Performance Evaluation ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Parametric performance completions for model-driven performance prediction

Jens Happe [a,*], Steffen Becker [a], Christoph Rathfelder [a], Holger Friedrich [b], Ralf H. Reussner [c]

[a] Forschungszentrum Informatik – FZI, 76131 Karlsruhe, Germany
[b] Andrena objects, 76131 Karlsruhe, Germany
[c] Universität Karlsruhe (TH), 76131 Karlsruhe, Germany

ARTICLE INFO

Article history:
Received 26 January 2009
Received in revised form 22 July 2009
Accepted 23 July 2009
Available online xxxx

Keywords:
Model-driven performance engineering
Software performance engineering
Performance completion
Message-oriented middleware
Software architecture

ABSTRACT

Performance prediction methods can help software architects to identify potential performance problems, such as bottlenecks, in their software systems during the design phase. In such early stages of the software life-cycle, only a little information is available about the system's implementation and execution environment. However, these details are crucial for accurate performance predictions. Performance completions close the gap between available high-level models and required low-level details. Using model-driven technologies, transformations can include details of the implementation and execution environment into abstract performance models. However, existing approaches do not consider the relation of actual implementations and performance models used for prediction. Furthermore, they neglect the broad variety of possible implementations and middleware platforms, possible configurations, and possible usage scenarios. In this paper, we (i) establish a formal relation between generated performance models and generated code, (ii) introduce a design and application process for parametric performance completions, and (iii) develop a parametric performance completion for Message-oriented Middleware according to our method. Parametric performance completions are independent of a specific platform, reflect performance-relevant software configurations, and capture the influence of different usage scenarios. To evaluate the prediction accuracy of the completion for Message-oriented Middleware, we conducted a real-world case study with the SPECjms2007 Benchmark [http://www.spec.org/jms2007/]. The observed deviation of measurements and predictions was below 10% to 15%.

1. Introduction

Software architects can use design-time performance predictions to evaluate the resource utilisation, throughput, and timing behaviour of software systems prior to implementation. Using predictions, potential problems, such as bottlenecks and long delays, can be detected early avoiding costly redesigns or re-implementations in later stages. 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 (overview in [2]). The results derived of the prediction point out potential performance problems.

In general, the architectural models have to remain abstract since not all information is available about the final implementation and execution environment. However, such details can be of crucial importance for accurate

* Corresponding author. Tel.: +49 721 9654 634
E-mail addresses: jhappe@fzi.de (J. Happe), sbecker@fzi.de (S. Becker), rathfelder@fzi.de (C. Rathfelder), holger.friedrich@andrena.de (H. Friedrich), reussner@ipd.uni-karlsruhe.de (R.H. Reussner).

0166-5316/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2009.07.006

performance predictions. 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.

Woodside et al. proposed performance-related completions to close the gap between abstract architectural models and required low-level details [3]. They use completions to add performance influences of a system's infrastructure to prediction models and, thus, increase prediction accuracy. Later approaches (e.g., [4–6]) extend this idea using model-driven technologies. However, all of these approaches still neglect crucial aspects that hinder their application in practice. First, they do not establish a defined relation of the performance model used for prediction and the actual implementation. For accurate performance predictions, there must be a fixed relation between both. Otherwise, the decisions made during implementation can significantly change the system's performance. Second, existing approaches do not take into account different configurations of execution environments and generated code. For example, Message-oriented Middleware platforms are highly configurable to meet customer needs. The configuration includes, for instance, durable subscription or guaranteed delivery. The chosen configuration has a significant impact on the MOM's performance and, thus, has to be reflected by performance completions. Third, current performance completions capture the performance influences of a middleware platform offered by one specific vendor only. Theoretically, all middleware vendors have to provide separate performance completions for their middleware platforms. No suitable abstraction that is independent of vendor implementation is available yet. Fourth, the way the middleware platform or generated code is used heavily influences its performance. For example, the size of a message determines its delivery time. Verdickt et al. [5] already pointed out the importance of a service's usage (i.e., its input parameters) for performance. However, their approach cannot address such influences.

In this paper, we present an approach to define domain-specific languages that capture the performance-relevant configurations of middleware platforms and generated code. For this purpose, we use general information about the implementation that is available in design and collaboration patterns. Software architects can choose among different implementation features based on these languages on an abstract level. We developed and validated a method for identifying performance-relevant patterns and designing parametric performance completions on this basis. Parametric completions abstract from platform-specific details. They can be instantiated for different target environments using measurements of a test-driver. Test-drivers are designed especially for one parametric performance completion. They collect the necessary data to instantiate the completion for a specific platform. The abstract model in combination with measurement data makes performance completions applicable to various vendor implementations. 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. Furthermore, we explicitly couple the respective transformations to performance models and implementation adding the necessary details. The coupling of transformations makes the inclusion of implementation details deterministic.

We applied our approach to designing a parametric performance completion for Message-oriented Middleware (MOM) platforms. We chose MOM platforms, which comply to Sun's Java Message Service (JMS) specification [7]. Furthermore, we use the Palladio Component Model (PCM) [8,9] to realise the parametric performance completion and its corresponding coupled transformations. The PCM is an architecture description language supporting design-time performance evaluations of component-based software systems. To evaluate the prediction accuracy of the generated performance models, we conducted a real-word case study using the SPECjms2007 Benchmark [10,11]. The deviation of predicted and measured performance metrics (message delivery time and CPU utilisation) was below 10% to 15%.

The contributions of this paper are the theoretical background for parametric performance completions, a practical method for their design and application, and a real-world case study demonstrating their benefit:

Theoretical Background. Coupled transformations formalise the relation between the generated implementation and the generated performance model.

Practical Method. The proposed method guides the design and application of parametric performance completions. Parametric performance completions combine measurement data with knowledge on collaboration patterns. They abstract from the actual (vendor-specific) implementations and make completions independent of the hardware used.

Application. The application demonstrates the feasibility and benefit of parametric performance completions. For this purpose, we developed and implemented a parametric performance completion for Message-oriented Middleware. Software architects can include messaging in their architecture using a domain-specific configuration language based on messaging patterns.

The structure of this paper is as follows. Section 2 introduces the basic concepts of model-driven architectures and performance completions. Based on these concepts, Section 3 introduces coupled transformations which can be used to integrate low-level details into prediction models and implementations. Section 4 presents the concept and design process of parametric performance completions. In Section 5, we demonstrate the applicability of parametric performance completions. The section describes a real-world case study on message-based communication. In Section 6, we discuss assumptions and limitations of the proposed approach while, in Section 7, we compare it to related work. Section 8 concludes this paper and highlights future research directions.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

2. Background

Parametric performance completions make heavy use of model-driven technologies to include details about the implementation and execution environment into performance prediction models. In the following, we introduce the basic concepts of model-driven software development (Section 2.1) and describe the original idea of performance completions as introduced by Woodside et al. [3,12] (Section 2.2).

2.1. Model-driven software development

Software models abstract from details of the implementation of a software system and, thus, allow software architects to reason on larger and more complex systems. In model-driven software development processes like the OMG's Model-Driven Architecture (MDA) process, models also serve as input for transformations to generate the system's implementation.

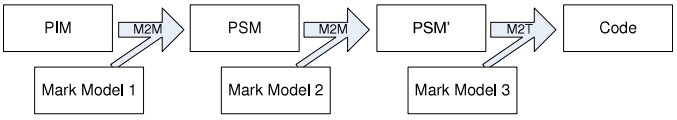


Fig. 1. MDA models and transformations.

Model-Driven Architecture (MDA) [13] leverages the role of models in software development. According to the MDA process, the first model to create is an abstract model of the business domain, the computation independent model (CIM). Based on this model, developers create a model of the system under development without using any details of the technical solution platform. This model is called a platform independent model (PIM) (cf. Fig. 1). Automatic model-2-model (M2M) transformations refine this model by adding implementation details of particular platforms. The term platform is a broad concept in this context. For example, it can define the type of the realisation (database application, workflow management, etc.) or a specific implementation of a technical concept like different industrial component models (.NET, CORBA, Java EE). Furthermore, a platform can refer to implementation dependent details like different types of configuration files depending on a particular middleware selection. A model which depends on such details is a platform specific model (PSM, cf. Fig. 1) for its platform.

Each model in such an MDA process has to be defined by a so-called meta-model. A meta-model defines the set of valid models both from a syntactical as well as from a semantical perspective. For example, the UML2 meta-model [14] defines the set of valid UML models. It defines the elements available in an UML model and their connections (syntax). Additionally, it contains the Object Constraint Language (OCL), which allows the definition of semantical constraints.

In Fig. 1, the refinement process is distributed among a number of transformations forming a transformation chain. Each transformation takes the output of the previous one and adds its own specific details. When refining high-level concepts of transformations into concepts on lower abstraction levels, different alternatives may be available. For example, if different applications communicate via messaging, different patterns for realising the message channels can be used, e.g., with or without guaranteed delivery. If developers want their transformations to be flexible, they can parameterise them allowing transformation users to decide on mapping alternatives themselves. The OMG's MDA standard allows transformation parameterisation by so called mark model instances.

Czarnecki and Eisenecker [15] introduced generator options in their book on Generative Programming which is a predecessor of today's MDA paradigm. They used so called feature diagrams to capture different variants in the possible output of code generators. Feature diagrams model all valid combinations of a set of features called (feature) configuration where a single feature stands for a certain option in the respective input domain. For example, using *Guaranteed Delivery* is an optional feature of the higher level feature *Sender* (cf. Fig. 7 in Section 5.2).

Using feature diagrams to parameterise model transformations bears the advantage of having a model for the possible transformation parameters which introduces the options in terms easily understandable by software architects. Using feature diagrams as mark models captures the variability in the mapping in a focused way as feature diagrams tend to be small. As such, they make mapping decisions explicit and allow the selection of appropriate completions for performance prediction.

2.2. Performance completions

When doing performance predictions in early development stages, the software model needs to be kept on a high level of abstraction. The middleware's complexity and the specific knowledge on the implementation that is required to create the necessary models would dramatically increase the modelling effort. By contrast, detailed information about a system is necessary to determine its performance accurately. Performance completions, as envisioned by Woodside and Wu [3,12], are one possibility to close this gap. Performance completions refine prediction models with components that contain the performance-relevant details of the underlying middleware platforms. To apply performance completions,

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

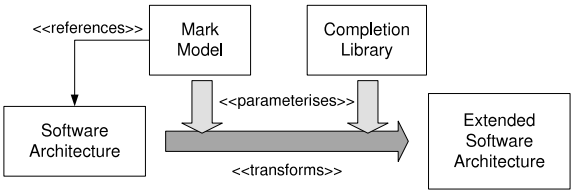


Fig. 2. Transformation integrating performance completions.

software architects have to extend their software model with annotations (or rules) whose refinements (such as additional components, execution environments, or communication patterns) are injected into the software model.

