A Generic Platform for Transforming Monitoring Data into Performance Models

Jonas Kunz
Karlsruhe Institute of Technology
76131 Karlsruhe, Germany
jonas.kunz@student.kit.edu

Christoph Heger
NovaTec Consulting GmbH
70771 Leinfelden-Echterdingen, Germany
christoph.heger@novatec-gmbh.de

Robert Heinrich
Karlsruhe Institute of Technology
76131 Karlsruhe, Germany
robert.heinrich@kit.edu

ABSTRACT

The performance of software systems is an ongoing issue in the industry, including the development of corresponding performance models. Recently several approaches for deriving such performance models from monitoring data have been proposed. A current limitation of these approaches is that most of them are bound to certain monitoring tools for providing the data, limiting their applicability.

We therefore propose a generic platform for transforming monitoring data into performance models, encapsulating these approaches for deriving performance models. This platform gives the flexibility of exchanging the monitoring tool or the used performance modeling approach, allowing more comprehensive performance analysis without additional manual transformation work. A seamless exchangeability of the performance modeling approach enables the generation of different types of performance models based on the same monitoring data, while the exchangeability of the monitoring tool enables the same approaches to be employed on a wider range of systems, as often the applicability of certain monitoring tools is limited by environmental properties. In addition, the generic nature of the platform aims to support the rapid development of prototypes of new, upcoming ideas within the context of performance modeling based on monitoring data.

During our evaluation we examine the quality of our approach in terms of accuracy and scalability. We show that our platform for transforming monitoring data into performance models scales with a very low overhead and that the results of the integrated performance modeling approaches are very accurate in comparison to the results of the non-integrated versions.

Keywords

Model Transformation, Application Monitoring, Usage Profile Extraction, Performance Model Generation

ICPE '17 Companion, April 22-26, 2017, L'Aquila, Italy

DOI: http://dx.doi.org/10.1145/3053600.3053635

1. INTRODUCTION

With the broad range of applications for software systems, the performance of these systems continues to be an important issue for the industry. This is especially the case for web-based business applications, like e-commerce software. In this field, the performance of the underlying system can have a direct impact on the business success, as customers expect fast response times as part of an appealing user experience, regardless of for example the amount of data to be transferred over the available network [3, 7].

To address the issue of software performance, a common approach is software monitoring. A wide range of tools exists for this task [14, 9, 2]. The metrics provided by these tools reach from plain response times to detailed end-to-end execution traces. These metrics can then be examined to uncover existing performance problems either through tool-guided or manual analysis.

A drawback of measurement based approaches is their reactive nature: performance problems can only be uncovered after they occurred and already had an impact on the running system. This is where performance modeling comes into play, as performance models can be used to predict performance issues possibly arising in the future. A current problem with performance models however is that they are often cumbersome to generate. The process of developing such models usually involves a lot of expert knowledge about the target domain an the modeled system. For this reason, a tendency of monitoring approaches and performance modeling approaches to converge has been identified by Woodside et al. [16]. The idea is to improve the quality of performance models and to ease their creation by taking monitoring data into account during the model generation process. Therefore, several approaches have recently been presented for transforming monitoring data into performance models. A current limitation is that these approaches are mostly bound to certain measurement environments: they usually require monitoring data of a specific tool and have to be configured using approach specific mechanisms. This restricts the applicability of the approaches to systems which are supported by the used monitoring solution.

The idea of this paper is to solve this issue by providing a common infrastructure for transforming monitoring data independently from the input format into performance models based on existing approaches. We present the *Monitoring Data Transformation Platform (MDTP)* to decouple the monitoring tool from the performance model generation approaches to make both, the tool and the modeling ap-

proach, exchangeable. Additionally, this allows to combine the benefits of the different model generation approaches by executing each on the same set of monitoring data. Another aspect of the generalization is to provide an infrastructure for quick prototyping of new ideas within the topic of performance modeling.

