Integrating Software Performance Curves with the Palladio Component Model

Alexander Wert
SAP Research
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
alexander.wert@sap.com

Jens Happe
SAP Research
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
jens.happe@sap.com

Dennis Westermann
SAP Research
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
dennis.westermann@sap.com

ABSTRACT

Software performance engineering for enterprise applications is becoming more and more challenging as the size and complexity of software landscapes increases. Systems are built on powerful middleware platforms, existing software components, and 3rd party services. The internal structure of such a software basis is often unknown especially if business and system boundaries are crossed. Existing model-driven performance engineering approaches realise a pure top down prediction approach. Software architects have to provide a complete model of their system in order to conduct performance analyses. Measurement-based approaches depend on the availability of the complete system under test. In this paper, we propose a concept for the combination of model-driven and measurement-based performance engineering. We integrate software performance curves with the Palladio Component Model (PCM) (an advanced model-based performance prediction approach) in order to enable the evaluation of enterprise applications which depend on a large software basis.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques

1. INTRODUCTION

The performance (response time, throughput, and resource utilisation) of enterprise applications is crucial for their success. Performance directly influences the total cost of ownership (hardware and energy) as well as customer satisfaction. In order to develop performant applications, software architects need to evaluate the influence of different design alternatives on software performance, identify critical components, and plan capacities of their system early in the development cycle.

The increasing complexity of software systems makes design time performance analyses a difficult task that requires a high expertise in the system under study and performance engineering. Today's enterprise applications are built on a complex technology stack including powerful middleware platforms, different operating systems and virtualisation technologies. Furthermore, systems are placed in a software landscape with which they interact and on which they depend. The external systems largely contribute to the performance of an enterprise application and thus have to be considered for performance analyses. In most cases, no performance models are available for systems that have been developed over the past decades or are provided by 3rd parties. The necessary information to build performance models for these systems is not available or performance modelling is too much effort.

Model-driven performance engineering approaches [3] implement a pure top down approach and thus require performance models for all relevant parts of a system. This requirement can render their application impossible for software systems placed in a software landscape. Approaches for measurement-based performance evaluation (such as [19, 6]) require no (or at least no detailed) models of a system. However, they depend on the availability of large parts of the application and can only be applied later in the development cycle. During these stages, the evaluation of design decisions and the identification of critical components requires large and costly code changes and thus is inefficient. Other approaches combine performance prediction models with systematic measurements (such as [14, 12]) for resource demand estimation. However, these approaches still rely on knowledge about the internal structure of a system.

In this paper, we integrate model-driven and measurement-based performance prediction approaches. We introduce the concept of *software performance curves*, which describe the performance (response time, throughput, and resource utilisation) of a service or (sub-)system in dependence on its usage and configuration. Performance curves are inferred from systematic measurements using statistical methods and machine learning techniques. Performance curves characterise the response time, throughput, and resource utilisation of services which are available, but whose internal structure is unknown or too complex. To enable design-time performance analysis, we integrated performance curves into the Palladio Component Model (PCM) [4], an advanced approach for model-driven performance engineering. With the combination of model-driven and measurement-based performance analysis, software architects can evaluate design decisions, identify critical components, and plan capacities during early development stages including the effects of the system's environment.

The contribution of this paper is a concept for the integration of measurement-based and model-driven performance engineering approaches. Moreover, we introduce an interpreter of performance curves which extends the PCM model-to-text transformations that map a performance model to a discrete-event simulation.

ICPE'12, April 22-25, 2012, Boston, Massachusetts, USA
ACM 978-1-4503-1202-8/12/04.

This paper is structured as follows. Section 2 provides a brief overview of our approach. In Section 3, we present the concept of performance curves and provide an examples. Section 4 describes the integration of performance curves with the PCM. In Section 5, we describe related research. Finally, Section 6 concludes this paper.