Fig. 2 shows how performance completions can be realised using the MDA concepts described in Section 2.1. Elements of a software architecture model, such as components or connectors, are annotated by elements of a *Mark Model* using, for example, feature diagrams. Mark models annotate elements in the architecture which shall be refined and provide the necessary configuration options. For example, if a connector can be replaced by message-passing the mark model can provide information about the type of the messaging channel, e.g., using guaranteed delivery. Model-to-model transformations take the necessary components from the *Completion Library*, adjust them to the configuration, and insert them in the software architecture prediction model. The result of the transformation is an architecture model whose annotated elements have been expanded to its detailed performance specifications.

For the sound application of performance completions, it is essential to understand the relation of the generated performance models and the generated source code. Coupled transformations described in the following section provide the necessary formal background for the relation of both.

3. Coupling transformations to implementation and performance models

Model-based or model-driven performance predictions with current methods [1] rely on the information available in the source model only, e.g., UML models annotated with the UML-SPT profile. However, the performance of a software system is a runtime property, i.e., a property of the deployed and executed implementation of the system. Hence, the implementation has to correspond to the model. However, if a team of developers uses the model as blueprint to implement the system manually, it often cannot be ensured that the code corresponds to the model. Model-driven software development can reduce this variability, as design decisions have to be modelled explicitly and source code is generated automatically. Model-based performance predictions should use this knowledge to achieve more accurate predictions.

Coupled transformations [16] formalise the relationship between generated code and performance completions. Mark models configure the code transformation as well as transformation to performance models. Considering the same information used for generating code, middleware configurations, and deployment options for software performance prediction is a powerful means to put performance completions to practice.

In the next Section, we give a motivating example demonstrating the central idea of coupling transformations to code and prediction models. The formalisation of coupled transformations follows in Section 3.2.

3.1. Motivating example

Technical design decisions involved in implementing abstract model concepts may lead to different implementations of the same concept by different developers. The following example presents such an open design decision of the model and additionally shows its influence on the system's performance.

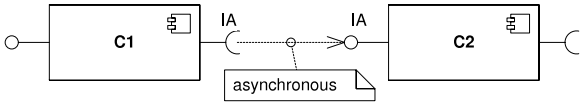
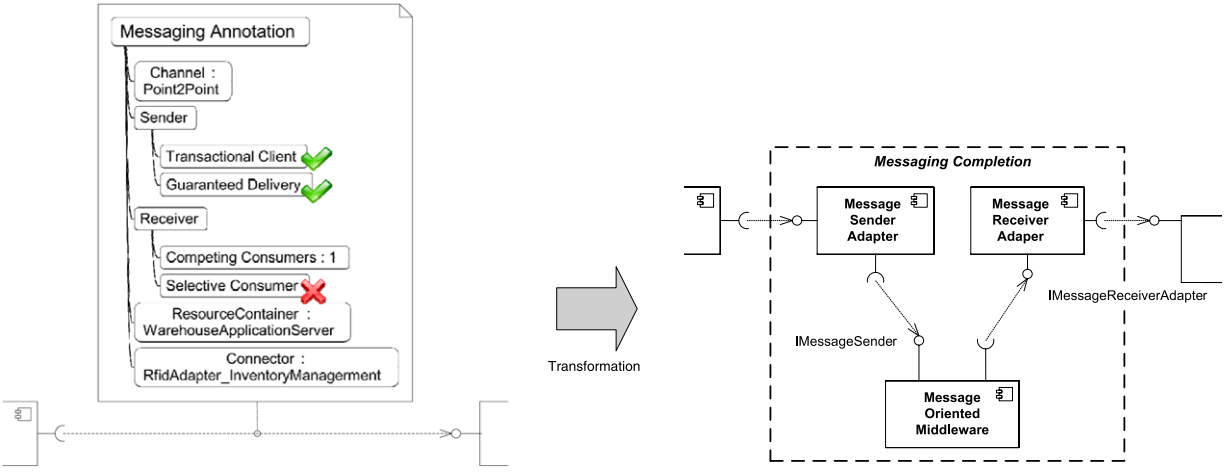


Fig. 3. Motivating example for coupled transformations.

Fig. 3 shows the structural view of a simple architectural model. Two components `C1` and `C2` communicate using a connector. The attached UML note resembles the mark model instance that marks the connection as asynchronous. Assume that two teams of developers implement the system. The first team uses a Message-oriented Middleware (MOM) and configures the channel with guaranteed delivery to realise the connection. The other team implements the asynchronous communication manually not considering any special features of MOM. The implementations of both teams are valid, as they are consistent with the given model (assuming no additional information or implementation rules existed). Despite being functional equivalent, the performance impact of both implementations is likely to be different. Model-driven performance predictions have to rely on correct implementation assumptions about the connector.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006



(a) Annotated connector. (b) Result of the transformation.

Fig. 4. Parameterised transformation.

A solution to the presented consistency problem between model and code is provided by a model-driven development process. Using deterministic transformations to generate an implementation from a model limits the degrees of freedom assumed for manual implementations, i.e., the result of a transformation solely depends on the input models. Transformations restrict the amount of possible outputs to the number of different input model instances as the mapping to implementations is fixed in the transformations and cannot be changed. If multiple mapping options are available, the transformation should be able to reflect these options. For our example, it must be able to generate a message channel with or without guaranteed delivery.

However, the introduction of different types of messaging in the model undermines the abstraction property of software architecture models [17], since unimportant details of the modelled entities should be omitted. Since the additional information should not be part of the source model, we use transformation parameters (mark models) to include the necessary information in the transformation. For example, Fig. 4(a) shows a connector annotated by a mark model for message-based communication. The coupled transformation takes the software architecture model and the mark model as input and deterministically generates the corresponding source code and/or performance models. The tight coupling of code and performance model generation allows us to establish a guaranteed relationship between the implementation and the prediction model.

3.2. Formalisation

In this section, we give a formal description of coupled transformations and clarify their theoretical background. The concept can be applied to many model-driven prediction approaches for various quality attributes. In the following, we first define model transformations and chains of transformations formally. Based on these definitions, we introduce the formal description of coupled transformations.

Models and meta-models. Let MM denote a meta-model expressed as instance of some meta-meta-model MMM, e.g., the PCM (Palladio Component Model, see Section 5.1) is an instance of the Meta Object Facility (MOF) [18], which is the meta-meta-model promoted by the OMG. Then the set of all valid model instances of the meta-model MM is defined as

$$inst(MM) = \{M | M \text{ is a valid instance of } MM\}. \quad (1)$$

An instance is valid only if it conforms to MM's abstract syntax and (static and dynamic) semantics where MMM defines the semantics of the term conformance. Using MOF as meta-meta-model, i.e., $MMM = MOF$, then the MOF specification's semi-formal definition of conformance applies [18, p.53 cont.]. For example, the following holds using the notation introduced above: $inst(PCM)$ is the set of all valid component-based software architectures expressible in the PCM. $PCM \in inst(MOF)$ expresses the fact that the MOF is the meta-model used to define the PCM. $MOF \in inst(MOF)$ formalises the fact that MOF is an instance of itself.

Transformations. In the following, we formalise model transformations. Let t be a computable function which maps an instance of a source meta-model MM_{src} and an instance of a mark meta-model MM_{mark} to an instance of a target meta-model MM_{dest}.

$$t : inst(MM_{src}) \times inst(MM_{mark}) \rightarrow inst(MM_{dest}). \quad (2)$$

The function t represents a parameterised transformation. For example, consider a transformation:

$$t_{\text{PCM}\times\text{EJBMARK}\rightarrow\text{EJB}} : inst(PCM) \times inst(EJBMARK) \rightarrow inst(EJB)$$

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

mapping instances of the PCM ($inst(PCM)$) to instances of a meta-model to define EJB based applications. The latter serve as basis for the generation of an EJB based implementation. The transformation takes EJB specific parameters, e.g., which kind of Java EE communication the connectors use, as instances of an EJB mark meta-model ($inst(EJBMARK)$). Another transformation mapping PCM instances to implementations using the Fractal [19] component model has the following definition

$$t_{\text{PCM}\times\text{FRACTMARK}\to\text{FRACT}} : inst(PCM) \times inst(FRACTMARK) \to inst(FRACT)$$

where *FRACT* is a meta-model to describe instances of the Fractal component model and *FRACTMARK* a mark-model used to define Fractal specific implementation details. The role of the mark models in the two examples given as parameter set specific to the destination meta-model.

We did not cover the case in which a transformation has no mark model, i.e., takes no parameters other than the input model instance. For this, let *EMPTY* denote the *emtpy* meta-model, for which $inst(EMPTY) = \{\epsilon\}$ holds. This is analogue to the empty word used in grammar definitions. A parameterless transformation t is then

$$t : inst(MM_{src}) \times inst(EMPTY) \to inst(MM_{dest}). \tag{3}$$

In this case, t takes the empty set as second parameter $t(m, \epsilon)$ with $m \in inst(MM_{src})$.

Chains of transformations. Next, we consider composed transformations, which represent an ordered *chain* of transformations as introduced in Section 2.1. Let $T = \{t_i | i \in [1 \ldots N-1]\}$ be an ordered set of transformations t_i which are executed sequentially with t_1 being the first transformation and t_{N-1} being the last one. Each transformation t_i maps instances of meta-model MM_i and instances of mark-model MM_{mark_i} to instances of meta-model MM_{i+1}:

$$t_i : inst(MM_i) \times inst(MM_{mark_i}) \to inst(MM_{i+1}). \tag{4}$$

In the following, we show that a chain of transformations is itself a transformation t_{comp} fitting the definition in Eq. (2), that transforms instances of MM_1 directly into instances of MM_N. For this purpose, we use a mark-model instance of a meta-model $MM_{mark_{comp}}$, where $MM_{mark_{comp}}$ is a meta-model which is derived by combining the meta-models $MM_{mark_1} \ldots MM_{mark_{N-1}}$. To define meta-model compositions, we introduce the $\otimes$ operator which stands for the Cartesian product applied to meta-models. Let MM_1 and MM_2 be two meta-models, then $MM_1 \otimes MM_2$ is the set of all possible ordered pairs whose first component is an instance of MM_1 and whose second component is an instance of MM_2. Using function $inst()$, we can define $\otimes$ as follows:

$$inst(MM_1 \otimes MM_2) = \{(m_1, m_2) \mid m_1 \in inst(MM_1) \text{ and } m_2 \in inst(MM_2)\}.$$

Now, we can define $MM_{mark_{comp}}$ based on the individual mark-models:

$$MM_{mark_{comp}} = MM_{mark_1} \otimes MM_{mark_2} \cdots \otimes MM_{mark_{N-1}}. \tag{5}$$

