Reusable QoS Specifications for Systematic Component-based Design

Lucia Kapova
Software Design and Quality Group,
Karlsruhe Institute of Technology (KIT),
Germany
Email: kapova@ipd.uka.de

ABSTRACT

For successful and effective software development the ability to predict impact of design decisions in early development stages is crucial. Typically, to provide accurate predictions the models have to include low-level details such as used design patterns (e.g., concurrency design patterns) and underlying middleware platform. These details influence Quality of Service (QoS) metrics, thus are essential for accurate prediction of extra-functional properties such as performance and reliability. Existing approaches do not consider the relation of actual implementations and performance models used for prediction. Furthermore, they neglect the broad variety of implementations and middleware platforms, possible configurations, and varying usage scenarios. To allow more accurate performance predictions, we extend classical performance engineering by automated model refinements based on a library of reusable performance completions.

Categories and Subject Descriptors

D.2.2 [**Software**]: Software Engineering

General Terms

Performance Engineering, Model-Driven Development

Keywords

Model Refinement, Completion, Palladio Component Model

1. INTRODUCTION

In software performance engineering, abstract design models are used to predict and evaluate response time, throughput, and resource utilisation of the target system during early development stages prior to implementation. In model-driven (or model-based) software performance engineering [1], software architects use architectural models of the system under study and base their analyses on them. Transformations map the architectural models to simulation-based or analytical prediction models, such as queueing networks, stochastic Petri nets, or stochastic process algebras. However, to provide accurate predictions, performance models have to include many low-level details. For example, the configuration of a message-oriented middleware (e.g., a size of a transaction) can affect the delivery time of messages [5]. Unfortunately, software architects cannot include these details into their architectural models. The middleware's complexity and the specific knowledge on the implementation (that is required to create the necessary models) would increase the modelling effort dramatically. While most of the implementation details are not known in advance, a rough knowledge about the design patterns that are to be used might be already available. This knowledge can be exploited for further analysis, such as performance and reliability prediction, and for code generation.

Including low-level details in prediction models conflicts with the abstract architecture paradigm and leads to a significant effort for software architects. Additionally, such models are very complex leading to a decreased understandability, reusability and models credibility. The resulting **challenge** in a form of conflict between including low-level details into prediction models and maintaining highly-abstract models is addressed by this work. The presented solution is based on the parametrized model completions that include extra-functional properties of lower levels into high-level architecture. We call model refinements that specifically address quality attributes of software systems *completions* [6]. In the original approach of Woodside et al. [6], performance completions have to be added manually to the prediction model. Model completions express low-level details as reconfigurable black-box constructs. This way the resulting model complexity is hidden from software architects. Software architects only have to provide a configuration of modeled low-level detail. To support software architects in building more accurate prediction models, they need a library of such completions (e.g., configurations models for different middleware platforms) to build on.

In literature, this problem was already identified. However, the proposed approaches suggest only annotation models that extend prediction models through parametrization of resource demands by results of measurements on real systems (in the case of performance prediction, for example, number of processor cycles). They concentrate on the properties of the underlying platform and do not consider architectural changes such as inclusion of certain design pattern (such as Replication, Barrier, Connector patterns etc). One reason why such details are not considered is the high level of variability in the architecture that would be required. It is not feasible to create such models manually. Therefore, for building such detailed models the automation of their development process is crucial, the lack of automation for performance modelling has been clearly stated in [6]. However, to automate this process we have to deal with a classical problem of a conflict between variability and automation. The solution for this problem is based on the transformation generation by Higher-Order Transformations (HOT).

The difficulty of automation is a result of the flexibility and variability required for performance completions [6]. In order to provide tool support and to apply performance completions, we have to address this problem. Model-driven development can provide the

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.
ACM 978-1-4503-0519-8/11/03.

needed automation by means of model transformations. For example, the authors of [2] analyse design patterns for Message-oriented Middleware. They use the selected combination of messaging patterns as configuration (also called mark model) for model-to-model transformations. Basically, existing solutions [2, 4] focus on the integration of only one completion at a time. The scenarios where more than one completion is applied to model element are discussed in [3].

