Tools for Declarative Performance Engineering

Tutorial Paper

Jürgen Walter
University of Würzburg
Germany

Simon Eismann
University of Würzburg
Germany

Johannes Grohmann
University of Würzburg
Germany

Dušan Okanović
University of Stuttgart
Germany

Samuel Kounev
University of Würzburg
Germany

ABSTRACT

Performance is of particular relevance to software system design, operation, and evolution. However, the application of performance engineering approaches to solve a given user concern is challenging and requires expert knowledge. In this tutorial paper, we guide the reader step-by-step through the answering of performance concerns following the idea of declarative performance engineering. We explain tools available online, which can be used for automating huge parts of the software performance engineering process. In particular, we present a performance concern language, for which we provide automated answering and visualization referring to measurement-based and model-based analysis. We also detail how to derive performance models using automated extraction of architectural performance models and modeling of parametric dependencies.

CCS CONCEPTS

• **General and reference → Evaluation**; **Experimentation**; **Performance**; • **Software and its engineering → Model-driven software engineering**; **Abstraction, modeling and modularity**; **Software performance**;

KEYWORDS

Declarative performance engineering, Model-based performance analysis, Measurement-based performance analysis, Software performance engineering

ACM Reference Format:
Jürgen Walter, Simon Eismann, Johannes Grohmann, Dušan Okanović, and Samuel Kounev. 2018. Tools for Declarative Performance Engineering: Tutorial Paper. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3185768.3185777

1 INTRODUCTION

Currently, there is a huge abstraction gap between the level on which performance concerns are formulated and the level on which performance evaluations are actually executed. Declarative Performance Engineering (DPE) [14] decouples the description of performance concerns to be solved (performance questions and goals) from the task of (automatically) selecting and applying specific solution strategies to answer these concerns. In this tutorial paper, we present how to apply open source tools available online to implement DPE.[1] Figure 1 provides a high-level overview of how concerns can be answered using measurement- and model-based analysis. In particular, we present the following building blocks:

Specifying performance concerns DPE requires a language to formulate concerns. We present how to query performance indicators, as well as the latest SLA language features, which we integrated into the DQL (Descartes Query Language) [2, 11] (Section 2).

Answering using measurements One way to answer performance concerns is interpreting measurements derived from application performance measurement (APM) tools. We present how to answer concerns based on the measurements obtained using the Kieker monitoring framework [8] and the adapter to DQL [1] (Section 3).

Answering using performance models Performance concerns can also be answered based on performance models. We present how to answer performance concerns using Descartes Modeling Language (DML) models [3] and the adapter to DQL (Section 4).

Result visualization In addition to providing performance metrics as plain data, there are visualizations which can help stakeholders to better understand problems at hand. We present visualizations based on the PAVO (Performance Analysis Visualization) framework [10] (Section 5).

Performance model extraction Performance models may be created in an editor or derived form monitoring data. We present PMX (Performance Model Extractor) [12] to derive performance models from APM data (Section 6).

Parametric dependencies We present novel modeling features to accurately and efficiently depict parametric dependencies (Section 7).