A valid instance of the combined meta-model is a sequence of model instances of its constituting meta-models. For example, in MOF such a combined model instance can be realised as a model extend which contains n (sub-)models. In our case, an element

$$\vec{ma}_{comp} = (ma_1, ma_2, \ldots, ma_{N-1}) \in inst(MM_{mark_{comp}})$$

characterises a full set of parameters or mark model instances of all transformations t_i contained in a transformation chain, i.e., ma_i is a valid parameter for transformation t_i. A transformation

$$t_{comp} : inst(MM_1) \times inst(MM_{mark_{comp}}) \to inst(MM_N)$$

is the composed transformation of a chain of transformations t_i if

$$t(m_1, \vec{ma}_{comp}) = t_{N-1}(t_{N-2}(\ldots t_1(m_1, ma_1)\ldots), ma_{N-2}), ma_{N-1} = m_N \tag{6}$$

where $m_1 \in inst(MM_1)$, $m_N \in inst(MM_N)$, and $\vec{ma}_{comp} \in inst(MM_{mark_{comp}})$. We write $m_i \xrightarrow{t_i(ma_i)} m_{i+1}$ as abbreviation. If $m_{i+1} = t_i(m_i, ma_i)$ holds, a chain of transformations $t_1 \ldots t_{N-1}$ can be written as follows:

$$m_1 \xrightarrow{t_1(ma_1)} m_2 \xrightarrow{t_2(ma_2)} \cdots \xrightarrow{t_{N-1}(ma_{N-1})} m_N \Leftrightarrow m_1 \xrightarrow{t_{comp}((ma_1, ma_2, \ldots, ma_{N-1}))} m_N.$$

The following extends the previous PCM to EJB example into a chained transformation by appending a second transformation. This transformation adds details specific for the Sun Application Server, i.e., special configuration setting only available in this server. If both transformations are executed in a chain, a transformation results which transforms PCM instances into EJB applications for the Sun Application Server. Let the additional transformation be:

$$t_{\text{EJB}\times\text{SUNAPPMARK}\to\text{EJBSUN}} : inst(EJB) \times inst(EJBSUNMARK) \to inst(EJBSUN)$$

where *SUNAPPMARK* denotes a mark model defining parameters specific the Sun's application server and *EJBSUN* denotes an extended EJB meta-model containing Sun Application Server specific settings. Then, the transformation chain is

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

$$m_{PCM} \xrightarrow{t_{\text{PCM}\times\text{EJBMARK}\rightarrow\text{EJB}}(ma_{EJBMARK})} m_{EJB}$$

$$m_{EJB} \xrightarrow{t_{\text{EJB}\times\text{SUNMARK}\rightarrow\text{EJBSUN}}(ma_{SUNAPPMARK})} m_{EJBSUN}$$

The equivalent composed transformation is then

$$m_{PCM} \xrightarrow{t_{comp}(ma_{EJBMARK}, ma_{SUNAPPMARK})} m_{EJBSUN}$$

The example shows how transformation chains can separate several aspects of a transformation into multiple transformation steps.

Coupled transformations have to deal with the fact that not all functional relevant parameters of a transformation influence performance. Furthermore, performance models do not have to reflect the functional aspects of a system but their impact on software performance. Therefore, the design of performance completions requires a systematic approach to identify the performance-relevant features and create accurate performance models even if limited information on the system under study is available. In the following section, we address the question of how to build performance completions that capture performance-relevant information.

4. Parametric performance completions

Performance completions can be used to include the influence of execution environments and generated parts of the implementation in software performance prediction. They focus on performance-relevant influences and abstract from functional details. To design performance completions, we need to identify those factors that actually influence software performance and can neglect those that play a minor role. In this section, we present the concept of parametric performance completions (Section 4.1), a systematic approach to the design of performance completions (Section 4.2), and their parameterisation for different execution environments (Section 4.3).

4.1. Concept

The high complexity and diversity of middleware platforms (and even more of generated code) make the design of individual performance completions for specific target platforms cumbersome and time-consuming. The high effort for the design of performance completions may even void their potential benefit. Therefore, parametric performance completions abstract from the platform-dependent influences. They extend and generalise our pattern-based completion presented in [20]. Parametric performance completions consist of completion model skeletons that reflect the middleware's general (performance-relevant) behaviour. The skeletons are structurally similar for different platforms but their resource demands vary. For performance prediction, the missing resource demands have to be determined separately for each middleware platform and hardware environment. This approach allows accurate predictions using performance completions independent of the middleware implementation used and hardware platform.

The design and application of parametric performance completions involves two roles. *Performance analysts* are responsible for the design of parametric performance completions. They develop the model skeletons and test-drivers that evaluate different target platforms. *Software architects* use parametric performance completions for performance prediction. They instantiate the parametric completions for specific platforms.

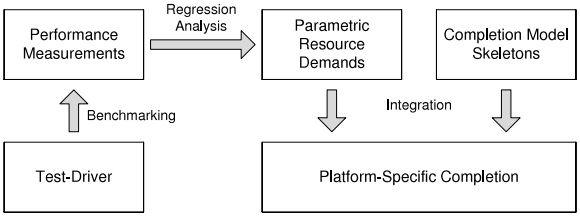


Fig. 5. Overview of the concept of parametric performance completions.

Fig. 5 sketches the idea of parametric performance completions. Their core concept is the separation of structural and quantitative information. While the structure is captured by *Completion Model Skeletons* which are the same for all target platforms, the quantitative information, i.e., the *Parametric Resource Demands*, is adjusted for each platform.

To capture the quantitative information for a specific platform, software architects execute *Test-Drivers* that take the necessary measurements. For performance prediction, the test-drivers are executed independently of the application under development. A running middleware platform is sufficient for this purpose. Thus, the test-driver decouples the performance model from the application development. Based on *Performance Measurements* of the test-drivers, software architects can determine realistic resource demands for complex middleware platforms, like today's Java EE application servers. The

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

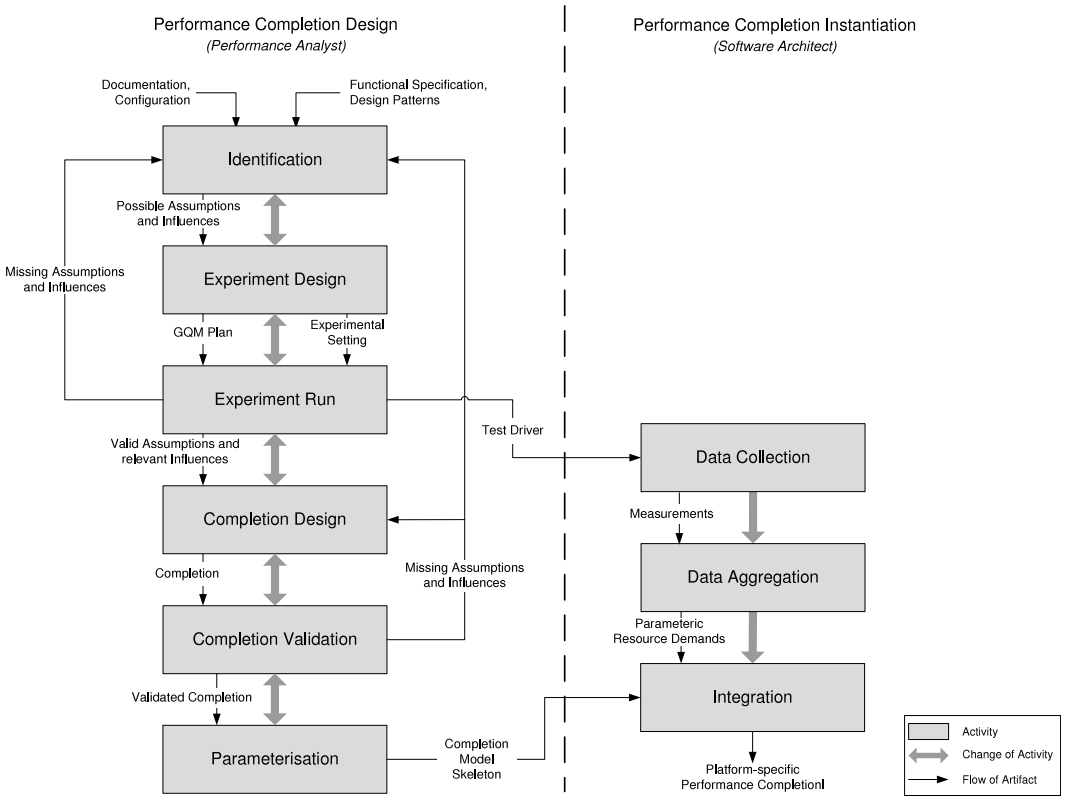


Fig. 6. Design and application of parametric performance completions.

combination of measurements and abstract performance models holds the middleware's complexity off the performance models while enabling accurate predictions.

Furthermore, test-drivers evaluate the quantitative effects of various feature configurations and their combination that can be specified in the mark model of the parametric performance completion. For the scope of this paper, we use feature diagrams to specify the available configurations and their possible combination for an execution environment or generated code. For example, software architects use a feature diagram to capture the various properties of messaging systems, e.g., the number of competing consumers at the receiver's side (see Fig. 7 in Section 5 for details).

In subsequent steps, software architects use the measurements to include the effect of different configurations in their prediction model. They analyse the measurement results and derive platform-specific *Parametric Resource Demands*. Regression analyses can be used to determine the dependencies of resource demands on input parameters (or more general on the usage profile). For example, software architects want to capture the effect of message sizes on resource demands for a specific Message-oriented Middleware. They perform regression analyses that yield the approximated functional dependency of resource demands on the message size. Ideally, the analyses are performed automatically.

The integration of the *Completion Model Skeletons* and *Parametric Resource Demands* yields the *Platform-specific Completion*. Therefore, the platform specific resource demands are attached to their corresponding actions of the model skeletons that structurally model the completion's behaviour. The combination of parametric resource demands and model skeletons yields a complete performance model for the specific target platform.

In the following, we describe the design process for parametric performance completions and their instantiation for specific platforms in more detail.

4.2. Design process

The key challenge of performance completion design is finding the right performance abstraction for the system under study. To identify the performance-relevant behaviour and factors, we employ a combination of goal-driven measurements and existing knowledge about the functional system behaviour. In general, only a little information about the behaviour of the system is available and, thus, we have to consider the middleware as a black box. To design performance completions, we have to use more general knowledge on the system, for example, implemented collaboration and design patterns.