The main contributions of this paper are as follows:

- We propose a generic platform for transforming monitoring data into performance models, allowing the integration of existing and upcoming monitoring data based performance modeling approaches. Hereby, we decouple the model generation from the input monitoring data using a new, generic monitoring data format extending existing generalizations. This format is designed to shift the execution of common analysis tasks usually done by each performance modeling approach into our proposed platform, avoiding a duplicate execution of these tasks.
- For our platform, we provide proof-of-concept integration for two common monitoring tools and two recently presented performance modeling approaches which focus on the generation of system usage models.
- We give a short overview of an evaluation we perform of our proposed platform based on the implemented integrations presented in this paper.

2. STATE OF THE ART

The decoupling of the performance model generation approaches from their employed monitoring tools requires a universal, tool-independent format for exchanging monitoring data. Based on the data typically provided by monitoring tools and required for performance model generation, several requirements for this format can be deduced. We based this deduction on two specific performance model generation approaches, iObserve [4] and WESSBAS [15], however the requirements are kept generic to also be applicable on other approaches. The requirements (REQ) are as follows:

REQ1 The format has to be capable of representing trace-based monitoring data in a high level of detail, including complex inter-dependencies, like the hierarchical structure of method invocation sequences.

REQ2 The format must represent data on a high level of abstraction hiding the details of the used monitoring tool to capture the data. Therefore, typical analysis steps performed on raw monitoring data, like session and trace reconstruction from atomic events should not be necessary anymore.

REQ3 The format should provide a mechanism to not only represent control flow, but also other high level information, like observed structural changes (e.g., deployment changes) or other occurring environmental events, like incoming requests from users.

REQ4 The format should be extensible in order to support new upcoming approaches in the future and to support the integration of other existing approaches.

Several approaches for generalizing monitoring data have already been proposed. Among them is OMG's *Structured Metrics Meta-Model (SMM)* [10] and an extension to it, the *Measurement Architecture for Model-based Analysis*. Another similar approach is the *Software Measurement Framework* introduced by Mora et al.[8]. These meta-models can be used to represent data in a tool-independent manner and are designed domain-independent, therefore they fulfill **REQ2** and **REQ4**. However, these approaches focus on the representation of the monitoring algorithm, the description of how the monitoring data is derived. Their high level of abstraction prevents an straightforward realization of **REQ1** and **REQ3**, making them not suitable for our purposes. A more similar approach to the desired generic format is the *Common Trace API (CTA)*, also known as *OPEN.XTRACE* [12]. The CTA is an API for representing monitored execution traces in a tool independent fashion, therefore immediately satisfies **REQ1** and **REQ2**. Additionally, as the CTA is designed as an API it is very extensible, fulfilling **REQ4**. However, currently the representable data is limited to method traces. Other high-level data, like observed deployment changes, can not be directly represented and require further analysis of the data. For this reason **REQ3** is not fulfilled. A common modeling formalism meeting **REQ3** are event-driven state machines. For example, the UML state machines [11] are a widely used formalism for modeling the behaviour of a system, especially the reaction to events including environmental changes. We therefore decided to extend the CTA with an event modeling mechanism, which is outlined in the following section alongside with the structure of the MDTP.

3. PLATFORM DESIGN

In this section, we introduce the *Monitoring Data Transformation Platform (MDTP)* [6], our proposed approach for generalizing the transformation process from monitoring data to performance models.

The goal of the MDTP is to increase the applicability of existing approaches for generating performance models from monitoring data. This is done by encapsulating the import process of the monitoring data into a separate module for each monitoring tool. Similarly, each integrated approach for deriving the performance models gets encapsulated, allowing a combination of each monitoring tool with each approach with less effort.