2. OVERVIEW

In model-driven performance engineering (survey in [3] and [10]), architectural models of a software system are annotated with performance-relevant information such as resource demands and branching probabilities. These models are transformed to analytical models, such as stochastic Petri nets, stochastic process algebras, and queueing models (overview in [5]) or to discrete-event simulations [15, 13].

For the integration of model-driven and measurement-based performance analysis, we assume that some parts of the system are already available (for example, 3rd party services or software artefacts) and other parts are newly developed. The performance analysis follows the process shown in Figure 1.

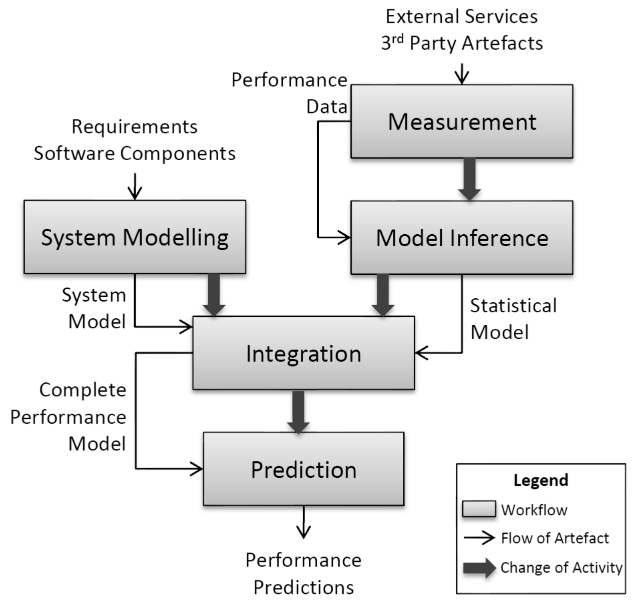

Figure 1: Overview of integrating model-driven and measurement-based performance analysis.

For the newly created parts, we can apply the standard approach for model-driven performance engineering [11]. Software architects specify the system's components, behaviour, deployment, and usage (*System Modelling*). This activity results in a *System Model* that describes the newly developed parts as well as its usage. In order to consider the effect of existing parts in performance analysis, we need to include them in the prediction model. However, modelling existing (sub-)systems can be difficult if the system has been developed by or is provided by a 3rd party. For in-house components, performance modelling can also become a difficult task due to the complexity and heterogeneity of systems. However, existing systems can be measured (*Measurement*) resulting in *Performance Data* of the system. Such data can be used for *Model Inference*. In this step, statistical methods and machine learning techniques derive a model of the system that describes its performance properties on an abstract level. Such models can be either simple performance models (such as used in [12]) or arbitrary statistical models (such as used in [7]). To consider the effect of system external parts on performance, these models have to be integrated with or made available in model-driven prediction approaches (*Integration*). This step merges both model types and creates a common basis for further performance analysis (*Prediction*). Based on the *Performance Predictions*, software architects and performance analysts can decide about design alternatives, plan capacities, or identify critical components. In the following section, we introduce the concept of *performance curves* illustrating it on an example.

3. SOFTWARE PERFORMANCE CURVES

In most cases, the necessary information to build performance models for 3rd party software systems is not available. Thus, we propose to specify the timing behaviour of such systems by more abstract models called *performance curves.* A performance curve describes the performance metrics of interest (e.g., response time) in dependence on a set of input parameters (e.g. number of requested table rows).

Formally, a performance curve describes the performance $\mathcal{P}$ (response time, throughput, and resource utilisation) of a system in dependence on a set of input parameters $A_1, \ldots, A_n$ with $n \in \mathbb{N}$. It is a function $f\colon A_1 \times A_2 \times \ldots \times A_n \to \mathbb{R}$, where each input parameter A_i is a number ($\subset \mathbb{R}$), an enumeration, or a boolean value. The function's result represents the performance metric of interest. A performance curve is inferred from systematic measurement of a software system using statistical methods [8], such as symbolic regression or multidimensional interpolation techniques like Kriging.