The design process of performance completions comprises six activities shown on the left-hand side in Fig. 6. During *identification*, performance analysts use the existing knowledge of the system (e.g., design patterns) to identify a set of

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

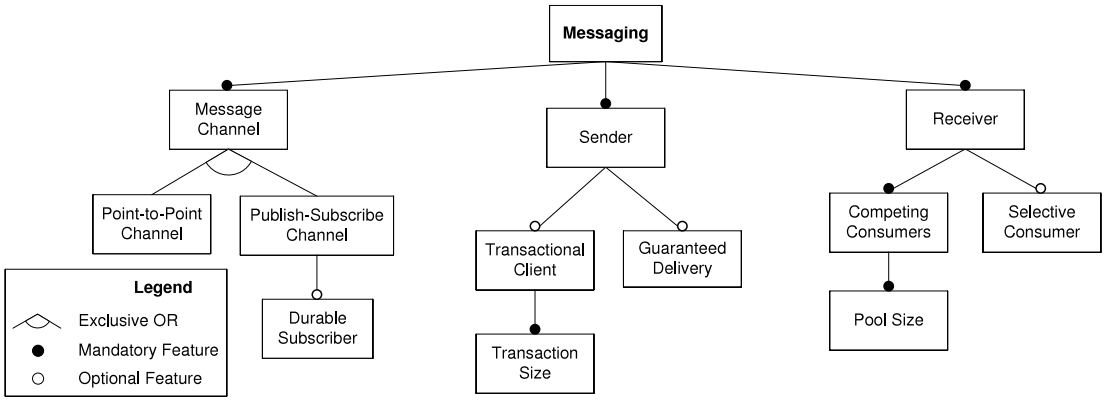


Fig. 7. Feature diagram of the relevant messaging patterns.

possible performance-relevant factors of the system under study. For the example of message-based communication from the previous section, the optional feature of guaranteed delivery can represent such a factor.

For the selected factors, performance analysts *design experiments* that evaluate their actual influence. For an efficient evaluation, we propose the use of goal-driven measurement methods such as the Goal/Question/Metric approach [21]. For our example, performance analysts may be interested in the influence of the amount of transferred data on the delivery time of a message. The result of this activity is a set of test-drivers that systematically evaluate the identified factors.

The *experiment run* yields the necessary measurements that give insight into the actual influences on software performance. Based on the results of the experiments, performance analysts can decide whether a feature can be included in the performance model or not. If they want to include it, they need to *design performance completions* that model the relevant behaviour. The behaviour of the completion for message-based communication might simply consist of some resource demands on the client and server side as well as utilization of the network connection.

In order to ensure that the model captures all relevant factors, performance analysts compare predictions and measurements in a *validation* activity. Based on the outcome of the validation, it might be necessary to execute further experiments to evaluate observed deviations of predictions and measurements. For the message-based communication, performance analysts may compare different predictions and measurements for the delivery time of a message with and without guaranteed delivery.

When the desired degree of accuracy is reached, performance analysts can parameterise the performance completions. Therefore, they remove the resource demands from the completion components and adjust the test-drivers (that are a result of the experiment design) to provide the information necessary to derive resource demands for any target platform. Ideally, they extend the test-driver so that it automatically derives the parametric resource demands and adds them to the model skeletons. For our example, the resource demands (specified as times) for the client and server side as well as the network are removed from the completion leaving its skeletons. Next, software architects can use the specific test-driver to instantiate the parametric performance completion for a specific platform and predict its performance influence.

4.3. Instantiation process for specific platforms

The design phase yields completion model skeletons that capture the structure of the completion and a test-driver that collects the necessary information of the target platform. The right-hand side of Fig. 6 illustrates how software architects can instantiate parametric performance completion for their target platform.

During *data collection*, the test-driver is executed on the target platform. It measures the performance metrics for all relevant parts and configurations of the middleware that are necessary to instantiate the performance skeletons for the target platform. For the example above, the test-driver measures (for instance) the time from sending a message until its `onMessage` method is executed at its destination, i.e., the delivery time.

So far, the measurements only provide raw data, e.g., a series of response time and throughput measurements. In order to use the data for performance prediction, further analyses are necessary. These analyses are performed in the *Data Aggregation* phase. The aggregation can subsume a broad rage of different analyses. For example, regression analyses may be necessary to determine the dependency of different parameters on observed performance metrics (e.g., [22]). For the messaging system, the delivery time of a message may depend on its size. Furthermore, the observed timing behaviour may not be sufficient for performance prediction. For more accurate performance models, queueing theory (e.g., the Service Demand Law) can be applied to estimate resource demands based on measured response times, throughput, and resource utilisation (cf. Section 5.3.2).

Once the resource demands have been determined, they can be integrated into the model skeletons. The result of the *Integration* activity are platform-specific performance completions. Software architects can use these completions to predict

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

the performance of the system under study for the specific platform. For example, software architects may create a specific instance of the messaging completion for their company network or some client's environment.

In the next section, we present a real-world case study that illustrates the design and application of parameterised performance completions.

5. Parametric performance completions for message-oriented middleware

In today's enterprise application, messaging represents a central mean for communication. Message buses allow the loose coupling and easy distribution of software systems. To consider the influence of messaging on software performance, we developed a parametric performance completion for message-based communication based on our previous work [20]. We chose the Palladio Component Model (PCM) [8,9] for this purpose. The PCM supports different analytical methods (e.g., simulation [8] and layered queueing networks [23]). It can also generate code skeletons from architectural specifications [24] and, thus, is well suited for realising coupled transformations. In the following, we briefly introduce the PCM (Section 5.1). Furthermore, we describe the performance-relevant configurations of Message-oriented Middleware (Section 5.2), specify the necessary components and their behaviour (Section 5.3), and present a real-world case study of the SPECjms2007 Benchmark [10] to validate the parametric performance completion (Section 5.4).

5.1. The palladio component model

The Palladio Component Model (PCM) [8,9] is an architecture description language supporting design-time performance evaluations of component-based software systems. In the following, we introduce the concepts necessary for the middleware completion.

Software components are the core entities of the PCM. Basic components contain an abstract behavioural specification called Resource Demanding-Service Effect Specification (RD-SEFF) for each provided service. RD-SEFFs describe how component services use resources and call required services using an annotated control flow graph. Basic components cannot be further subdivided. In contrast, composite components are assembled from other components introducing hierarchy into the component model. To connect components, a connector binds a required interface of one component to the provided interface of another component.

Following Szyperski's definition [25], a component is a unit of independent deployment with explicit dependencies only. As a consequence, component specifications in the PCM are parameterised for their later environment. The parameterisation of a component's performance specification covers influences of required services, different soft- and hardware environments, as well as different input parameters of provided services. Similar to UML activities, RD-SEFFs consist of three types of actions: Internal actions, external service calls, and control flow nodes.

Internal actions model resource demands and abstract from computations performed inside a component. For performance prediction, component developers need to specify demands of internal actions to resources, like CPUs or hard disks. Demands can depend on parameters passed to a service or return values of external service calls.

External service calls represent invocations by a component of the services of other components. For each external service call, component developers can specify performance-relevant information about the service's parameters. For example, the size of a collection passed to a service can significantly influence its execution time, while the actual values have only little effect. Modelling only the size of the collection keeps the specification understandable and the model analysable. Apart from input parameters, the PCM also deals with return values of external service calls. Note that external service calls are always synchronous in the PCM, i.e., the execution is blocked until a call returns. This is necessary to consider the effect of return values on performance. However, asynchronous calls are essential to model MOM. A combination of external service calls and fork actions (that allow parallel execution) can introduce asynchronous communication into the model.

Control flow elements allow component developers to specify branches, loops, and forks of the control flow.

Branches represent "exclusive or" splits of the control flow, where only one of the alternatives can be taken. In the PCM, the choice can either be probabilistic or determined by a guard. In the first case, each alternative has an associated probability giving the likelihood of its execution. In the latter case, boolean expressions on the service's input parameters guard each alternative. With a stochastic specification of the input parameters, the guards are evaluated to probabilities.

Loops model the repetitive execution of a part of the control flow. A probability mass function specifies the number of loop iterations. For example, a loop might execute 5 times with a probability of 0.7 and 10 times with a probability of 0.3. The number of loop iterations can depend on the service's input parameters.

Forks split the control flow into multiple concurrently executing threads. The control flow of each thread is modelled by a so-called forked behaviour. The main control flow only waits for forked behaviours that are marked as synchronised. Its execution continues as soon as all synchronised forked behaviours have finished their execution (Barrier pattern [26]). Fig. 13(a) shows a fork action with a single forked behaviour whose synchronised property is set to false. The fork action spawns a new thread and immediately continues the execution of the main control flow. This models an asynchronous service call in the PCM.

In the PCM, *parameter characterisations* [27,28] abstractly specify input and output parameters of component services with a focus on performance-relevant aspects. For example, the PCM allows to define the `VALUE`, `BYTESIZE`,

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

Table 1
Messaging patterns and features categorised according to their performance influence.

Messaging pattern	Performance influence			
	≈ 0	<0.1	≤ 1.0	>1.0
Point to point	✓			
Publish subscribe	(✓)			
Guaranteed delivery			✓	
Idempotent receiver	✓			
Selective consumer		✓		
Transactional client				✓
Durable subscriber			✓	
Competing consumers				✓
Message size				✓
Remote receiver				✓

NUMBER_OF_ELEMENTS, or TYPE of a parameter. The characterisations can be stochastic, e.g., the byte size of a data container can be specified by a probability mass function:

$$\texttt{data.BYTESIZE} = \texttt{IntPMF}[(1000; 0.8)(2000; 0.2)]$$

where IntPMF is a probability mass function over the domain of integers. The example specifies that data has a size of 1000 bytes with probability 0.8 and a size of 2000 with probability 0.2.

Stochastic expressions model data flow based on parameter characterisations. For example, the stochastic expression

$$\texttt{result.BYTESIZE} = \texttt{data.BYTESIZE} * 0.6$$

specifies that a compression algorithm reduces the size of data to 60%. Thus, the expression yields: IntPMF[(600;0.8)(1200;0.2)]. Stochastic expressions support arithmetic operations ($*$, $-$, $+$, $/$, ...) as well as logical operations for boolean expressions ($==$,$>$,$<$,AND,OR, ...) on random variables.

Finally, *resource containers* model the hardware environment in the PCM. They represent nodes, e.g., servers or client computers, on which components can be allocated. They provide a set of processing resources, such as CPUs and hard disks, that can be used by the hosted components. Processing resources can employ scheduling disciplines such as processor sharing or first-come-first-served. In the following, we present an evaluation of the influence of different design patterns for message-based communication on software performance.

5.2. Identified performance influences

Our first step in the design of a parametric performance completion for MOM is the evaluation and identification of its performance-relevant factors. Therefore, we conducted measurements for all message patterns on the JMS implementation of Sun's Java System Message Queue 3.6 (see [29] for details). Table 1 lists the resulting classification. We distinguish features without performance influence (mean delivery time not changed), a small influence (below 10%), a moderate influence (between 10% and 100%), and a large influence (more than 100%). All features of the last category depend on input parameters, e.g., message size, number of messages in a transaction, or the number of competing consumers.