The benefit of our approach can be illustrated by a simple scenario: the owners of an already productive software system decide to employ performance modeling to further improve their quality-of-service. Two performance modeling approaches are chosen for this purpose, e.g. one for building architecture level performance models (like iObserve [4]) and one for generating usage models (like Wessbas [15]). However, considering the state of the art these approaches are bound to certain monitoring tools which are very likely to be different or at least to require different formatted monitoring data. There are two possible solutions for this problem currently, which are both not feasible: (a) instrument the productive system with both monitoring tools required at the same time or (b) rewrite one of the performance modeling approaches to be compatible with the monitoring data required for the other one. Alternative (a) is very likely to induce a significant overhead on the system, which is a productive system in our scenario. Alternative (b) would solve the problem temporary, however the effort of rewriting an approach would be necessary again when another performance modeling approach is employed. A better so-

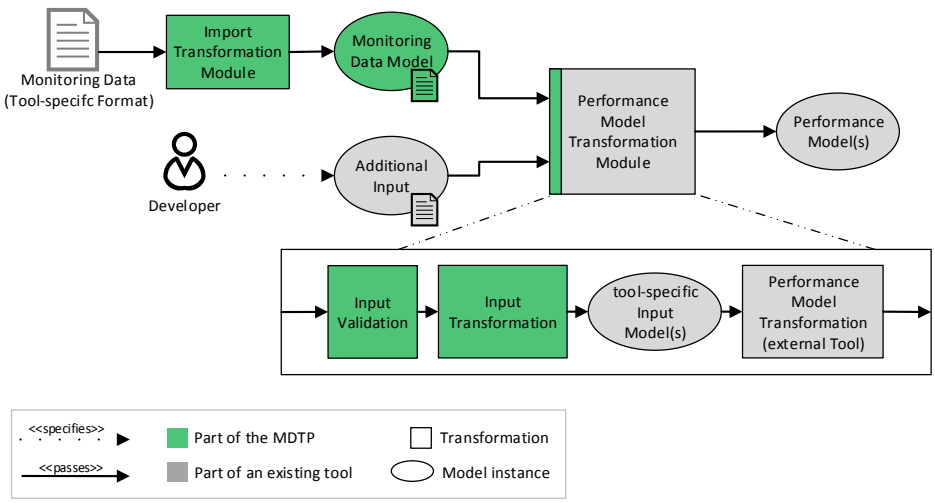

Figure 1: Monitoring Data Transformation Platform Structure

lution is to employ the MDTP: for the MDTP, the modules integrating each monitoring tool and performance modeling approach have to be written only once and can afterwards be combined in any way across different projects without additional effort. Therefore, the MDTP significantly lowers the barriers for employing performance modeling on real-world systems.

3.1 Structure

An overview of the proposed structure of the MDTP is shown in Figure 1. The process starts with monitoring data collected using a certain monitoring tool in it's specific format. This data then gets transformed by a corresponding **Import Transformation Module** into an instance of the proposed MDM. For each monitoring tool integration a separate, independent module is implemented. Similarly, the existing approaches for generating performance models from monitoring data each get encapsulated in an own **Performance Model Transformation Module**. Ideally, this type of modules is very lightweight: the MDTP aims to integrate existing approaches for deriving performance models, referenced as "external tool" part in Figure 1 without requiring many adaptions. The idea behind this encapsulation is to allow an eased exchange of the individual approaches to analyse the same set of monitoring data with different approaches. Most of these approaches require different input in addition to the monitoring data, like predefined architecture models of the target system. To allow the seamless integration, the modules offer an approach-independent solution for configuring this additional input data: each module supplies meta-information about the required additional input in a generic fashion through an extensible configuration API. Based on this API, tools can make use of the MDTP without any required knowledge of the internals of the used performance models generation approaches. For example, many approach require architecture models for generating their performance models, these can be supplied through this interface.

Before deriving the actual performance models, most approaches perform a preprocessing step and extract the required information from the monitoring data into an instance of an approach-specifc input model. Based on this common pattern, the Transformation Modules can usually be structured in a similar way: the first action performed by these modules is an **Input Validation** to check whether the provided MDM instance together with the approach-specific additional input artefacts are valid and provide the information required. Afterwards, the monitoring data alongside with the additional input artefacts are transformed into instances of these approach-specific input models. This significantly reduces the effort required to integrate such transformation approaches: After the specific input model has been derived, the module can execute the encapsulated approach without additional modifications. Example instantiations for monitoring data import modules and performance model transformation modules are given Section 4.