We present an approach to define domain-specific languages that capture the performance-relevant configurations of different implementation details. The configuration (feature model) provides the necessary variability. The transformations are applied to model elements specified by the software architect. We realised the completions by means of model-to-model transformations. Depending on a given configuration, these transformations inject the completion's behaviour into performance models.

2. THE CHILIES APPROACH

The contribution of this work is a novel approach called CHILIES to automated feature model-based generation of refinement transformations that are used to integrate variants of model completions into prediction models. The feature model is used as a definition of the variation space. The completions in a completion library are implemented in a form of feature models with so-called feature effects specifications. Considering that a model could require more than one completion to be integrated the approach has to deal with the *chains* of such refinement transformations. Furthermore, this approach was extended by domain-specific reusable templates for architectural refinement transformations. This way it is easier for the developer of the feature model (that defines possible configuration options of modeled details) to specify the effects of feature selection on the transformed model.

The automated model refinement process is based on a chain of Higher-Order Transformations (HOTs): the first one for transformation synthesis from the feature model; the second one for transformation composition based on the structure of completion library (mapping the metamodel structure) and the rule-based composition; the third one for the instantiation of parametrized domain-specific templates as a partial transformation synthesis. This process is integrated in the Palladio Component Model (PCM). The tool takes a complete PCM instance as input and produces a new PCM instance by refining the model based on a configuration of completion variant.

Unlike existing approaches, CHILIES, does not require heavy development effort, which limits what can be attempted. Introduced solution automates development and integration of performance abstractions in form of completions into the prediction models and closes the semantic gap between performance concerns and functional concerns, which prevents many developers from addressing performance at all. For the same reason many developers do not trust or understand performance models, even if such models are available. Performance modeling is effective but it is often costly; models are approximate, they leave out detail that may be important, and are difficult to validate. Consequently, Chilies provides more accurate predictions, decrease development costs, increase usability and maintainability of prediction models.

To **validate** performance prediction capabilities of presented approach for a systematic refinement of performance models was an initial set of completions for concurrent component-based systems in a completion library introduced, where were design patterns for concurrency (such as Replication, Internal State, Locking) analysed. Additionally, completions for connectors were included to illustrate the high-level of variability in the architecture that is in this case required. To validate completions the prediction results based on the refined model were compared to the measurements on the real systems. The validation was performed in an end-to-end manner, by using the PCM workbench extensions.

Finally, for a completeness of introduced approach, this work discusses the quality of generated transformations. The transformations maintainability is evaluated through a set of introduced code metrics for model-to-model transformations. In the analysis the classical parametrized model transformations are compared to the generated transformations by introduced HOT-based approach. Lastly, the collection of transformation metrics is automated by a higher-order transformation, too (transforming into the metrics model).

3. CONCLUSIONS AND VISIONS

The automated integration of performance completions helps software architects and performance analysts to systematically design and apply performance completions. Performance completions reduce the necessary modelling effort (for software architects and performance analysts) as well as the complexity of the software architecture models. The transformation include the necessary information about low-level details and allows more accurate performance predictions. The presented work is a part of continuous research on the automatic transformation composition and generation. Support for presented techniques would allow performance architects to evaluate different implementation variant for combination of different sets of completions.

4. REFERENCES

[1] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 2004.

[2] Jens Happe, Holger Friedrich, Steffen Becker, and Ralf H. Reussner. A Pattern-Based Performance Completion for Message-Oriented Middleware. In *International Workshop on Software and Performance (WOSP '08)*. ACM, 2008.

[3] Lucia Kapova and Steffen Becker. Systematic refinement of performance models for concurrent component-based systems. In *International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA)*. Elsevier, 2010.

[4] Lucia Kapova and Thomas Goldschmidt. Automated feature model-based generation of refinement transformations. In *EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2009.

[5] Lucia Kapova, Barbora Zimmerova, Anne Martens, Jens Happe, and Ralf H. Reussner. State dependence in performance evaluation of component-based software systems. In *International Conference on Performance Engineering (WOSP/SIPEW '10)*. ACM, 2010.

[6] Murray Woodside, Greg Franks, and Dorina C. Petriu. The Future of Software Performance Engineering. In *International Conference on Software Engineering (ICSE)*. IEEE, 2007.