Asking “What?”, Automating the “How?”: The Vision of Declarative Performance Engineering

Jürgen Walter
University of Würzburg
97074 Würzburg Germany

Andre van Hoorn
University of Stuttgart
70569 Stuttgart Germany

Heiko Koziolek
ABB Corporate Research
68526 Ladenburg Germany

Dušan Okanović
University of Stuttgart
70569 Stuttgart Germany

Samuel Kounev
University of Würzburg
97074 Würzburg Germany

ABSTRACT

Over the past decades, various methods, techniques, and tools for modeling and evaluating performance properties of software systems have been proposed covering the entire software life cycle. However, the application of performance engineering approaches to solve a given user concern is still rather challenging and requires expert knowledge and experience. There are no recipes on how to select, configure, and execute suitable methods, tools, and techniques allowing to address the user concerns. In this paper, we describe our vision of Declarative Performance Engineering (DPE), which aims to decouple the description of the user concerns to be solved (performance questions and goals) from the task of selecting and applying a specific solution approach. 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 goal is to hide complexity from the user by allowing users to express their concerns and goals without requiring any knowledge about performance engineering techniques. Towards realizing the DPE vision, we discuss the different requirements and propose a reference architecture for implementing and integrating respective methods, algorithms, and tooling.

1. INTRODUCTION

During the life cycle of a software system, performance analysts continuously need to provide answers to and act on performance-relevant questions about response times, resource utilization, bottlenecks, trends, anomalies, etc. Their everyday work includes questions such as [7]: *“What performance would a new service or application deployed on the infrastructure exhibit and how much resources should be allocated to it?”, “What would be the performance impact of adding a new component or upgrading an existing component as services and applications evolve?”, “If an application experiences a load spike or a change of its workload profile, how would this affect the system performance?”, “What would be the effect of migrating a service or an application component from one physical server to another?”*. Despite advances in measurement-based and model-based performance engineering (concerning compatibility, reuse, and modeling convenience)[3, 5], it is still challenging to apply such tools in practice without having extensive knowledge and experience in performance engineering. Additionally, the performance of a software system may be evaluated with different techniques over time, each having specific parameters and capabilities. Searching for solutions, performance analysts typically follow the process depicted in Figure 1 to find an answer to a given performance query. Starting with a performance query (specifying a specific question that needs to be answered), the process consists of the choice of an analysis approach, its parametrization, processing, result filtering and interpretation. The application of the process to answer such queries is complex, time-consuming and error-prone—even for performance experts. To illustrate this, we consider a software service evolution example, focusing on the response time of a single service. At system design-time, predicting the response time of the service involves complex decisions such as the selection of a suitable modeling formalism (predictive formalisms, like layered queueing networks or queueing Petri nets, or architecture-level modeling languages, such as UML MARTE, PCM [2], or DML [11], or

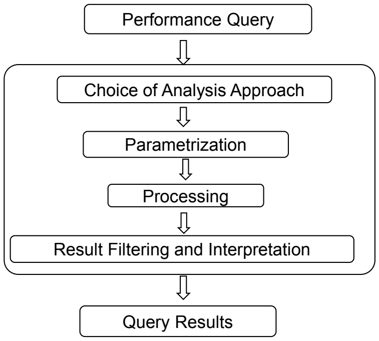
Performance Query
Choice of Analysis Approach
Parametrization
Processing
Result Filtering and Interpretation
Query Results

Figure 1: Performance Query Answering Process

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICPE'16, March 12 - 18, 2016, Delft, Netherlands

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4080-9/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2851553.2858662

even intermediate approaches like CSM [16] or KLAPER [8]), the choice of modeling granularity (e.g., black box, coarse-grained, or fine-grained), solvers and solution technique (e.g., Markovian analytical solvers, product-form solution or simulation-based solvers [4]), and the derivation of model parameters. These decisions affect the modeling accuracy, as well as the speed and overhead of the analysis, and require a lot of expert knowledge. At system testing and deployment time, there is the opportunity to evaluate the service response time by conducting performance measurements on the real system. However, again complex decisions about the measurement configuration have to be made, for example regarding sufficient experiment run length, the configuration of ramp-up time, and the choice of an appropriate instrumentation granularity allowing to obtain the required measurement data. During system operation, it is often also required to predict the effects of possible system reconfigurations or the expected impact of an increased or varied workload mix. This enables proactive resource management but requires modeling techniques that support predicting future system states. The response time query remains the same, however, the analysis approach and parametrization have to be tuned for a fast response. Another source for answering performance queries are historical monitoring logs. The answering process then parameterizes a database request.

