Modeling and Predicting Performance Impacts in a Service-oriented, Industrial Software System from the Automation Domain[1]

Heiko Koziolek, Roland Weiss, Jens Doppelhamer

Industrial Software Technologies
ABB Corporate Research, Forschungszentrum Deutschland
Wallstadter Str. 59
68526 Ladenburg
heiko.koziolek@de.abb.com
roland.weiss@de.abb.com
jens.doppelhamer@de.abb.com

Abstract: Industrial software systems today have reached sizes and complexity such that introducing changes like adding new features or fixing bugs requires significant investments. How these changes affect system qualities like performance or maintainability is typically not know a priori and therefore increase the risks for the investment. In this work we show how based on architecture models early estimations of performance become possible. We present an industrial demonstrator from the automation domain for model based performance prediction. First, the architecture models of a service in a service-oriented automation system are manually created and annotated with performance parameters. After that, we compare performance predictions with actual measurements. The used analysis tools allow performance estimates of implementation, usage, or deployment changes to the system, relying on the created architecture models. The results of the initial demonstrator incarnation look promising, as the deviation of the performance estimations are below 10%.

[1] This work is supported by the European Union under the ICT priority of the Seventh Research Framework Program contract FP7-215013.

1 Introduction

The increasing use of model-driven methods in software development [SVC06] creates the desire to use the designed models not only for code generation, but also for analyzing the expected non-functional properties of a system [BDI04, Gok07]. For example, performance-annotated design models can help developers to predict the response times, throughput, and resource utilizations of their systems. These predictions can be used to evaluate architectural design alternatives during system development and maintenance activities, thus reducing the risks for the required investments of the development project.

While this approach is highly beneficial during early development stages to rule out designs with poor non-functional properties, it is also applicable after system implementation. With reverse engineered models from a running system, it is possible to predict the impact of system changes to the non-functional properties on the model level before actually implementing them. The models are on a higher abstraction level than the code, and are therefore helpful in reducing the complexity of the analyses.

Existing, classical performance models based on queuing networks (e.g., [SW02]), stochastic Petri nets (e.g., [LTK02]), or stochastic process algebras [HHK02] are not directly applicable on today's service-oriented software systems. First, their basic modeling elements are not aligned with elements from service-oriented systems and are therefore difficult to understand and create. Second, there are hardly any approaches for creating such models automatically based on reverse engineering methods.

Special performance models for component-based and service-oriented software systems (e.g., KLAPER [GMS07], PALLADIO [BKR09]) use classical performance models internally, but model systems more aligned with typical software modeling languages (e.g., UML), which makes them easier to understand and enables the use of reverse engineering methods. However, their industrial applicability is still unknown.

We report on current ongoing work in the EU project Q-ImPrESS (Quality Impact Prediction for Evolving Service-oriented Systems)[2]. In the course of the project a new modeling language for performance, reliability, and maintainability of service oriented-systems is being created. It shall combine the advantages of different existing methods. To assess the applicability and benefits of the model-driven quality analysis method, the industry partners in the project build multiple demonstrators.

[2] http://www.q-impress.eu

This paper reports on the initial implementation of a demonstrator for analyzing the performance impacts of different change scenarios in a service-oriented system from the automation domain. As a first step, we have manually created performance models by instrumenting the code of the system and executing different test cases. In a second, future step, the models shall be created by reverse engineering tools. The performance models can be used to predict the response times and resource utilizations of the system for different usage profiles and deployment scenarios.

The remainder of the paper is organized as follows: Section 2 provides a quick overview of the Q-ImPReSS project. Section 3 describes the demonstrator implementation and measurement studies, which were used to construct a service-oriented performance model. Section 4 presents some initial prediction results, before Section 5 concludes the paper.

2 Overview of the Q-ImPrESS project

The EU FP7 STREP project Q-ImPrESS has four academic partners (Charles University Prague, Politecnico di Milano, Forschungszentrum für Information Karlsruhe (FZI), and Mälardalen University Västerås) and four industrial partners (ABB, Itemis, Ericsson Nikola Tesla, Softeco).

The overall goal of the project is to define a new service engineering method to create and evolve service-oriented software with predictable end-to-end quality. The key constituents for reaching this goal are:

- Support for the evolution of service-oriented software through what-if analyses by predicting the impact of design decisions on performance, reliability and maintainability based on architecture models,
- Highlight trade-offs between these quality attributes for alternative architectures such that conscious decisions are made traceable back to the priorities of the quality attributes,
- Facilitate the reengineering of legacy code to service-oriented architectures by iterative model abstractions form implementation to architecture models.