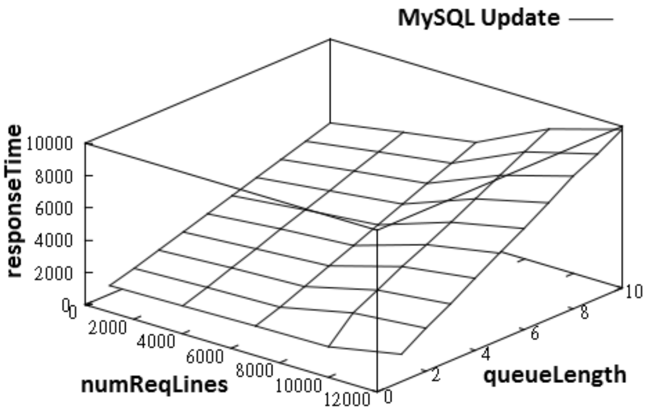

Figure 2: Performance curves for Apache Derby and MySQL.

In Figure 3, we depicted a simple example for a performance curve with two input parameters. Therefore, we measured the response time of a database systems' (MySQL 5.5 [1]) update operation in dependence of the queue length and the number of requested rows. We measured the response times for a queue length of 1 to 10 (step size 1) and for a number of requested rows between 1 and 12,000 (step size 3000). This example should only illustrate the idea behind performance curve. In most cases, accurate perfor-

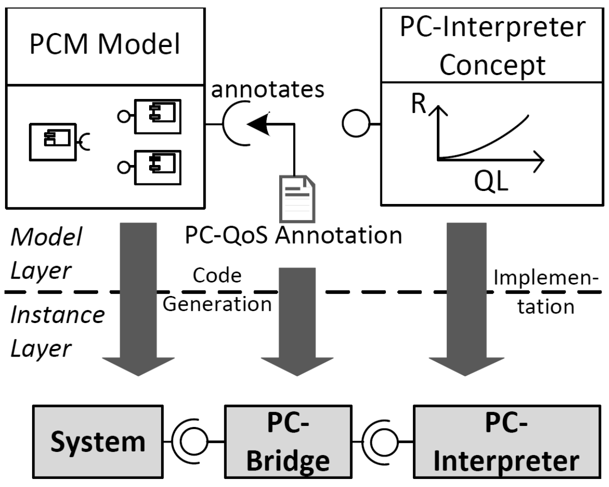

Figure 3: Integrating performance curves using QoSAnnotations.

mance curves of real software systems are much more complex and high dimensional. For the measurements, we used the *Software Performance Cockpit* [17, 16] applying the *FullExplorationStrategy*, which measures the performance of all combinations of possible parameter values. More sophisticated and efficient strategies have been presented in [18].

4. INTEGRATION CONCEPT

In the following, we introduce our concept for integrating performance curves with the Palladio Component Model and describe our extension of the transformations that map PCM-instances to a discrete-event simulation called SimuCom (for details about SimuCom see [4]).

We can integrate a performance curve into a component-based architecture using three different approaches: i) A performance curve models latencies and thus could be modelled as a specific kind of resource, ii) a performance curve describes the performance of a black-box component and thus could be specified as special components, and iii) performance curves model external services and components considering no further service dependencies and thus could specify the performance of system external calls. The first two approaches have significant drawbacks. Resources cannot provide the complex interfaces needed to capture the parameter dependencies of performance curves. Therefore, the first approach is not applicable. Furthermore, the performance of software components depends on their usage, deployment, and external services. Performance curves are currently not parametrised for arbitrary deployments or for the exchange of external services. Therefore, the second approach can lead to erroneous models and inaccurate predictions. In the following, we describe the integration of performance curves with model-driven prediction approaches as system external calls (iii).