In this paper, we describe our vision of Declarative Performance Engineering (DPE) aiming to provide a simplified and unified performance engineering interface. The idea is to use a declarative language allowing to specify performance queries independent of the various approaches that can be applied in the context of the considered system to obtain the required information. The processing of a performance query can be automated and optimized, while hiding complexity from the user. Besides the setup challenges described in the example, the filtering and interpretation of results provided by applying performance engineering techniques can be automated. Further, our approach enables a convergence of model-based and measurement-based analysis as demanded in [15]. Agile software development processes—including the DevOps paradigm—have received increasing attention in the last years [1]. Their implied shorter release cycles raise the need for novel unified interfaces for applying performance engineering techniques that hide complexity from the user and automate the performance analysis process, such that performance engineering techniques can be applied efficiently by developers throughout the software engineering life cycle [5]. Our vision, described in this paper, proposes an approach for addressing this need.

The remainder of this paper is organized as follows: At first, Section 2 formulates the problem statement, which is the basis for our DPE vision described in Section 3. After discussing the vision, we propose a realization approach analyzing the various requirements and proposing a possible architecture for building DPE tools in Section 4. Section 5 reviews our previous work and related work that will serve as foundation of DPE. Finally, Section 6 provides a conclusion and an outlook.

2. PROBLEM STATEMENT

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 there is no generalized layer or a unified interface that provides:

- A high-level language allowing to specify goals and queries independent of the solution approach (e.g., using measurement-based or model-based techniques)
- 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

3. THE VISION OF DECLARATIVE PERFORMANCE ENGINEERING

The high-level objective of Declarative Performance Engineering (DPE) is to support system developers and administrators in performance-relevant decision making. Performance analysts formulate their concerns using a declarative domain-specific language. We aim to reduce the currently huge abstraction gap between the level on which performance-relevant concerns are formulated and the level on which performance engineering techniques are typically applied. 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. The core characteristics of the DPE vision 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 enables the integration of both, 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. For maximum 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 (DevOps) [1].
- Supporting extensibility of the declarative language and the implementation platform and respective tools.

4. DPE APPROACH

The DPE vision requires an architecture for building frameworks and tools that allow to reach the targeted goals. Figure 2 provides an overview of our envisioned DPE archi-

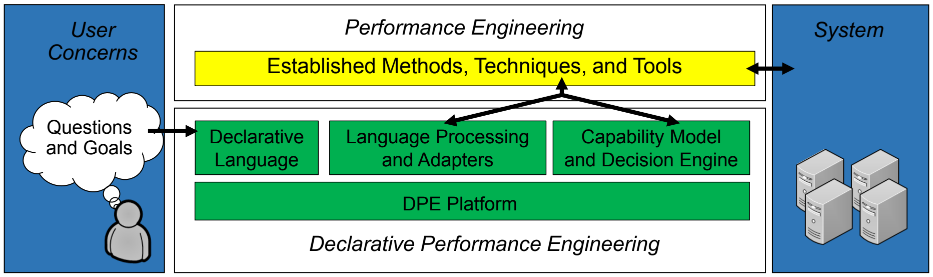
User Concerns
Questions and Goals
Performance Engineering
Established Methods, Techniques, and Tools
Declarative Language
Language Processing and Adapters
Capability Model and Decision Engine
DPE Platform
Declarative Performance Engineering
System

Figure 2: Declarative Performance Engineering Architecture

tecture. It comprises a declarative performance evaluation language, a capability model and a decision engine for solution techniques, as well as language processing algorithms all being part of the DPE platform. In the following, we explain the specific objectives for these components in more detail.