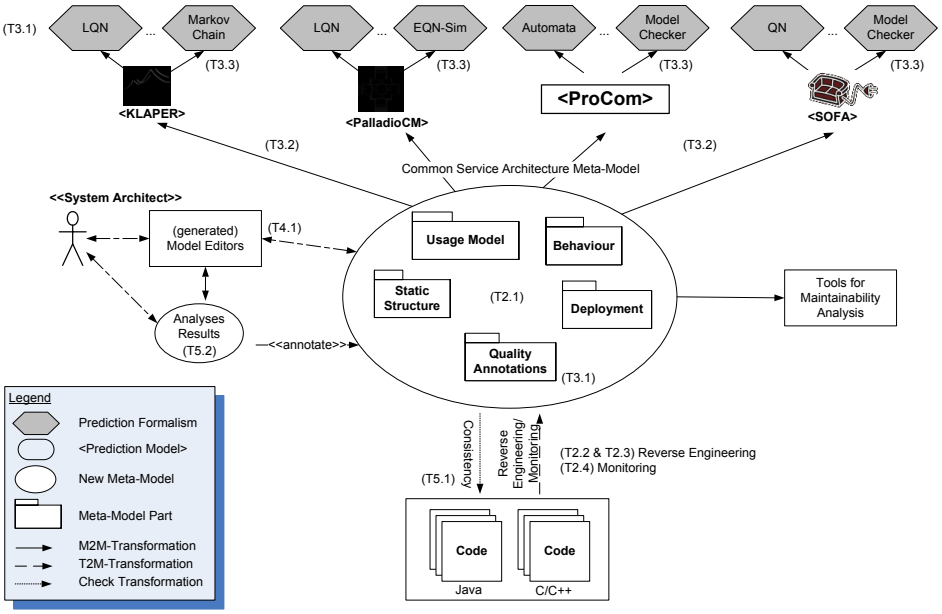

Figure 1: Q-ImPrESS Overview, planned Models and integrated tools

At the core of the Q-ImPrESS methodology is a new service architecture meta model (SAMM), see Figure 1. The meta model consists of five different submodels according to parts of the system that can be built by separated developers:

1. A usage model for call frequencies and input parameters
2. A behavior model for control flow within and between services
3. A static structure model for services, components, and connectors
4. A deployment model for the hardware resource and the mapping of services to hardware resources
5. A quality annotation model for execution times, failure probabilities, and costs

There are two main means to create instances of architecture models according to this meta model. On the one hand, they can be created and edited with graphical modeling tools. This works well for new development projects and allows early reasoning about design decisions at the architectural level. On the other hand, models can be obtained by reverse engineering legacy code (Java, C++, Delphi) through a semiautomatic process. A GAST representation is extracted from the source code, abstracting behavior within components. The SoMoX software model extractor then mines for components on this representation iteratively, taking input from users to guide the clustering and component identification.

The complete model is transformed into existing meta models, i.e. to KLAPER for performance and reliability, to Palladio for performance, reliability, and maintainability, to ProCom for real-time performance, and to SOFA for protocol correctness checking. The model can omit some data if not all analysis capabilities in Q-ImPrESS are relevant, e.g. information from the deployment model is not required for protocol checking.

At the moment, the user has to run the analysis tools consecutively to predict the impact on different quality attributes. It is planned to support tradeoff analyses with a dedicated tool to assess the interdependencies between performance/reliability/maintainability with respect to different design alternatives.

3 Demonstrator Implementation

The goal of our demonstrator implementation is to validate the accuracy of the quality predictions of the Q-ImPrESS method and tools on an industrial-sized system as well as the applicability in terms of the required effort to achieve accurate predictions. With the final demonstrator, we want to assess the quality impact of different change scenarios with respect to performance, reliability, and maintainability, and perform trade-off analyses for different design alternatives on the model level.

As the reverse engineering tools from the academic partners are still under development, we opted to create a demonstrator model manually in a first iteration of the validation phase. This helps us to evaluate the involved meta-models for their modeling expressiveness and to initially assess the possibilities for getting the necessary data from a running system.

We chose a service-oriented, distributed control system from ABB as the system under study for our demonstrator implementation. The system consists of controllers with embedded real-time software, as well as PC software implemented in C/C++ running on a general purpose operating system.