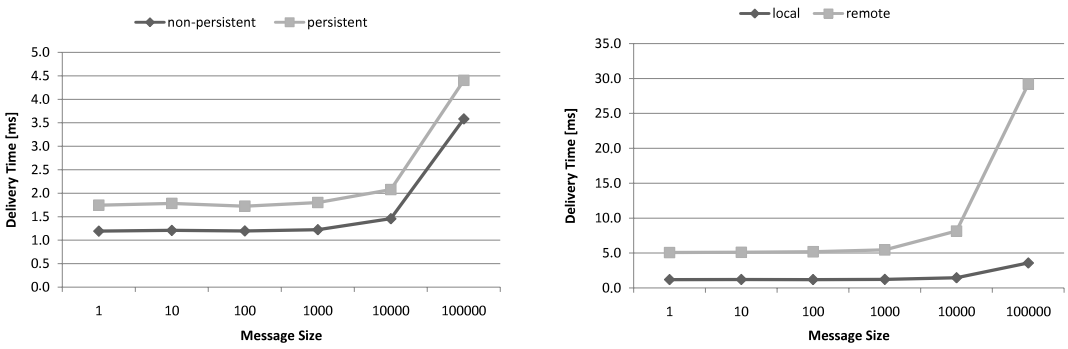
For each messaging pattern, we have measured the *delivery time* of a message (the time passed from sending a message until its onMessage method is executed at the receiver's side) in a series of experiments. The results of the benchmark form the basis for the pattern selection presented in Fig. 7. The feature diagram distinguishes patterns for message channels, receivers, and senders. In the following, we explain the patterns and their performance influences.

Message channels are logical connections between communicating components. They can be considered as queues. While *point-to-point* channels allow only a single receiver for messages, multiple receivers can subscribe to *publish–subscribe* channels. Optionally, a receiver can *durably subscribe* to the latter. In this case, the MOM keeps all published messages for a receiver disconnected from the channel until they can be delivered.

For a single receiver, the choice between publish–subscribe and point-to-point channels has no considerable effect on the delivery time. However, this distinction is necessary for modelling multiple receivers and, thus, is included in the model (see for example [30] for a detailed evaluation of the influence of multiple receivers). Furthermore, durable subscription leads to longer delivery times even if the receiver is never disconnected.

Senders add messages to a message channel. The sender of a message determines its size, transaction boundaries, and type of delivery. The *message size* depends on the data that needs to be transferred from the sender to the receiver. A message is a simple data structure containing a header and a body. However, message size refers only to the body of a message neglecting the influence of possible overhead of the message, such as its header. To *guarantee the delivery* of a message, the MOM stores messages persistently during their transfer. The implementation of a MOM determines how messages are stored, for example using a database or file system. Stored messages can survive system crashes and are delivered after restart, if possible. A *transactional client* sends one or multiple messages as a single transaction. The transaction boundaries are specified by the sender.

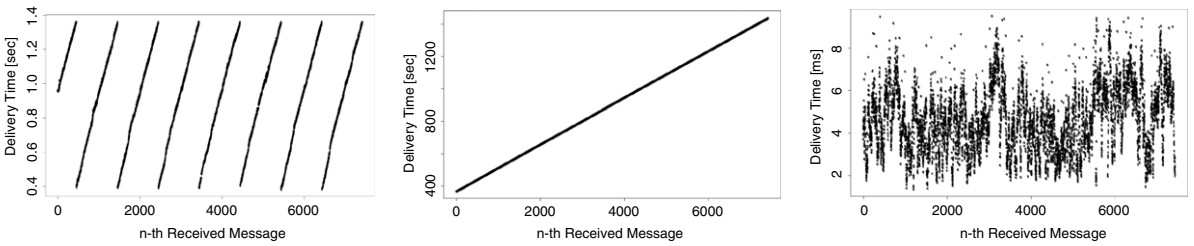
Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006



(a) Persistent vs. non-persistent message transfer.

(b) Local vs. remote message transfer.

Fig. 8. The influence of message size on the delivery time.



(a) Delivery time of messages in a transaction set with 1000 messages.

(b) Single consumer.

(c) Multiple (128) consumers.

Fig. 9. The effect of transactions and competing consumers on delivery time.

The size of a message significantly influences its delivery time. Fig. 8 illustrates this effect. With an increasing message size the delivery time of a message increases. While the slope of the curves is rather small for short messages, its impact grows for messages larger than 10 000 bytes. The influence of the message size strongly depends on the platform. For the system evaluated in Fig. 8, the influence is especially strong for the distributed setting.

For guaranteed delivery (Fig. 8(a)), the access to additional resources, e.g., the hard disk, leads to longer delivery times. The increase is approximately 25%. If the MOM or the message receiver is deployed on a remote machine, the necessary transfer over the network further delays the delivery of a message (Fig. 8(b)). As to be expected, the network's influence is more significant for larger messages.

For transactional clients, the delivery time of a message strongly depends on the number of messages in a transaction set and the message's position. The delivery time increases linearly with the message's position in the transaction set (see Fig. 9(a)). The MOM stores all messages until it receives the last message of a transaction set and then delivers the messages sequentially. Since messages arrive much faster than they are processed, the waiting time of the first message (0.4 s) is exceeded by the processing time of the successive messages. The sequential processing of messages leads to the observed linearly increasing delivery times.

Receivers remove messages from a message channel. They can employ multiple, *competing consumers* to process incoming messages concurrently. Consumers wait for incoming messages. When a message arrives, it is processed by the next waiting consumer. If no consumer is available, messages queue up until a consumer finishes processing its current message. Furthermore, message receivers can filter messages delivered via its subscribed channels. These *selective consumers* only accept messages, which match their filter criteria.

Competing consumers can have a large impact on performance. If too few consumers are available, congestion is likely leading to long delivery times. For example, if messages are received and processed sequentially by a single consumer, the consumer can easily become a bottleneck leading to congestion on receiver side as shown in Fig. 9(b). Message delivery times increase constantly up to 1400 s. When multiple consumers process the same load (Fig. 9(c)), the system can maintain the pace of message arrivals yielding acceptable message delivery times of less than 10 ms. Thus, multiple competing consumers can avoid congestion at the receiver side. The influence of selective consumers depends on the complexity of their filters [31]. For the simple filters considered in this evaluation, the influence on delivery times was marginal.

5.3. Completion design

In the following, we present how the PCM (cf. Section 5.1) can be used to design and apply a parametric performance completion for Message-oriented Middleware. The messaging completion takes into account the patterns evaluated in

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

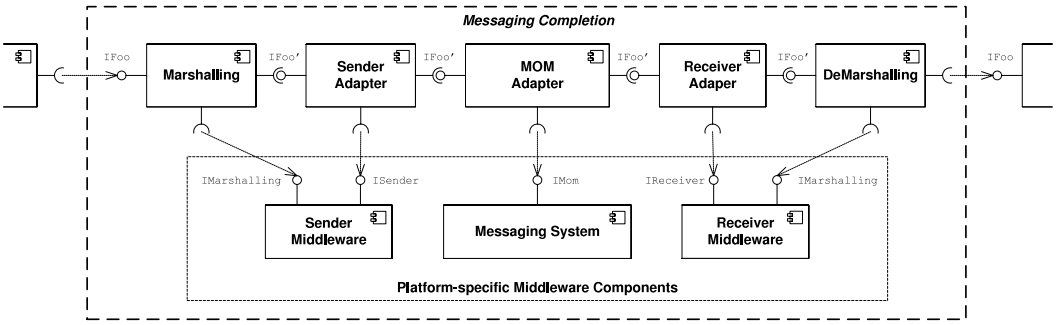


Fig. 10. Messaging completion components.

Section 5.2 together with the allocation of the MOM. The feature diagram in Fig. 7 allows software architects to easily choose among different design alternatives regarding the messaging service.

Coupled transformations generate a series of adapter components that models the performance-relevant behaviour of the Message-oriented Middleware. The adapters hide the message-based communication from the sender and receiver. Resource demands are generated by middleware components for specific platforms. Fig. 10 shows the resulting components.

5.3.1. Parameterisation

To capture the effect of different message sizes on software performance, we exploit the parametric resource demands of the PCM. Stochastic expressions [8,9] reflect the influence of different parameters on software performance. They support basic arithmetic operations on probability distributions and parameters (cf. Section 5.1).

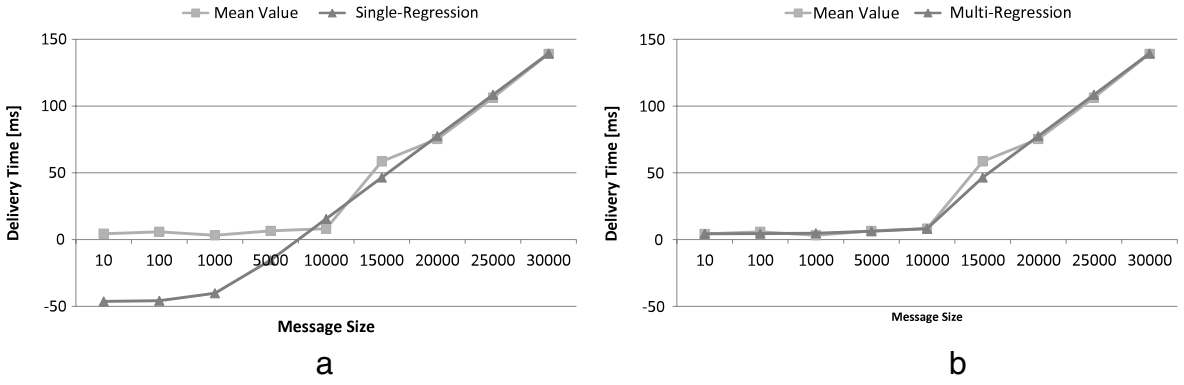


Fig. 11. Regression analysis for different message sizes.

We use multiple regressions to determine the stochastic expressions that approximate the influence of the message size. The regression can be applied on the delivery time of a message (as described in the following) or on estimated resource demands (as described in Section 5.3.2). Fig. 11 shows an example of how the message size can affect the average delivery time. Here, the sender, receiver, and the MOM are deployed on the same machine. A single regression analysis [32] over the measured times yields the linear function in Fig. 11(a). While the approximation is good for large messages, it deviates a multiple of the measurements for small ones. To achieve better prediction results, we used two separate regression functions: One for messages smaller than 1000 bytes and one for messages larger than 1000 bytes (cf. Fig. 11(b)). This reduces the estimation error to approximately 5%–30%. For example, the average delivery time of messages larger than 1000 bytes can be computed by a linear function with a slope of 0.02 and a y-intercept of -32.8. This yields the following stochastic expression:

$$0.02 * \texttt{message.BYTESIZE} - 32.8$$

In the prediction model, a branch condition selects the correct regression for a specific message size. For performance prediction, the stochastic expressions resulting from the regression analysis are integrated in the platform-specific messaging completion.