3.2 Monitoring Data Model

The MDTP requires a generalized monitoring data format for the exchange of monitoring data between the import modules and the transformation modules. Deducing from our analysis in Section 2, we decided to extend the CTA with an event modeling mechanism, resulting in our *Monitoring Data Model (MDM)*. These events are used to enrich the technically detailed monitoring data representation with more semantic information, which normally has to be manually derived by the performance modeling approaches: The MDM models the observed system as event-driven state machine. Therefore, we assume that any computation the observed system performs has been triggered by an event. Examples for such events are incoming HTTP user requests, operating system events or deployment changes. The triggered processing of these events can then in turn trigger other, internal events. For example, a call to a service within the boundaries of the system could be modeled as such an event. So in total, the MDM can be seen as a log of the execution of a state machine: we log all observed events and resulting processing into an MDM instance. The processing of the events in turn is represented using the CTA, which is referenced by attaching method invocation traces to events.

The MDM is modeled as an API, just like the CTA: we provide interfaces to interact with a model instance but do

not specify an implementation of it, as this allows the MDM to be extensible and allows the implementation of different technology specific versions of the MDM. However, for validation purposes we provide a default implementation of the MDM API[1].

4. PROOF-OF-CONCEPT INTEGRATION

In the previous sections we derived the requirements and the design of the MDTP based on the state of the art. As a proof-of-concept and as basis for the evaluation we provide an integration of two recent approaches for transforming monitoring data into performance models as well as two monitoring tools. For each, we provide a short introduction in this section. The goal is to give an overview of the conceptual differences in the integrated tools and approaches making a generalization using the MDTP beneficial.

The first integrated approach is iObserve [4]. iObserve aims to accompany the entire life cycle of an application in order to monitor changes of the system's environment and to predict necessary adjustments. The approach especially takes cloud applications into account with their very specific environmental properties, like the restricted knowledge about changes behind the facades of used services [5]. Based on monitoring data, iObserve currently generates Palladio Component Model [13] instances, manifesting observed structural changes as well as changes in the usage profile of the monitored system. Currently, the iObserve approach uses the Kieker Monitoring Framework [14] as monitoring solution. iObserve follows the common pattern of transforming the input monitoring data (form Kieker in this case) into a custom, preprocessed input model. This model mainly consists of events for representing deployment changes and an entry call sequence model. The entry call sequence model is used to model the interaction of users with the system: for each user session, the sequence of calls received by the observed system is stored.

The second approach we integrated is WESSBAS [15]. In contrast to iObserve, WESSBAS focuses entirely on the modeling of system usage based on observed user behaviour through monitoring data. As resulting performance models, WESSBAS outputs either architecture level usage models or load scripts for simulating the observed workload with load generator tools (e.g., Apache JMeter). WESSBAS also uses Kieker as monitoring solution and transforms the resulting monitoring data into a custom input model, so-called session logs. A session is a list of all user sessions observed. A user session consists of an ordered list of user requests with their mapping to the resulting system actions. As WESSBAS especially focuses on the observation of Web-based applications, HTTP protocol information, such as the request URI and parameters can be embedded into the log. This allows the generation of ready to use load scripts.

The MDM proposed in the previous section therefore is well suited for generating instances of the approach specific input models for iObserve and WESSBAS: in the case of iObserve, the events representing deployment changes can be directly translated into corresponding MDM events. User interaction with the system is modeled through HTTP events, which also store relevant information such as a session identifier and the URL. Having a trace of the resulting processing attached, the transformation into session logs or the entry call sequence model is straightforward.

To evaluate the exchangeability of the used monitoring tool for performing the analysis, we choose two integrate two tools. First we integrated the Kieker Monitoring Framework, as it is used by both WESSBAS and iObserve. Kieker stores the monitoring in form of basic records consisting of primitive type attributes, requiring processing for reconstructing their interdependencies. For example, a monitored method call sequence is represented by one record for each method call, making a tree reconstruction necessary before further analysis is possible. Though Kieker offers a framework for performing such common analysis tasks, these tasks are currently necessary to be performed by the performance model generation approaches, further binding them to Kieker. When integrated in the MDTP however, we employ these analysis within the import module responsible for transforming the Kieker data into an MDTP instance, which significantly increases the flexibility.