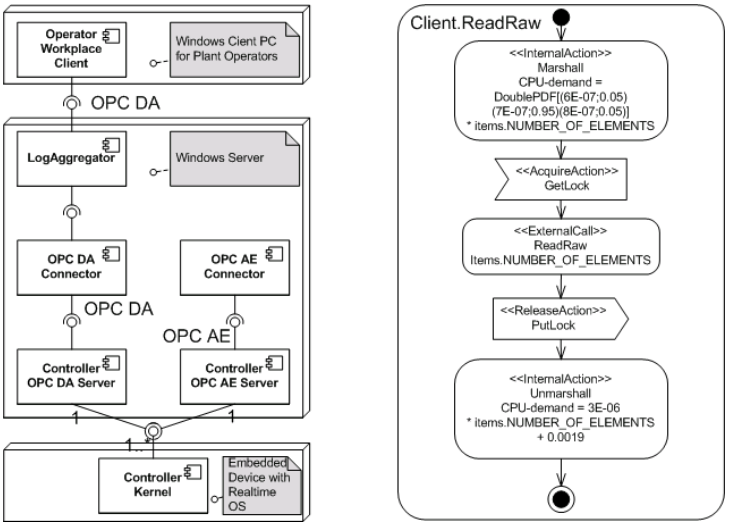

Figure 2: Demonstrator Static Structure Model (left), Behavior Model (right)

While the whole system consists of several million lines of code, we focused our analysis on a service called 'Log Aggregator' with ca. 30 KLOC. This service runs in the so-called 'Data Integration Layer' and manages logs of recorded signal data from an industrial production process. Clients (e.g. HMIs for plant operators) can request an arbitrary number of data values (items) from the data integration server, process the logged data, and generate trend charts via the service.

Fig. 2 (left) shows part of the static structure model of the Log Aggregator. The data integration server is connected via a standardized protocol (OPC DA) to a number of controllers, which may send thousands of items per second to the server. Fig. 2 (right) shows part of the behavior model of clients of Log Aggregator. The clients first perform marshalling, and then acquire a synchronization lock, before sending their requests to the server. After receiving the response from the server, they release the lock and unmarshal the data.

We instrumented the code of the 'Log Aggregator' and inserted numerous measurement points into the implementation. As a behavioral model of the service was not available, we first had to reconstruct it. We used high precision counters from Windows for generating timestamps. After some tests, we found that the execution time of the system can be abstracted into three distinctive resource demands: client CPU demand before sending a request, server CPU demand, and client CPU demand after getting a response.

Fig. 3 shows the measurement results for the three different resource demands in relation to the number of requested items. We repeated the test cases multiple times to derive reliable results for the average execution times. The results indicate that the dependency between the number of requested items and the execution times is linear. We used this data to construct a performance model with the SAMM.

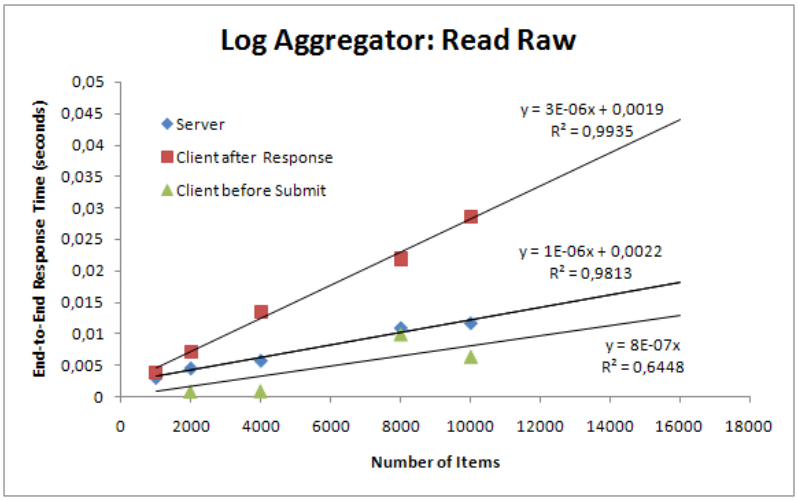

Figure 3: Demonstrator Measurement Results

4 Prediction Results

For the performance predictions, we relied on the Palladio component model [BKR09] (PCM) and its built-in extended queuing network simulator. The PCM is connected to the SAMM of Q-ImPrESS via a model transformation as depicted in Figure 1. We simulated the performance model with the same number of clients and number of requested items as in our measurement study. Then, we compared the mean response times predicted by the simulation and the mean response times measured (Table 1). The deviation is below 10 percent in all cases and therefore we deem the model sufficiently accurate for extrapolation.