5.3.2. Approximation of resource demands

Using delivery times directly as input for performance prediction can be sufficient, if the system's load is moderate. For higher loads, we have to consider the actual resource demands in order to accurately predict contention effects. We use

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

the Service Demand Law [33] for this purpose. Based on this law, we can compute the demand D at a particular resource using its utilisation U and throughput X: $D = U/X$. In the following, we assume that a single resource limits the overall performance of the MOM. Furthermore, we require that the scheduling policy of the bottleneck resource (e.g., the CPU) can be approximated by processor sharing. Thus, when n messages are processed or delivered simultaneously, each message receives $1/n$th of the bottleneck resource. Based on these assumptions, we can approximate resource demands for message delivery and for message processing.

Let t_t be the total time from sending a message until its processing is finished, t_d the delivery time of a message, and t_p the processing time of a message, i.e., the time needed for executing the method invoked at the receiver's side, then we know that $t_t = t_d + t_p$. The values t_d, t_p, and t_t can be, for example, determined by measurements of the test driver. Moreover, let d_t, d_d, and d_p be the respective resource demands at the bottleneck resource with $d_t = d_d + d_p$. For performance predictions, we need to know d_d and d_p. These values cannot be derived directly using the Service Demand Law. Therefore, we propose the following approximation.

In a first step, we estimate the number of competing messages n based on the total time t_t. For this purpose, we have to consider an additional delay, δ, caused by waiting and processing times at other resources. So, t_t is given by:

$$t_t = n * d_t + \delta \tag{7}$$

If $\delta/d_t \to 0$ (the demand at the bottleneck resource is much larger than the delay at other resources), we can use Eq. (7) to approximate the average number of messages handled simultaneously:

$$n = t_t/d_t - \delta/d_t$$
$$n \approx t_t/d_t$$

n refers to the total number of messages in the system that are either processed or delivered. A further distinction (such as $n = n_d + n_p$) is not necessary, since the delay included in the processing time t_p is a result of all messages that are either delivered or processed. Furthermore, if only the bottleneck resource (e.g., the CPU) is used during message processing, we have that $t_p = n * d_p$. Then, we can approximate d_d by:

$$d_d = d_t - d_p \tag{8}$$
$$= d_t - t_p/n \tag{9}$$
$$\approx d_t - t_p/t_t/d_t \tag{10}$$
$$\approx d_t(1 - t_p/t_t) \tag{11}$$

d_t is the only remaining unknown variable in Eq. (11). Applying the Service Demand Law ($d_t = U/X$), where U is the utilisation of the bottleneck resource and X the measured throughput of the MOM, yields:

$$d_d \approx U/X(1 - t_p/t_t). \tag{12}$$

Eq. (12) enables us to estimate resource demands of message delivery (d_d) and of message processing ($d_p = d_t - d_d$). However, the approximation can only be applied if a single resource limits the overall performance, the resource's scheduling policy can be modelled by processor sharing, and the demands at all other resources are comparatively low. In our case study (Section 5.4), we successfully used this approach to estimate parametric resource demands of the message deliveries. In addition, we applied linear regression analyses on the estimated resource demands (cf. Section 5.3.1) to capture the influence of different message sizes.

5.3.3. Marshalling and demarshalling

Apart from the parametric dependency of resource demands and message sizes, we also have to determine the size of a message resulting from a specific service call. Therefore, we developed a marshalling completion for the PCM. To consider the influence of message sizes, marshalling is always the first, demarshalling the last processing step (cf. Fig. 10). All resource demands of the following steps depend on the size of the serialised message. The `Marshalling` component computes the message size depending on the serialisation method used (binary or XML) and the parameter specifications of a signature. Fig. 12 shows its RD-SEFF for an artificial signature `service(p1,.., pn)` with parameters `p1` to `pn`. To pass on the size to the following processing steps, the transformation extends the initial interface used for remote communication. Assuming `IFoo` is the considered interface, the transformation derives a new interface `IFoo'` whose signatures are equal to the ones of `IFoo` except for an additional parameter `stream`, e.g., `service(p1,.., pn)` becomes `service(p1,.., pn, stream)`. Characterisation `stream.BYTESIZE` contains the probabilistic distribution of the message size.

To determine the size of the message, the `Marshalling` component calls the `Sender Middleware` on its interface `IMarshalling`. Since the PCM does not support overloaded parameters, the marshalling service takes all basic data types as input: `marshall(Strategy s, int[] ints, double[] doubles, String[] strings...)`. In the transformation, OCL function `number(sig:Signature, direction:ParameterModifier, t:PrimitiveDataType)` determines the occurrences of the `PrimitiveDataType t` in Signature `sig` and sets the corresponding `NUMBER_OF_ELEMENTS` (`NoE`) characterisation to the resulting values. Given this information the `Sender Middleware` computes the size of the resulting serialisation. See [34, pp. 208–217] for details of the marshalling and demarshalling.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

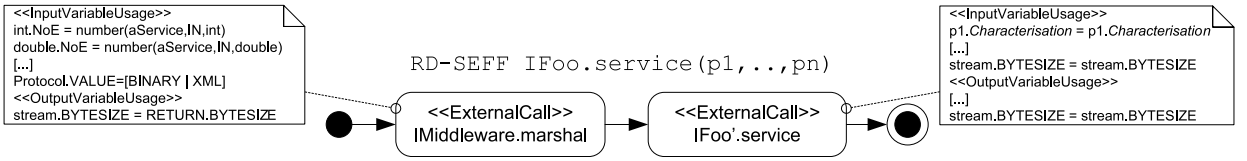


Fig. 12. Generated RD-SEFFs of the marshalling example.

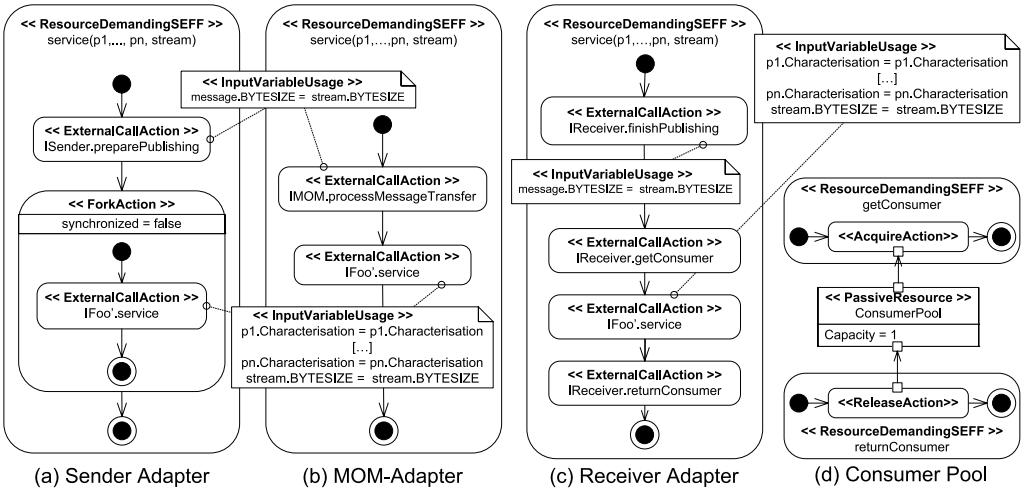


Fig. 13. Behavioural specifications for the sender, MOM and the receiver adapter.

Following the compositional approach presented in our previous work [16], the messaging completion can further be combined with any completion introduced there, e.g., encryption and authentication. This approach is possible since we are mainly interested in timing behaviour and not the actual data that is passed. Thus, we can consider (and model) message passing as asynchronous method calls.

5.3.4. Adapter components

Messages are passed from the sender to the receiver using the `SenderAdapter`, `MOM-Adapter`, and `Receiver Adapter` components. The adapters model the MOM's behaviour according to the configuration used. Furthermore, they generate the load caused by the sender, the MOM, and the receiver using the platform-specific middleware components. All adapters implement the interface generated by the marshalling components (`IFoo'` in Fig. 10). Thus, they can use the message size for performance evaluation. Furthermore, the adapters do not load the resources directly but call methods of the (platform-specific) middleware components that load the resource accordingly. In the following, we describe the behaviour of all components continuing the example in Fig. 10.

The `SenderAdapter` asynchronously forwards messages to the `MOM-Adapter` and, thus, allows the sender to continue its execution directly after the message has been sent. Fig. 13 shows its behaviour modelled as an RD-SEFF. Before the message is sent, the sender adapter calls method `preparePublishing` on the `SenderMiddleware`. The method's `InputVariableUsage` sets the size of the message. The middleware can use this value to determine the resource demand necessary to prepare the message. The size of a message can be a probabilistic distribution over different message sizes. When the preparation is finished, a `ForkAction` starts a new thread which forwards the service call to the `MOM-Adapter`. As its property `synchronize` is set to `false`, the execution of `SenderAdapter` continues immediately and does not wait for the behaviour of the `ForkAction` to finish. Thus, the sender adapter decouples the delivery of a message from the sender's process. Analogously, the `MOM-Adapter` loads the MOM's resources and passes the message to the `ReceiverAdapter` component (cf. Fig. 13(b)).

The `ReceiverAdapter` has to consider the influence of (competing) consumers that process incoming messages. In Fig. 13, their influence is modelled by the passive resource `CompetingConsumers` and corresponding `Acquire-` and `ReleaseActions`. The receiver adapter first pre-processes the incoming message. The overhead of this step is modelled by an `ExternalCallAction` to service `finishPublishing` of the receiver's middleware. Like for the sender and MOM adapter, the resource demand can depend on `message.BYTESIZE` passed to the service. When the pre-processing is finished, the receiver adapter tries to acquire a consumer for the message. Therefore, it calls service `getConsumer` on interface `IReceiver`. Its RD-SEFF is shown in Fig. 13(d). The passive resource `ConsumerPool` contains the maximum number of available consumers specified in the mark model. The consumers limit the number of concurrently processed messages. Method `getConsumer` executes a single `AcquireAction` on `ConsumerPool`. The `AcquireAction` blocks

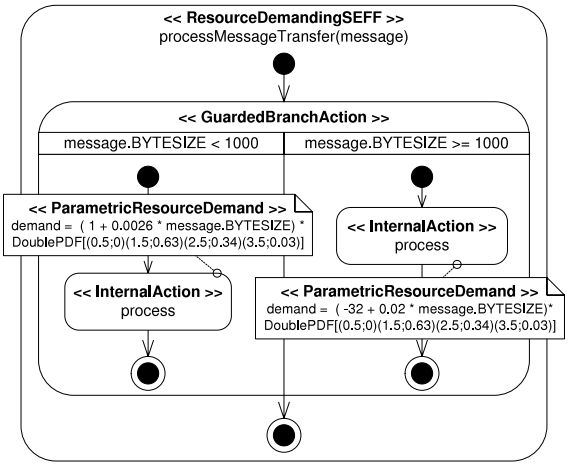


Fig. 14. Behavioural specification for the Message-oriented Middleware.

until a consumer becomes available. Then the receiver adapter forwards the message to the `DeMarshalling` component calling `service` on interface `IFoo'`. Once `service` is processed, the receiver adapter returns its consumer to the pool (calling `returnConsumer` on the receiver's middleware) and its execution is terminated.