[1]Descartes Tools: http://descartes.tools/

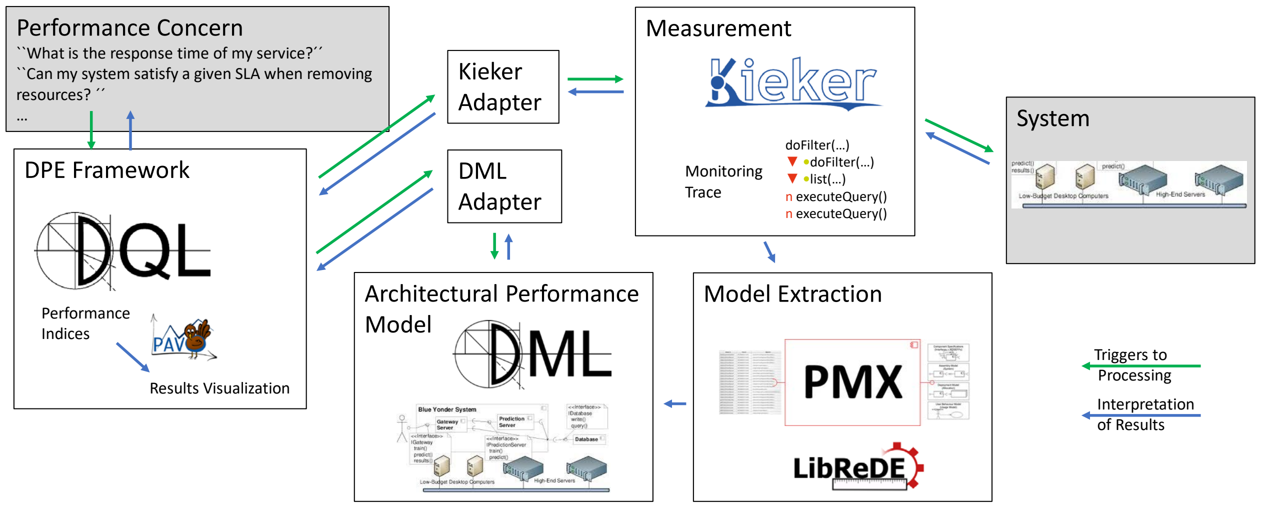


Figure 1: Overview of Tools for Declarative Performance Engineering

2 FORMULATION OF PERFORMANCE RELATED CONCERNS USING DQL

Descartes Query Language (DQL) [2] enables to query performance of a system using adapters to various solution approaches. Thereby, DQL is a realization of the vision of Declarative Performance Engineering (DPE), which decouples the description of user concerns (performance questions and goals) from the task of selecting and applying a specific solution approach.

DQL allows for specification of the following types of concerns: (i) performance indicators [2], (ii) service level agreements (SLAs) [11], and (iii) variation and *what if* analysis. Examples for these types of concerns can be seen in the following listings. Listings show respectively querying of a service response time, SLA evaluation, and analysis of a service response time when the number of CPUs is varied.

```
SELECT service.responseTime
FOR SERVICE 'CatalogActionBean.getItem()' AS
    service
USING kieker@'dql.properties';
```

```
EVALUATE
AGREEMENTS sla1 CONTAINS slo1
GOALS
  slo1: welcome.responseTime < 0.2ms
FOR SERVICE "welcomeGET" AS welcome
CONSTRAINED AS fast
USING connector@'domain_access';
```

```
SELECT session.responseTime
FOR SERVICE 'Session' AS session
VARYING 'processing units cpu'
  AS cores <1 .. 40 BY 1>
USING dml@'model.properties';
```

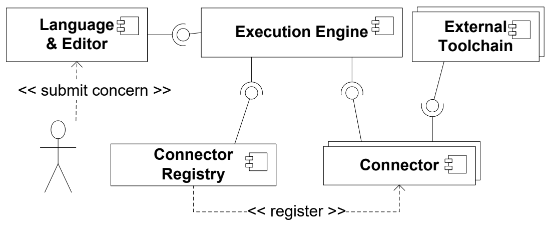


Figure 2: DPE Framework Architecture

The corresponding processing framework, depicted in Figure 2, enables to integrate performance evaluation techniques that may benefit from reusing a set of generic processing algorithms. More details on DQL and query processing can be found in [2, 11].

3 MEASUREMENTS-BASED ANSWERING OF CONCERNS USING KIEKER

In order to showcase answering performance concerns based on the data available in running systems, we developed a DQL adapter [1] for the Kieker monitoring framework [8].

The adapter consists of two modules. A stated performance concern is first analyzed by a *configuration generation module* to create a tailored monitoring configuration for that particular concern. Based on the query, the module identifies points of interest in monitored software. Only required points will be instrumented. This generated configuration is stored in a Kieker configuration file. Obtained data is used by the *filtering module*. This module filters and interprets collected execution traces according to concerns. The implementation of the module uses the pipe-and-filter architecture of the Kieker analysis framework: the chain of filters take out the data required for answering of the stated concern, and convert it into DQL data structures. The *filtering module* can work with monitoring data that is collected using both tailored and default

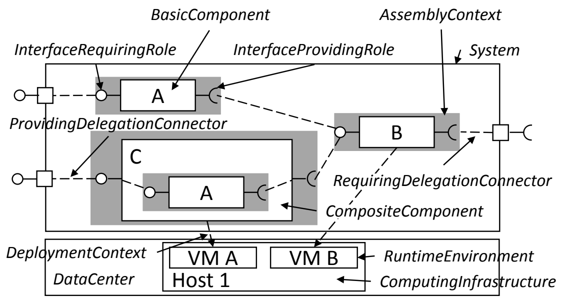


Figure 3: DML notation

monitoring configurations, with equal results in both cases. However, when using tailored configuration, performance overhead is significantly lower.