4.1 Declarative Language

The DPE language should be easy to adopt and understandable by software developers and administrators, as well as by non-technicians. We aim to cover a wide range of performance concerns. In particular, we plan to support the following types of language statements:

- **System element queries** enable self-description of the system concerning its elements, e.g., *"What services or resources are provided by the system?"*.
- **User-controlled queries** enable asking questions about the system performance.
  1. *Basic queries* (e.g., *"What is the response time of service x [for workload y] [at time z]?"*)
  2. *Degree-of-freedom queries* or exploration space queries (e.g., *"What is the utilization of server x for [100, 150 or 200] users?"*)
  3. *Exploration space search queries* (e.g., scalability questions about the maximum number of users without SLA violation or bottleneck detection)
- **Temporal queries** enable asking questions concerning observations based on monitoring the system over an extended period of time, e.g., trends, forecasting, and anomaly detection.
- **Sensitivity queries** evaluate the effect/influence of a service or resource, e.g., *"influence of service x on the utilization of resource y"*.
- **Goal definitions** allow to describe user level goals. They provide a basis to derive concrete actions in order to achieve a specified goal. Goals can be in conflict to each other. Hence, the optional specification of priorities or criticality levels should be supported.

For some scenarios, it is not sufficient to solely specify the desired result but also to constrain the way of how results are obtained. This is important given that system perturbation, solution time, and accuracy may differ significantly depending on the applied performance engineering technique. Measuring the performance of a system may have side effects on the system operation. While for some scenarios, it may be acceptable to generate additional load on the system for better measurement precision, in others it may be required to minimize overheads. Hence, the language should support expressing such *tradeoffs*.

4.2 Language Processing and Adapters

To process performance queries expressed using the declarative language, the DPE platform should support plugging in adapters for different performance evaluation approaches. The adapter interface shall be generic and centered around the needs of the language. The adapters control any monitoring or processing required by the respective performance evaluation approach. The requirements on the adapter design are:

- **Automated processing**: Experiment design, load script generation, and experiment execution are carried out automatically.
- **Optimized processing**: The performance query processing shall be efficiently parameterized for the specific scenario. It should take into account user concerns about accuracy, time-to-result, and system perturbation.
- **Light-weight design**: The complexity of tool adapters shall be minimized. Therefore, the generic composition of complex results out of sub-results should be integrated into the language processing.

4.3 Capability Model and Decision Engine

The idea of DPE is to integrate multiple solution adapters. At a first step, it has to be evaluated if an adapter is able to deduce the requested metrics. In case multiple adapters are capable to solve a given query, the choice should be based on a matching of tool and query capabilities. For model-based analysis, there exist many analysis approaches optimized for certain model capabilities (e.g., independence of model subparts, open or closed workload, applied queueing strategies, etc.). Furthermore, the best solution technique also depends on the requested metric. For example, if the performance analyst is solely interested in aggregated values, faster approaches, like product-form solutions, can be applied. Besides improving efficiency under the constraint of preserving accuracy, there are approximation procedures, like fluid analysis, which trade off analysis speed versus the cost of accuracy.

We propose to introduce a *decision engine* that, for a given language statement, automatically selects from the set of plugged in tool adapters, an appropriate one to solve the respective query. The aim of the decision engine is to decide based on matching statement requirements against the tool capabilities. This requires a *capability model* of each tool, capturing its functional and non-functional properties. In particular, capturing information relevant for the aforementioned tradeoffs between accuracy, time-to-result, and system perturbation shall be supported. The capability model will be designed using a hierarchical structure. We propose the use of at least one abstract capability layer where a general model, e.g., for monitoring tools, can be specified. Then, a capability model for a concrete monitoring tool instance can inherit from the abstract monitoring capability model. This enables the reuse of capability specifications for other monitoring tools. The abstract capability models can be overwritten by a concrete tool capability model in case an adapter is not yet implemented.

5. RELATED WORK

In our previous work, we proposed initial query languages and frameworks to automate the application of performance analysis techniques and to parameterize, execute, and filter results accordingly. These are MAMBA [6] for measurement-based analysis and DQL [7] for model-based analysis. However, they consider measurement and model-based analysis separately, instead of, for example, using measurements where available, while resorting to model predictions where measurements are not feasible. Further, these languages currently only support basic queries, no complex or composite queries, e.g., for bottleneck analysis. Besides expressing performance queries, system goal specifications can be declaratively described in the form of service level agreements (SLAs), e.g., based on standards such as WSLA, WSOL, WS-Agreement, SLA⋆, and SLAng [13]. Some of the named approaches assess the SLA conformity of runtime performance properties based on reactive approaches. However, these SLA languages are not yet connected to proactive reconfiguration mechanisms that prevent SLA violations, as done in our preliminary work in [10]. Further, we intend to include several related approaches for implementing our vision, from which we want to name a few. To reduce the overhead of creating model transformations in performance engineering, intermediate models like CSM [16] and KLAPER [8] address the N-to-M problem. Obviously, experiment automation like initialization bias detection [9] should be included. Moreover, degrees-of-freedom exploration can be optimized. Efficient experiment selection can reduce the total number of experimental setups to analyze as performed in [14].