Table 1: Prediction vs. Measurement for the Initial System

Number of Items	Predicted Mean Response Time	Actual Mean Response Time	Error	Error (%)
1000	0,0089	0,009441	0,000541	5,73
2000	0,0137	0,014676	0,000976	6,64
4000	0,0233	0,024475	0,001175	4,80

8000	0,0425	0,040395	-0,0021	5,20

In the following, we report on quality impact predictions for different evolution scenarios involving (i) changing the implementation, (ii) changing the client workload, and (iii) changing the resource environment:

Changing the Implementation: After analyzing the code of the LogAggregator it is expected that the processing overhead per item can be reduced by 50 percent from 0.0022 seconds to 0.0011 seconds by optimizing the involved algorithms. Using our performance model, we can predict the impact of this change to the overall response time perceived by the clients before actually implementing the change. After adapting the model to the new resource demand for the server component, the predicted mean response time for requesting 8000 items is 0.040 seconds instead of 0.042 seconds in the original system. The result indicates that this optimization yields only a small overall impact on the performance.

Changing the Client Workload: Because of the evolution of the underlying industrial system in terms of sensors and throughput, it is expected that the number of requested items will increase beyond 8000 items per requests. Using our model, we found that the mean client response time will be 0.079 seconds for 16000 requested items and 0.154 seconds for 32000 requested items. Additionally, it is expected to have clients with multithreaded access to the LogAggregator in the future to update multiple trend charts in parallel. Our model predicts that clients initiating two threads at a time for 8000 items will receive a mean response time of 0.060 seconds, while clients initiating four threads at a time would receive their responses after 0.130 seconds.

Changing the Resource Environment: Both the Operator Workplace Client and the Log Aggregator run on Intel Core 2 Duo PCs with 2.66 GHz initially. Our model can predict the performance impact of using faster hardware when evolving the system. If the server PC is changed to an Intel Core 2 Duo PC with 3.33 GHz, the overall mean response time for requesting 8000 items will decrease from 0.042 seconds to 0.040 seconds. If the client PC is changed in the same way the response time will decrease to 0.036 seconds, because of its higher processing load. If both PC are changed to the faster processors, the response time will be 0.034 seconds. Additionally the impact for saving hardware can be predicted. If both the Operator Workplace Client and the Log Aggregator are deployed on the same PC (2.66 GHz) the response time stays the same for single-threaded access, but increases to 0.090 seconds for 2 parallel threads or 0.170 seconds for 4 parallel threads.

5 Conclusions

We have presented a demonstrator implementation for a model-driven approach called Q-ImPrESS for predicting quality impacts in service-oriented systems. The involved performance models have been created with the newly introduced service architecture meta-model. As a first step, we have shown how to construct an instance of the model manually and what can be achieved with such a model when being applied on a realistic, industrial system.

The approach shall help service developers to specify the performance properties of their services and software architects to assess the performance of a service-oriented design before implementation or after applying changes to an existing implementation. This paper serves as a first proof-of-concept case study, demonstrating that the envisioned approach is in principle applicable in practice, even on large-scale systems.

Our models for the industrial system are still limited in their expressiveness and shall be extended in future work. We will incorporate more hardware resources (e.g., storage devices) into the model, and model the performance properties of other services in the system. So far, we have built the models manually, but for a next iteration we will apply reverse engineering tools provided by the academic partners to decrease the time for creating the models from legacy systems. Furthermore, besides performance, also the reliability and maintainability of the demonstrator system shall be analysed with similar Q-ImPrESS tools.

Literaturverzeichnis

[BKR09] Becker, S.; Koziolek, H.; Reusser, R.: The Palladio Component Model for Model-Based Performance Prediction. In Elsevier Journal on Systems and Software, Vol 82, No. 1, pp. 3-22, January 2009

[BDI04] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction inSoftware Development: A Survey. IEEE Transactions on Software Engineering, 30(5):295-310, May 2004.

[SVC06] T. Stahl, M. Voelter, and K. Czarnecki. Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons, 2006

[SW02] C. U. Smith and L. G. Williams. Performance Solutions: A Practical Guide to Creating Responsive,cScalable Software. Addison-Wesley, 2002.

[LTK02] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O.P. Waldhorst. Performance analysis of time-enhanced uml diagrams based on stochastic processes. In ACM Proceedings of the International Workshop on Software an Performance, pages 25–34, 2002

[HHK02] H. Hermanns, U. Herzog, and J.P. Katoen. Process algebra for performance evaluation. Theoretical Computer Science, 274(1-2):43–87, 2002.

[Gok07] S. S. Gokhale: Architecture-based software reliability analysis: Overview and limitations," IEEE Trans. on Dependable and Secure Computing, vol. 4, no. 1, pp. 32–40, January-March 2007.

[GMS07] Vincenzo Grassi, Raffaela Mirandola, and Antonino Sabetta. Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. Journal on Systems and Software, 80(4):528–558, 2007.