Figure 3 gives an overview of the integration of performance curves into the PCM. At the model layer (top of Figure 3), PCM models with a system required role indicate their dependency on an external system. The external system is a black-box, whose internal structure and behaviour is unknown but a performance curve is available. To integrate a performance curve as an external system into the PCM, we extend its quality of service annotations (*QoSAnnotations*). *QoSAnnotations* specify the performance of system external calls by annotating the *RequiredRoles* and corresponding signatures of the required interface. We introduce the new annotation type *PerformanceCurveQoSAnnotation* (called *PC-QoSAnnotation* in Figure 3) which references the *RequiredRole* to be annotated and a *PerformanceCurveSpecification* (in the following called PC-SPEC) which describes the performance curve. Thus, the *PerformanceCurveQoSAnnotation* contains the information necessary to link the required role of the system to the performance information captured by the performance curve.

We designed and implemented a performance curve interpreter concept (*PC-Interpreter Concept*) providing an interface which is linked to the PCM's simulation. A *PC-Interpreter* (bottom right in Figure 3) loads a performance curve, determines response times and simulates delays. Essentially, the *PC-Interpreter* contains the generic logic for performance curve integration. A bridge (*PC-Bridge*) implements the logic necessary for adapting the application specific interfaces to the interface of the *PC-Interpreter*. This is necessary since the simulation-code generated from PCM models requires different interfaces than the *PC-Interpreter* provides. The interface and signatures which the required role of the PCM model comprises vary from case to case. Therefore, the *PC-Bridge* is generated for each application of a performance curve. In order to generate the code of the *PC-Bridge*, the *PerformanceCurveQoSAnnotation* and the contained *PerformanceCurveSpecification* are transformed to simulation code.

In the PCM, the simulation-code generation creates a Java class for each component of a system. For each SEFF-action and provided interface signature a method is generated containing the corresponding simulation code. The *PC-Bridge* can be integrated into the generated method of the external call (in the following called M) in order to redirect the simulation control flow to the *PC-Interpreter*. Then, the *PC-Interpreter* loads the specified performance curve, calculates the response time for the passed parameter values and simulates a delay. After that, the *PC-Interpreter* returns the control flow to *M*. The simulation engine measures the time for the execution of *M*, which can be used later for performance analysis and prediction.

5. RELATED WORK

In this section, we discuss research work dealing with measurement-based performance prediction and its combination with model-based approaches.

Woodside et al. [19] introduced the idea of a workbench for automated measurement of resource demands. The results are derived by function fitting and the maintenance of resource functions is done by a repository. Woodside assumes an analytical approach of performance consideration rather than using simulations. Furthermore, Woodside briefly mentions the idea of combining measurement results with system models by considering resource functions as model parameters. However, the measurements are used to refine an existing model and do not completely abstract from the internal behaviour of the (sub-)system.

Babka [2] introduced an approach for integrating (external) resource models into performance models based on queuing Petri nets. Babka also applies discrete-event simulation for performance prediction. However, in contrast to this paper, Babka focuses on low-level hardware and software resources like CPU caches or locks, whereas in our approach it does not matter whether the resource is a low-level resource like a CPU or a complex legacy system. Another difference is our assumption that the considered resource might be a black-box, i. e. the internal behaviour is unknown. A similar approach by Liu et al. [14] builds a queuing network model whose input values are computed based on benchmarks. The goal of the queuing network model is to derive performance metrics for J2EE applications.

Jin et al. [9] introduce an approach called BMM that combines benchmarking, production system monitoring, and performance modelling. Their goal is to quantify the performance characteristics of real-world legacy systems under various load conditions. However, the measurements are driven by the upfront selection of a performance model (e.g layered queuing network) which is later on built based on the measurement results.

6. CONCLUSIONS

In this paper, we presented a combination of model-driven and measurement-based performance analysis. Our approach allows software architects to combine models of a software system with performance curves of existing (sub-)systems. The latter are derived by systematic measurements using the Software Performance Cockpit [17, 16]. We integrated the performance curves into the Palladio Component Model (PCM) [4]. For this purpose, we extended the discrete-event simulation (SimuCom) of the PCM with an interpreter and adapter for performance curves.