5.3.5. Middleware components

While the adapters model the behaviour of the messaging system according to its configuration, the middleware components generate the platform-specific resource demands. If parametric resource demands with multiple regressions are used to approximate resource demands, their effect has to be reflected in the performance model. In the PCM, we can use stochastic expressions and guarded branch transitions for this purpose (cf. Section 5.1). Fig. 14 shows the RD-SEFF of method `processMessageTransfer` of the `Messaging System`. The RD-SEFF comprises a single `GuardedBranchTransition` that executes one of its branching behaviours depending on the message size. Whenever the `BYTESIZE` of a message is smaller than 1000 bytes the left branch is taken, otherwise the right one. Both branching behaviours differ in the `ParametricResourceDemands` of the internal action `process`. In both cases, the resource demands are stochastic expressions, whose resource demand increases with the message size (cf. Section 5.2). Currently, only the resource demand of a single, limiting resource is considered, which does not reflect the actual resource usage in distributed scenarios (discussion in Section 6).

5.4. Completion validation: A real-world case study

In this section, we present a case study that evaluates the prediction quality of the messaging completion described in Section 5.2. A comparison of predictions based on architectural specifications with measurements of an implementation gives an impression on the prediction accuracy of the messaging completion. The case study is based on the SPECjms2007 Benchmark [10,35,11] and focuses on the influence of the MOM on performance. Since the messaging completion should support early design decisions, the case study evaluates three design alternatives for one of the benchmark's interactions. The case study answers the question: Are the predictions using our messaging completion good enough to support the decision for the design alternative (i.e., configuration of the MOM) with the actual best performance given a specific configuration?

The SPECjms2007 Benchmark [10,35,11] provides suitable scenarios for the case study. It is a standard industry benchmark for performance analyses of JMS developed by SPEC's OSG-Java subcommitee (including IBM, TU Darmstadt, Sun, Sybase, BEA, Apache, Oracle, and JBoss). SPECjms2007 reflects the way messaging services are used in real-live systems including the communication style, the types of messages, and the transaction mix. Furthermore, it is focused on the influence of the MOM's implementation and configuration. Thus, the benchmark minimises the impact of other components and services that are typically used in the chosen application scenario. For example, the database used to store business data and manage the application state could be easily become the limiting factor of the benchmark and, thus, is not represented in the benchmark. This design allows us to focus our evaluation on the influences of the MOM without possible disturbances of other infrastructure components.

The SPECjms2007 Benchmark resembles a typical scenario of the supply chain management domain. It models a set of supply chain interactions between a supermarket company, its stores, its distribution centres, and its suppliers. In this case study, we focus on the inventory management of a single supermarket. *Inventory management* is necessary when goods leave the warehouse of a supermarket, to refill a shelf. RFID readers register goods leaving the warehouse and notify the

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

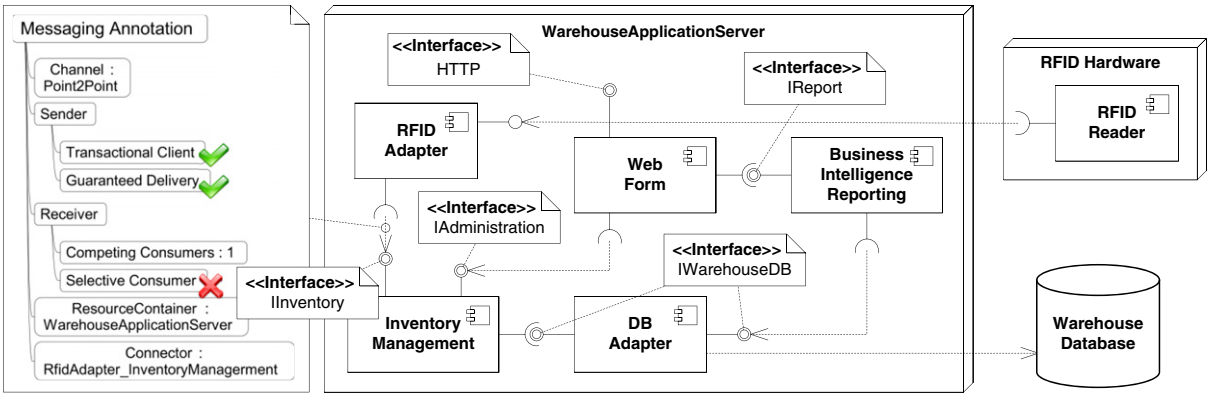


Fig. 15. Architecture of the warehouse application.

Table 2
Design alternatives.

	Alternative	Arrival Rate	Data Size	Configuration
1.	Persistent	100	Full	Persistent, Transacted
2.	Non-Persistent	100	Full	
3.	Small	100	Identifier	Persistent, Transacted

Full := Int PMF [(10;0.01) (100;0.04) (500;0.95)]
Identifier:= Int PMF [(10;0.95) (100;0.04) (500;0.01)]

local warehouse application, which updates its inventory. In the following, we describe the architecture of the warehouse application and propose three design alternatives.

Architecture of the warehouse application. Fig. 15 shows the static architecture of the warehouse application. A hardware `RFID Reader` is directly connected to the `Warehouse Application Server`. An `RFID Adapter` component manages the connection to the RFID reader. It converts and forwards the read data to the `Inventory Management`. A `Messaging Annotation` configures the connector between the `Inventory Management` and the `RFID Adapter` as persistent, transactional messaging channel. The message service allows `RFID Adapter` to quickly accept new requests from the `RFID Reader` as it will not block its execution. Persistency ensures that no inventory update is lost in case of failures. When notified, the `Inventory Management` updates the inventory data using the `DB Adapter` component.

Usually, many goods leave the warehouse at once, e.g., an employee brings a lorry with goods into the supermarket to refill the shelves. In this case, the RFID reader sends many messages in a short time period. Experts estimate the number of messages up to 100 in a second. The software architect now wants to know, if such a high load can be handled by the Message-oriented-Middleware. It also needs to ensure that the warehouse application itself is not affected.

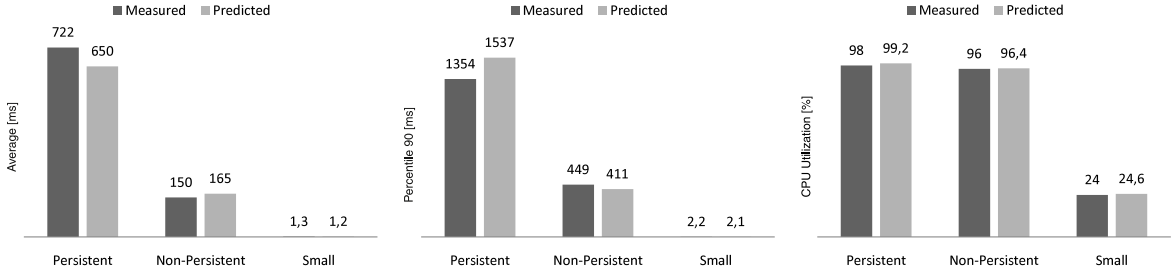
Design alternatives. The software architect considers three design alternatives of the warehouse application (Table 2). The original architecture (alternative 1, Persistent) sends the complete data, i.e., `data.BYTESIZE = Full` from the `RFID Reader` to the `Inventory Management`. Alternative 2 (Non-Persistent) uses a reconfigured message service, since persistency and transactionality might produce too much overhead. However, turning both off carries the risk of losing messages in case of failures, but might solve possible performance problems. Alternative 3 (Small) reduces the data size. Instead of transmitting all data kept on an RFID chip to the inventory management, the data could be limited to a single product identifier. This reduces the data size, but also requires changes of the `Inventory Management` component. Thus, this alternative should only be considered if really necessary.

Furthermore, the software architect defines performance requirements for the warehouse application. The RFID reader should not affect the rest of the application, so it should not utilise the system more than 50%, which enables the other components to keep working properly. In addition, the system should be able to handle 100 RFID reads per second, which is the expected maximum number of goods taken out of the warehouse at once. Finally, the delivery time of a message shall not exceed 1 s in 90% of all cases.

Results. We used the PCM's simulation environment SimuCom [8] to predict the performance for each design alternative. Basically, SimuCom interprets PCM instances as a queuing network model with multiple G/G/1 queues. To instantiate the parametric performance completions, we executed their test driver in the target environment. We derived the parametric resource demands for the CPU following the approximation method described in Section 5.3.2. Each simulation run lasted 5 min and simulated the delivery of over one million messages. A warm-up period of the first 2500 measurements was not included in the prediction results. The measurements were conducted with the SPECjms2007 Benchmark version 1.0. The benchmark was deployed on a single machine, to focus on the effects of message sizes and the message service's configuration. Sun's Java System Message Queue 3.6 provided the necessary infrastructure for the measurements. During the

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

measurement period, the benchmark executed only the inventory movement interaction. The upper 5% of measured values were removed, to exclude disturbances from the results. All other interactions were disabled and, thus, not considered in the case study. A warm-up period of 10 min preceded the measurement period of 30 min.



(a) Average of the delivery time. (b) 90% percentile of the delivery time. (c) CPU utilisation.

Fig. 16. Predictions and measurements of the three design alternatives.

Fig. 16 summarises the predictions and measurements for the three design alternatives. It shows the average and 90% percentile of the delivery time as well as the CPU's utilisation. Measured values are printed in dark grey, predicted values in light grey. The prediction error for the average delivery time (Fig. 16(a)) as well as the 90% percentile (Fig. 16(b)) is below 15% in all cases. The messaging completion predicts the CPU utilisation (Fig. 16(c)) with an error below 3% for all design alternatives. In the considered scenario, the usage of persistent message transfer had a major influence on the delivery time of a message. While the measured and predicted average delivery times for alternative 2 (Non-Persistent) are 150 ms and 165 ms, respectively, they are 722 ms and 650 ms for alternative 1 (Persistent). The 90% percentile of the latter exceeds the upper bound of 1 s. For the measurements it is 1354 ms, for the prediction 1537 ms.

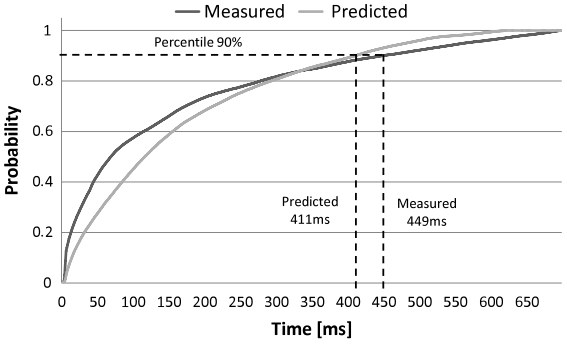


Fig. 17. Delivery time of alternative 2 (cdf).