As second monitoring tool to integrate we choose inspectIT [9], as it fundamentally differs from Kieker. While Kieker was designed for extensiblity and flexibility, the main goal of inspectIT is usability. The monitoring data format of inspectIT, was not designed to leaf the inspectIT eco system, but instead to be analyzed with the included UI. Therefore, the monitoring format is very high level, already connecting method calls and meta information belonging together in so-called Invocation Sequences. This makes the integration of inspectIT fundamentally different from the integration of Kieker.

5. EVALUATION

Using the proof-of-concept integration from Section 4, we performed an evaluation of our concept. In this section we give a short overview of our validation, which is presented in more detail in our previous work [6].

5.1 Goals

The idea of the evaluation is to examine the capability of the MDTP to generalize the transformation process from monitoring data into performance models. As this generic goal cannot be directly quantified, we instead examine the provided integrations of inspectIT, Kieker, WESSBAS and iObserve into the MDTP as indicators.

The first goal we evaluate is the accuracy of the provided integrations. For Kieker and inspectIT, we evaluate how accurate the resulting MDMs represent the traced execution of the monitored system. For iObserve and WESSBAS, we want the integrated modules to behave as similarly as possible compared to the non-integrated approaches.

The second goal concerns the usability of the MDTP. To be applicable in the real world, the MDTP must not introduce a noteworthy overhead to the analysis tasks in terms of runtime. Therefore, the second goal is to evaluate the scalability of all implemented MDTP modules.

5.2 Experiment Environment

For the examination of the goals presented in the previous section, we focus on the monitoring and analysis of user behaviour on a experiment system. This is the core aspect which is common to both WESSBAS and iObserve, showing the benefit of a generalization.

[1]https://github.com/research-iobserve/monitoring-data-transformation-platform/tree/master-dev/mdm.default.impl

For the evaluation of the accuracy and the scalability, we require comparable and scalable sets of measurement data generated with both inspectIT and Kieker. These sets have to be comparable in the manner that they represent the same monitored system under the same environmental conditions (e.g. same hardware and same load).

For this reason, we setup a custom application on a local machine, which we instrumented with both inspectIT and Kieker. As application to monitor we choose "The Heat Clinic". "The Heat Clinic" is a Web shop which is provided as a demo application for the Broadleaf Commerce Framework [1]. Therefore, it offers the common functionality to users of managing an account, a shopping cart and executing orders. For this reason, we identified "The Heat Clinic" as a close to real world application which is capable of complex user interaction which we can use to generate monitoring data.

With one monitoring tool active, we set the system under load using Apache JMeter and record the monitoring data. JMeter is capable of simulating users by issuing series of HTTP requests to the experiment system which are specified in a load script. The goal of our load script is to produce load as similar as possible to real world users of the application. Therefore, we chose to model our users through a markov chain, which is closer to real-world users than a typically used linear script. This increased variance also helps uncovering possible inaccuracies introduced by the MDTP's transformations. Our modeled customer starts by browsing around the shop, searching and inspecting some products in more detail. Eventually the user will login at some point of time. Afterwards, the user continues browsing but then possibly adds items to his cart. At the end of his visit before logging off, the user then possibly performs a checkout.

We examine the MDTP modules using differently scaled sets of monitoring data. Hereby, we scale the data in two dimension: first, we vary the complexity of the usage script by either making JMeter execute all available user actions or by limiting it to just three simple browsing actions. Second, we scale the data in size by varying the experiment duration from 10 to 100 minutes.

5.3 Accuracy Evaluation

To examine the accuracy, first two sets of monitoring data are generated using our experiment environment, one with Kieker and one with inspectIT as monitoring tool. The load script is configured to make use of all available user actions behaviour and an experiment duration of 100 minutes. This results in the MDM having to manage more varying requests, which is more sensitive for possible errors which we want to uncover. The test duration was chosen to minimize the variance introduced by the randomized nature of the load script while keeping the monitoring data sets small enough to be safely handled by our system.