The integration of model-driven and measurement-based performance analysis allows software architects to evaluate design alternatives, plan capacities, and identify critical components of a software system even though the system depends on external services, 3rd party components or legacy systems. Having this integration in place, performance engineering becomes more applicable in practice reducing the manual effort necessary to build prediction models.

In our future work, we look into issues that arise for the measurement of complex systems. With an increasing number of parameters, the number of measurements grows exponentially ("curse of dimensionality" [8]). A first measure to reduce the complexity is to identify and focus on the relevant parameters only. For this purpose, we look into different screening techniques that rank parameters according to their influence. Moreover, the integration of performance curves into the PCM requires further extensions. For example, performance curves have to consider interaction effects with other components or performance curves that occur if they share common resources. Last but not least, we need to gather more experience applying our approach in more complex scenarios within SAP.

7. REFERENCES

[1] Mysql 5.5. last visited 04.01.2011.

[2] V. Babka. Resource Sharing in QPN-based Performance Models. In J. Safrankova and J. Pavlu, editors, *In the Proceedings of WDS'08*. Citeseer, 2008.

[3] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

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

[5] M. Bernardo and J. Hillston, editors. *Formal Methods for Performance Evaluation (Int. School on Formal Methods for Design of Computer, Communication, and Software Systems, SFM2007)*. 2007.

[6] G. Denaro, A. Polini, and W. Emmerich. Early performance testing of distributed software applications. *SIGSOFT Software Engineering Notes*, 29(1):94–103, 2004.

[7] J. Happe, D. Westermann, K. Sachs, and L. Kapova. Statistical Inference of Software Performance Models for Parametric Performance Completions. In *Proc. of QoSA 2010)*, LNCS. Springer, 2010.

[8] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data mining, Inference ,and Prediction*. Springer, 2009.

[9] Y. Jin, A. Tang, J. Han, and Y. Liu. Performance evaluation and prediction for legacy information systems. In *Proceedings of 29th ICSE 2007*, Washington, DC, USA, 2007. IEEE Computer Society.

[10] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 2009.

[11] H. Koziolek, S. Becker, J. Happe, and R. Reussner. Life-Cycle Aware Modelling of Software Components. 5182, Oct. 2008.

[12] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson. Estimating service resource consumption from response time measurements. In *Proceedings of Valuetools 2006*, New York, NY, USA, 2006. ACM.

[13] P. L'Ecuyer and E. Buist. Simulation in Java with SSJ. In *Proc. of the 37th Conf. on Winter Simulation*, pages 611–620. WSC 2005, 2005.

[14] Y. Liu, A. Fekete, and I. Gorton. Design-Level Performance Prediction of Component-Based Applications. *IEEE Transactions on Software Engineering*, 31(11):928–941, 2005.

[15] B. Page and W. Kreutzer. *The Java Simulation Handbook. Simulating Discrete Event Systems with UML and Java.* 2005.

[16] D. Westermann and J. Happe. Software Performance Cockpit. http://www.sopeco.org/, 2011.

[17] D. Westermann, J. Happe, M. Hauck, and C. Heupel. The performance cockpit approach: A framework for systematic performance evaluations. In *Proc. of the 36th EUROMICRO SEAA 2010)*. IEEE CS, 2010.

[18] D. Westermann, R. Krebs, and J. Happe. Efficient experiment selection in automated software performance evaluations. In *EPEW '11: Proc. of the 8th European Performance Engineering Workshop*, Berlin, Heidelberg, 2011. Springer.

[19] C. M. Woodside, V. Vetland, M. Courtois, and S. Bayarov. Resource function capture for performance aspects of software components and sub-systems. In *Performance Engineering, State of the Art and Current Trends*, London, UK, 2001. Springer-Verlag.