4 MODEL-BASED ANSWERING OF CONCERNS USING DML

Another way to answer performance related concerns is using performance models. We propose to apply architectural performance models, as they preserve the semantics of the system under test. As a prototype implementation we apply Descartes Modeling Language (DML) [3, 5], due to efficient existing model-solvers.

4.1 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.

A DML instance contains a *repository* of *basic* and *composite components*. Each component has *interface providing* 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 each interface providing role and operation). The service behavior specifies the performance relevant control flow of the component (i.e., resources accesses, external calls to other services, loops, forks, etc.). Composite components bundle a set of components which are 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 3). 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 main entities are *containers* which can be a *computing infrastructure* (i.e., physical server) or a *runtime environment* (e.g. a VM or a middleware service). Each container contains a description of its resources (CPU, hard disks, network links, etc.). *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.

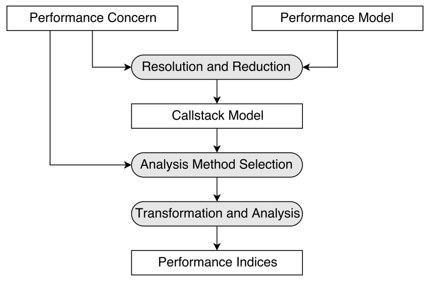


Figure 4: Tailored prediction process

4.2 Tailored Model Solution

Performance concern processing can be based on different requirements in terms of requested metrics, required accuracy, and time-to-result. Therefore, the performance prediction process should allow to flexibly take such requirements into account. DML offers flexibility in solution techniques based on model-to-model transformations (e.g., to Queueing Networks (QNs) or Queueing Petri Nets (QPNs)). Figure 4 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.

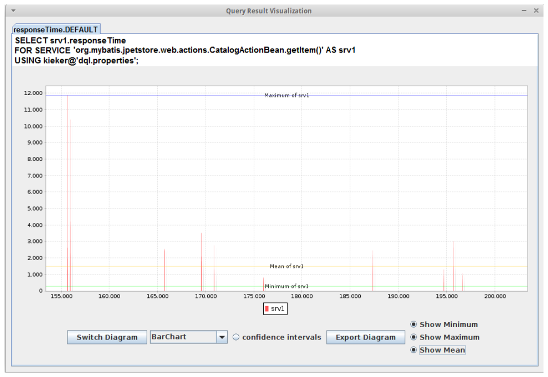
The first analysis method transforms the performance model into a QPN, utilizing the mappings proposed in [6]. The resulting QPN is solved using the SimQPN simulation engine [4]. SimQPN utilizes discrete event simulation to predict performance indices such as the utilization, response time, response 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. Since no simulation is required for the bounds analysis, this 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.

In the future, DML is planned to be expanded by additional analysis methods. The current selection of analysis methods is based on a hard coded decision tree which is unsuitable to maintain. To overcome this weakness, we developed a framework for automated decision support based on capability models [13].

5 RESULT VISUALIZATION

A suitable visualization allows for better understanding of application performance. Usually, analysis tools include some tool specific



```
SELECT srv1.responseTime
FOR SERVICE 'org.mybatis.jpetstore.web.actions.CatalogActionBean.getItem()' AS srv1
USING kieker@'dql.properties';
```

Figure 5: Visualization of performance analysis results

visualization, or no visualization at all. Our declarative approach allows for generalization of performance analysis visualizations, and reuse of these visualizations for multiple performance evaluation techniques.

PAVO framework [10] provides result visualization for a given performance analysis result. It includes several features such as a decomposition into multiple diagrams, flexibility in changing diagram types, and enrichment using aggregated metrics. To illustrate, Figure 5 shows a visualization for measurements triggered by DQL.

6 AUTOMATED PERFORMANCE MODEL EXTRACTION

We see a reluctance from industry to adopt model-based analysis approaches due to the required expertise and modeling effort. Building models from scratch in an editor does not scale for medium and large-scale systems available in industry. In order to automatically derive performance models we propose the use of Performance Model Extractor (PMX) [12], for which we also provide a web service [9]. PMX extracts architectural performance models from application performance management (APM) data.

