". In: *Proceedings of the 24th International Conference on Software Engineering (ICSE '02)*. Orlando, Florida: ACM, 2002, pp. 22–32 (see page 34).

[WW04] Xiuping Wu and Murray Woodside. "Performance Modeling from Software Components". In: *Proceedings of the 4th International Workshop on Software and Performance (WOSP '04)*. Vol. 29. Redwood Shores, California: ACM, Jan. 2004, pp. 290–301 (see page 42).

[ZAH11] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. "Security Versus Performance Bugs: A Case Study on Firefox". In: *Proceedings of the 8th Working Conference on Mining Software Repositories*. MSR '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 93–102 (see pages 47, 50).

[ZAH12] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. "A Qualitative Study on Performance Bugs". In: *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. MSR '12. Zurich, Switzerland: IEEE Press, 2012, pp. 199–208 (see pages 47, 50).

[Zen18] Inc. Zenoss. Website. Online available at `http://www.zenoss.com`. Last accessed: 2018-01-25. 2018 (see page 42).

[ZT07] Du Zhang and Jeffrey J. P. Tsai. *Advances in Machine Learning Applications in Software Engineering*. Hershey, PA, USA: IGI Global, 2007 (see pages 140, 190).

[Zha+15] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. "Performance Prediction of Configurable Software Systems by Fourier Learning (T)". In: *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ASE '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 365–373 (see pages 22, 140, 141).

[Zhu+05] Ningning Zhu, Jiawu Chen, Tzi-cker Chiueh, and Daniel Ellard. "TBBT: Scalable and Accurate Trace Replay for File Server Evaluation". In: *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '05. Banff, Alberta, Canada: ACM, 2005, pp. 392–393 (see pages 33, 45).

[ZB93] E. Zimran and D. Butchart. "Performance engineering throughout the product life cycle". In: *1993 CompEuro Proceedings Computers in Design, Manufacturing, and Production*. May 1993, pp. 344–349 (see page 48).

[Zsc10] Steffen Zschaler. "Formal specification of non-functional properties of component-based software systems". In: *Software and Systems Modeling (SoSyM)* 9.2 (2010), pp. 161–201 (see page 36).