6. CONCLUSION AND NEXT STEPS

Throughout the past decades, various established methods, techniques, and tools for modeling and evaluating performance properties have been proposed. However, the application of performance engineering techniques to address a given user concern is challenging and requires expert knowledge. In this vision paper, we described the Declarative Performance Engineering (DPE) paradigm aiming to decouple the description of user concerns to be solved (performance questions and goals) from the various possible techniques and solution approaches. We envision an automated selection and execution of a performance engineering approach tailored to address a given user concern. Towards accomplishing the DPE vision, we discussed the major requirements and proposed a possible architecture for building respective analysis tools and frameworks. We expect DPE to impact all fields in computer science where performance plays an important role, especially for real-time systems and cloud computing. Our architecture enables to already use sub parts, e.g., the decision support, the automated parametrization, or the result filtering. As next steps, we will continue our joint work on DPE based on a project funded by the German Research Foundation (DFG). Further, we will continue pushing the DPE vision within the SPEC DevOps community. In a long term perspective, the tools to answer performance questions and to process concerns can be used as a basis for self-aware computing systems [12].

Acknowledgments: This work is supported by the German Research Foundation (DFG) in the Priority Programme “DFG-SPP 1593: Design For Future—Managed Software Evolution” (HO 5721/1-1 and KO 3445/15-1) and by the Research Group of the Standard Performance Evaluation Corporation (SPEC).

7. REFERENCES

[1] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect’s Perspective*. Addison-Wesley Professional, 2015.

[2] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *Elsevier Journal of Systems and Software (JSS)*, 2009.

[3] A. B. Bondi. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice*. Addison-Wesley Professional, 2014.

[4] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev. Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. *IEEE Transactions on Software Engineering (TSE)*, 41(2):157–175, 2015.

[5] A. et al. Performance-oriented DevOps: A research agenda. Technical Report SPEC-RG-2015-01, SPEC Research Group — DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), 2015.

[6] S. Frey, A. van Hoorn, R. Jung, W. Hasselbring, and B. Kiel. MAMBA: A measurement architecture for model-based analysis. Technical Report TR-1112, Department of Computer Science, University of Kiel, Germany, 2011.

[7] F. Gorsler, F. Brosig, and S. Kounev. Performance queries for architecture-level performance models. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*, pages 99–110. ACM, 2014.

[8] V. Grassi, R. Mirandola, and A. Sabetta. Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software*, 80(4):528–558, 2007.

[9] K. Hoad, S. Robinson, and R. Davies. Automating warm-up length estimation. *Journal of the Operational Research Society*, 61(9):1389–1403, 2010.

[10] N. Huber, A. van Hoorn, A. Koziolek, F. Brosig, and S. Kounev. Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. *Service Oriented Computing and Applications Journal (SOCA)*, 8(1):73–89, 2014.

[11] S. Kounev, F. Brosig, and N. Huber. The Descartes Modeling Language. Technical report, Department of Computer Science, University of Wuerzburg, 2014.

[12] S. Kounev, X. Zhu, J. O. Kephart, and M. Kwiatkowska. Model-driven Algorithms and Architectures for Self-Aware Computing Systems (Dagstuhl Seminar 15041). *Dagstuhl Reports*, 5(1):164–196, 2015.

[13] K. Kritikos, B. Pernici, P. Plebani, C. Cappiello, M. Comuzzi, S. Benrernou, I. Brandic, A. Kertész, M. Parkin, and M. Carro. A survey on service quality description. *ACM Comput. Surv.*, 46(1):1:1–1:58, July 2013.

[14] D. Westermann, R. Krebs, and J. Happe. Efficient experiment selection in automated software performance evaluations. In *Proceedings of the 8th European Conference on Computer Performance Engineering*, EPEW’11, pages 325–339, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] M. Woodside, G. Franks, and D. C. Petriu. The future of software performance engineering. In *2007 Future of Software Engineering (FOSE ’07)*, pages 171–187. IEEE, 2007.

[16] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by unified model analysis (puma). In *Proceedings of the 5th International Workshop on Software and Performance (WOSP ’05)*, pages 1–12. ACM, 2005.