The parametrization of the required resource demands in order to complete the DML model can be done via Library for Resource Demand Estimation (LibReDE). LibReDE is a library of ready-to-use implementations of state-of-the-art approaches for resource demand estimation, which can be used for online and offline analysis [7]. It provides eight statistical estimation approaches to derive the resource demands based on generic system- and application-level measurements.

7 MODELING OF PARAMETRIC DEPENDENCIES

Currently, automated model extraction cannot recognize when model parameters depend on a range of input characteristics or various system parametrizations. In order to reflect parametric dependencies within a performance model, explicit modeling of input parameters and description of resource demands as a function of input parameters has been shown to be effective. DML provides so-called relationships to model dependencies between various parameters and therefore predict the impact of workload changes on model parameters. Additionally, correlation relationships allow to estimate parameters that cannot be monitored, by defining correlations to measurable variables.

8 CONCLUSION

This tutorial paper addresses tools for automation of software performance engineering approaches. We present a declarative language to specify performance concerns, as well as demonstrate tools which can be used to automatically answer them based on measurements and software performance models, and a framework to provide decision support. To enable model-based analysis, we discuss tools for the efficient creation of performance models.

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).

REFERENCES

[1] Matthias Blohm, Maksim Pahlberg, Sebastian Vogel, Jürgen Walter, and Dusan Okanovic. 2016. Kieker4DQL: Declarative Performance Measurement. In *Proceedings of the 2016 Symposium on Software Performance (SSP)*.
[2] Fabian Gorsler, Fabian Brosig, and Samuel Kounev. 2014. Performance Queries for Architecture-Level Performance Models. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM, New York, USA, 99–110.
[3] Nikolaus Huber, Fabian Brosig, Simon Spinner, Samuel Kounev, and Manuel Bähr. 2017. Model-Based Self-Aware Performance and Resource Management Using the Descartes Modeling Language. *IEEE Transactions on Software Engineering (TSE)* PP, 99 (2017). DOI: http://dx.doi.org/10.1109/TSE.2016.2613863
[4] Samuel Kounev and Alejandro Buchmann. 2006. SimQPN: A Tool and Methodology for Analyzing Queueing Petri Net Models by Means of Simulation. *Perform. Eval.* 63, 4 (2006), 364–394. http://dx.doi.org/10.1016/j.peva.2005.03.004
[5] Samuel Kounev, Nikolaus Huber, Fabian Brosig, and Xiaoyun Zhu. 2016. A Model-Based Approach to Designing Self-Aware IT Systems and Infrastructures. *IEEE Computer* 49, 7 (2016), 53–61.
[6] Philipp Meier, Samuel Kounev, and Heiko Koziolek. 2011. Automated transformation of Palladio component models to queueing Petri nets. In *19th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2011), Singapore.*
[7] Simon Spinner, Giuliano Casale, Xiaoyun Zhu, and Samuel Kounev. 2014. LibReDE: A Library for Resource Demand Estimation. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM Press, New York, USA, 227–228. http://doi.acm.org/10.1145/2568088.2576093
[8] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. 2012. 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)*. 247–248.
[9] Jürgen Walter, Simon Eismann, Nikolai Reed, and Samuel Kounev. 2017. Architectural Performance Model Extraction as a Service. In *Proceedings of the 2017 Symposium on Software Performance (SSP)*.
[10] Jürgen Walter, Maximilian König, Simon Eismann, and Samuel Kounev. 2016. PAVO: A Framework for the Visualization of Performance Analyses Results. In *Proceedings of the 2016 Symposium on Software Performance (SSP)*.
[11] Jürgen Walter, Dusan Okanovic, and Samuel Kounev. 2017. 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)*. ACM.
[12] Jürgen Walter, Christian Stier, Heiko Koziolek, and Samuel Kounev. 2017. An Expandable Extraction Framework for Architectural Performance Models. In *Proceedings of the 3rd International Workshop on Quality-Aware DevOps (QUDOS’17)*. ACM, 6.
[13] Jürgen Walter, Andre van Hoorn, and Samuel Kounev. 2017. Automated and Adaptable Decision Support for Software Performance Engineering. In *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools.*
[14] Jürgen Walter, Andre van Hoorn, Heiko Koziolek, Dusan Okanovic, and Samuel Kounev. 2016. Asking “What?”, Automating the “How?”: The Vision of Declarative Performance Engineering. In *7th ACM/SPEC Int. Conf. on Perf. Eng. (ICPE ’16)*.