These two sets of monitoring data are then each translated into a MDM instance using the corresponding MDTP modules. To measure the accuracy of these MDMs we compare them to a reference model which represents the ideally expected result. As we modeled the input load to the system as a markov chain, this model can be calculated as the stable state of this markov chain. Our hypothesis in case of a correct integration is that the MDMs are equal to the reference model except some statistical variance. To perform this comparison, we used the following criteria:

- The difference between the distribution of HTTP request URIs in the MDM and the corresponding distribution in the reference model.
- The difference between distribution of observed transitions from one user action to another and the expected transition distribution in the reference model.

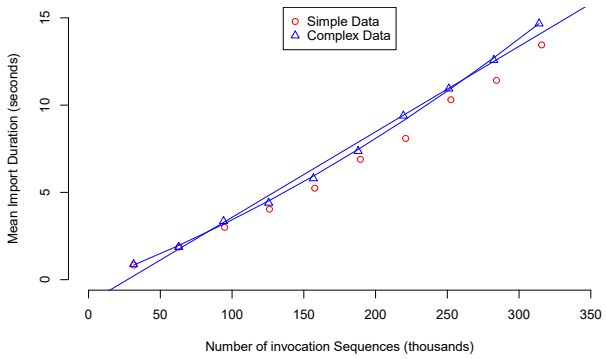

Figure 2: inspectIT Module scalability results. Linear and quadratic regressions lines included.

Based on these criteria, we showed by using the Chi-Squared test that the MDM instance generated from the Kieker data as well as the instance generated from the inspectIT data accurately represent the execution of our experiment system [6].

For evaluating the accuracy of the integration of WESSBAS and iObserve we used a similar approach. Both integrations were implemented by providing a transformation from the generic MDM to their specific input artefacts: for WESSBAS, these artefacts are session logs and for iObserve sets of entry call sequences and deployment events respectively. Afterwards, in both cases the MDTP executes the analysis approaches without any modifications. For this reason it is sufficient to analyse the transformation from the MDM into the approach-specific input artefacts. As both approaches use Kieker as monitoring solution in their non-integrated version, we reuse the Kieker data generated by our test environment as reference for the evaluation. The idea is to first generate these approach-specific input artefacts as reference models. We then perform a structural comparison of these reference models against the artefacts generated using our respective MDTP modules based on the MDM instance generated by the Kieker Import Module based on the same monitoring data.

This structural comparison of the artefacts showed that using the MDTP does not have a negative impact on the accuracy of the resulting performance models, as the input artefacts did not show any structural difference influencing the analysis.

5.4 Scalability Evaluation

To analyze the scalability of all four provided module implementations we reuse our experiment environment from the accuracy evaluation. We generate differently scaled sets of monitoring data for both inspectIT and Kieker by scaling (a) the experiment duration and (b) the complexity of the executed user actions. Based on these data sets, we benchmark the inspectIT and the Kieker modules.

As the transformation into MDM instances preserves the

size of the monitoring data (the transformation does not introduce data loss, as shown in the previous section), we generate differently scaled MDM instances using the Kieker Module and the generated Kieker monitoring data. These MDM instances of different sizes are then used to benchmark the WESSBAS and the iObserve module. For this benchmarking we focused on the transformation into the approach specific input artefacts, as we did not modify the actual analysis approaches which are executed afterwards.

Our analysis showed that the use of the MDTP does not introduce a noteworthy performance impact compared to using the standalone performance model generation approaches [6]: using least-squares regression, we can show that the Kieker, iObserve and WESSBAS Modules scale linearly with the size of the input data. For the inspectIT Module we can show at least an asymptotically quadratic behaviour, however without excluding the possibiltiy of a linear behaviour as shown by Figure 2. To give a finite answer for the inspectIT module, better hardware for the experiment system is necessary, as the benchmarking has to be performed for bigger input data sets, which our system was not able to handle due to memory limitations. With 16 gb of memory we were just able to process our largest monitoring data set, which has to be done in-memory currently due to API limitations of inspectIT. Removing this limitation to allow a streamed processing of the monitoring data is a future task.

6. CONCLUSION

In this paper, we proposed the *Monitoring Data Transformation Platform (MDTP))* alongside with the *Monitoring Data Model (MDM)* for solving the issue of current performance modeling approaches being bound to certain monitoring tools. The MDTP is designed as an extensible platform, allowing the integration of monitoring tools and performance modeling approaches in form of exchangeable modules, which employ the MDM as exchange format for monitoring data.

For a proof-of-concept we provided module implementations for two monitoring tools, inspectIT and Kieker, and two performance model generation approaches. As performance model generation approaches we chose to integrated the two state of the art approaches WESSBAS and iObserve.

The provided implementations were used to evaluate the MDTP: we evaluated the four modules in terms of scalability and accuracy. We showed that the platform is capable of executing different performance modeling approaches based on the same monitoring data, which was not possible without manual transformation work before the MDTP. Additionally, we showed that the MDTP made an exchange of the monitoring tool possible without affecting the analysis results, increasing the flexibility of the existing performance modeling approaches. Furthermore, through our scalability analysis of the modules, we deduced that the MDTP does not introduce a noteworthy overhead compared to the non-integrated analysis approaches.

7. ACKNOWLEDGMENTS

This work is being supported by the German Federal Ministry of Education and Research (grant no. 01IS15004, diagnoseIT), the DFG (German Research Foundation) under the Priority Programme SPP1593 and the Research Group of the Standard Performance Evaluation Corporation (SPEC).

8. REFERENCES

[1] Broadleaf Commerce, LLC. Open source enterprise ecommerce platform - broadleaf commerce. http://www.broadleafcommerce.com/, accessed 25.07.16.

[2] H. Eichelberger et al. Flexible resource monitoring of java programs. *Journal of Systems and Software*, 93:163 – 186, 2014.

[3] T. Everts. *Time is Money (Early Release)*. O'Reilly Media, 2015.

[4] R. Heinrich. Architectural run-time models for performance and privacy analysis in dynamic cloud applications. *SIGMETRICS Perform. Eval. Rev.*, 43(4):13–22, 2016.

[5] R. Heinrich et al. *Software Architecture for Big Data and the Cloud*, chapter An Architectural Model-Based Approach to Quality-aware DevOps in Cloud Applications. Elsevier, 2017. to appear.

[6] J. Kunz. A generic platform for transforming monitoring data into performance models. B. sc. thesis, Karlsruhe Institute of Technology, 2016.

[7] D. A. Menascé et al. A methodology for workload characterization of e-commerce sites. In *1st ACM Conference on Electronic Commerce*, pages 119–128. ACM, 1999.

[8] B. Mora et al. Model-driven software measurement framework: A case study. In *Ninth International Conference on Quality Software*, pages 239–248, 2009.

[9] NovaTec Consulting GmbH. inspectIT: Manage your Java Application's Performance. http://www.inspectit.rocks/, accessed 11.04.16.

[10] Object Management Group, Inc. Architecture-driven modernization (adm): Structured metrics meta-model (ssm), v. 1.1.1. http://www.omg.org/spec/SMM/, accessed 12.04.16.

[11] Object Management Group, Inc. Unified modeling language specification. http://www.omg.org/spec/UML/, accessed 05.04.16.

[12] D. Okanović et al. *Towards Performance Tooling Interoperability: An Open Format for Representing Execution Traces*, pages 94–108. Springer International Publishing, Cham, 2016.

[13] R. H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016.

[14] A. van Hoorn et al. Kieker: A framework for application performance monitoring and dynamic software analysis. In *3rd ACM/SPEC International Conference on Performance Engineering*, pages 247–248. ACM, 2012.

[15] A. a. van Hoorn. Automatic extraction of probabilistic workload specifications for load testing session-based application systems. In *8th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '14, pages 139–146, 2014.

[16] M. Woodside et al. The future of software performance engineering. In *Future of Software Engineering*, pages 171–187, 2007.