To allow a visual comparison, Fig. 17 shows the cumulative distribution function (cdf) of the predicted and measured delivery times for design alternative 2 (Non-Persistent). The measured time is printed in dark grey and the predicted time in light grey. Both functions match to a large extent. The model predicted that 90% of all messages are delivered in less than 411 ms. This is confirmed by the measurements, where 90% of all messages are delivered in less than 449 ms. In this case, the prediction error is 8.5%. However, the predicted and measured CPU utilisation (Fig. 16(c)) of about 96% for alternative 2 exceeds the required maximal utilisation of 50%.

In the considered scenario, alternative 3 (Small) shows the best performance. Its measured and predicted delivery times are much smaller than for the other alternatives. For example, 90% of all messages are delivered in less than 2.2 ms (measured) and 2.1 ms (predicted). The measured and predicted CPU utilisation is with 24% and 24.6% below the required upper bound of 50%. Therefore, alternative 3 is the best choice for the software architect with respect to performance. Coming back to the question posed in the beginning of this section, the messaging completion can correctly rank different design alternatives concerning message services. It can predict the delivery time of messages with an error of less than 15% and the resource utilisation with an error less than 3%.

In the following section, we discuss the assumptions and limitations of coupled transformations, parametric performance completions, and their application to Message-oriented Middleware.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

6. Discussion

The case study based on the SPECjms2007 benchmark presented in the previous section demonstrates the applicability of parametric performance completions as well as the good prediction results of the messaging completion. However, despite the good results, there are several assumptions and limitations necessary for coupled transformations and parametric performance completions.

Manual interactions not considered. In cases where the source code transformation t_c is incomplete, i.e., results in code skeletons, the final code completion is done manually by developers. However, as argued in Section 3.1, this is a non-formal process, whose outcome usually highly depends on the developer. It is even very likely that the same code skeletons will be completed differently by the same developer, if it is done repeatedly with some time in between. For these reasons, manual transformations are disregarded in t_q. Current modelling approaches tend to be incomplete, hence, they contain implementation parts which cannot be captured by Coupled Transformations. However, applying Coupled Transformations to those parts of the generated code describable in a mark model gives improved accuracy. This is important as the number of fully generated systems increases, e.g., in the automotive domain.

Feature models as mark models. The transformation of the messaging completion uses feature models as mark models. This does not limit applicability of the method for other common types of mark models, such as stereotypes and tagged values, since all of them add parameters to the transformation. However, the different types of mark models offer different degrees of freedom on the parameterisation.

(Slightly) different source models for code-generation and performance-evaluation. In cases where the initial input model for a coupled transformation is slightly different for the code generation and for the performance evaluation, the approach may still be applicable. This is true if the input model to the performance model generation is a refinement of the code generation model. For example, using an UML model for code generation while using the same UML model *including* performance annotations for the performance model generation is fine.

Measurement-based models. The resource demands of the completion's internal actions are based on measurements. To predict the performance of a middleware system on different hardware platforms, it is necessary to re-execute the benchmark application for each platform in order to determine its resource demands. If the target platform is available this is not a problem. However, if the performance of an application needs to be evaluated during early development stages the execution environment might not be available.

Relying on measurements of time consumption leads to further challenges. The middleware might access different resources during the measured period. For example, a persistent message channel will access the hard drive. Measuring the whole period makes it challenging to assign the correct load to single resources. In our case study, we used the Service Demand Law to determine the resource demand at a single resource. This strategy was appropriate for our case study, since the resource that we considered was the one that limited the overall performance. In general, focussing on a single resource can lead to large prediction errors. Especially in a distributed setting, where different resources are accessed, demands and contention effect at all resources have to be considered.

Not only is the assignment of load to different resources challenging, also allocating the load to the involved components is difficult. Resource demands of single components cannot be measured directly. The computation of the actual demand of the sender, receiver, and middleware components is challenging. In Section 5.3.2, we demonstrated how individual resource demands can be approximated. However, it remains unclear to what extent our approach can be extended to estimate resource demands in a distributed setting.

Case study for message-based communication. The case study in Section 5.4 demonstrates the prediction accuracy of the messaging completion. The different configurations of alternative 1 and 2 significantly influence performance. Furthermore, the delivery time of a message strongly depends on its size. Especially in highly loaded systems, different message sizes can change the delivery time by several orders of magnitude. This makes the MOM's configuration as well as message sizes important factors for performance of systems using message-based communication.

The case study also demonstrated that predictions and measurements can deviate by up to 15%. This is mainly caused by the abstraction of the model compared to a real system. In the model, we focus on the largest resource demand in the system. Furthermore, the model does not represent the actual arrival rates of messages in the benchmark. The benchmark tries to achieve the specified rate of messages. However, if the system is overloaded, the benchmark reduces the pace, since the workload driver does not get enough processing time. The approximation of the resource demands by linear regression introduces another abstraction to the model. Therefore, the uncontended resource demands derived from a linear function can already deviate from the demands in a real system.

Horizontal and vertical scaling. In the description of the SPECjms2007 Benchmark, Kounev and Sachs [11] distinguish horizontal and vertical scaling. For horizontal scaling, the number of receivers for a message is increased, while for vertical scaling the number of messages in the system is increased. As demonstrated in the case study presented in Section 5.4, messaging completions can successfully predict the influence of additional messages in the system. However, the influence of additional message receivers can only be predicted with limited accuracy.

In the next section, we discuss work related to coupled transformations, parametric performance completions, and the messaging completion.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

7. Related work

The work related to the results we presented in this paper can be classified into two areas. The first area covers approaches which explicitly utilise model-driven software development for software performance predictions. The other area contains approaches evaluating the performance impact of (message-oriented) middleware.

Woodside and Wu [3,12] envisioned the concept of completions in order to supply additional information not needed for functional specification but required for performance prediction. Our work draws upon this idea and uses transformations generating the application to derive these rules and encodes them in a coupled transformation. Furthermore, we define a practical method that allows designing and parameterising performance completions for various target environments.

In software performance engineering, several approaches use model transformations to derive prediction models (e.g., [36–38,8]). Cortellessa et al. surveyed three performance meta-models in [39] leading to a conceptual MDA framework of different model transformations for the prediction of different extra-function properties [6,40].

Verdickt et al. [5] developed a framework to automatically include the impact of CORBA middleware on the performance of distributed systems. Transformations map high-level middleware-independent UML models to UML models with middleware-specific information. Their work focuses on the influence of Remote Procedure Calls (RPCs) as implemented in CORBA, Java RMI, and SOAP. Neither Verdickt nor Cortelessa considered the influence of service parameters on performance. For example, the prediction model for marshalling and demarshalling of service calls in [5] neglects the influence of the service's parameters. Verdickt et al. considered it an important factor, which their approach cannot address.

Coupled transformations and parametric performance completions fit into this context. Coupled transformations investigate the relationship between the code and prediction transformations. Additionally, parametric performance completions represent a practical approach to realise performance completions in complex environments.

Gorton and Liu [41,42] as well as Denaro et al. [43] studied the influence of middleware on software performance. Both considered middleware as the determining factor for performance in distributed systems and, thus, focused on its modelling and evaluation. Gorton and Liu [41,42] proposed a measurement-based approach in combination with mathematical models to predict the performance of J2EE applications. Measurements provide the necessary data to compute the input values of a queueing network model. The computation reflects the behaviour of the application under concern. The queueing network is solved to derive performance metrics such as response time and throughput for the application.

Denaro et al. [43] completely focused on measurements and did not rely on predictions. They assumed that the infrastructure of a software system is available during early development stages. Thus, application specific test cases based on architecture designs can provide estimates of the performance of a software system. Both approaches strongly simplify the behaviour of an application neglecting its influences on software performance. Furthermore, they require a fully available and measurable infrastructure, which can limit their applicability.

Many middleware platforms implement design and architectural patterns for distributed systems. For example, the MOM standard Java Message Service (JMS) [7,44] realises messaging patterns such as publish–subscribe or competing consumers (as described in Section 5.2). The inclusion of architectural patterns into performance prediction models was studied by Petriu [37,45] and Gomaa [46]. Petriu et al. [37,45] modelled the pipe-and-filter and client–server architectural patterns with UML collaborations. The UML models are transformed into Layered Queueing Networks using graph transformations as well as XSLT transformations. Gomaa and Menasce [46] developed performance models for component interconnections in client/server systems based on typical interconnection patterns. These approaches build the basis for the later development of performance completions.

Recently, Sachs et al. [30] presented a detailed evaluation of a message-oriented middleware (BEA Weblogic server) using the SPECjms2007 Benchmark. They evaluated different configurations (and combinations of configurations) of senders, receivers, and message-channels on performance. For the BEA Weblogic server, they observe similar performance influences (not in terms of quantity but quality) for message size, competing consumers, and guaranteed delivery as we found for Sun's Java System Message Queue 3.6. The conformance of the results suggests a good portability of the messaging completion to other platforms. For our work, the evaluation of vertical scaling (i.e., publish–subscribe channels) is of special interest. The findings and results of Sachs et al. can help us to further improve message completion.

8. Concluding remarks

In this paper, we (i) formalised the dependency of generated parts of the implementation (including middleware configurations) and performance models in coupled transformations, (ii) presented an approach to design and apply performance completions parametrising platform-specific influences, and (iii) applied our approach to capture the influence of Message-oriented Middleware on software performance.

Coupled transformations formalise the relation between generated code (including middleware configurations and deployment options) on one hand and performance models on the other hand. They limit the design space to a restricted set of features specified in the transformation's mark model. Chains of transformations realise these options deterministically. Using the knowledge about the deterministically generated code, coupled transformations can generate performance models based on the same restricted set of features. The determinism as well as the knowledge on the generated code allows performance analysts to design performance completions that accurately reflect its performance influence.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

To put performance completions into practice, we introduced a general process to design and apply performance completions in a parametric way. Performance analysts define completions based on abstract communication and design patterns and, thus, parameterise over the platform and vendor-specific properties of middleware platforms. In our design process, a specifically developed test-driver measures the performance of the target middleware. The results allow software architects to instantiate the parametric performance completion for their target platform. The combination of model skeletons with resource demands derived from measurements represents a powerful tool to predict the influence of complex middleware on software performance.

To demonstrate the applicability of parametric performance completions, we developed a completion for Message-oriented Middleware based on the Palladio Component Model. For the design of the completion, we modelled the MOM's behaviour using well-known messaging patterns. In a real-world case study, we applied the messaging completion to predict the performance of an interaction of the SPECjms2007 benchmark. The observed deviation of measurements and predictions was below 10% to 15%.

The approach of parametric performance completions in combination with coupled transformations helps software architects and performance analysts to systematically design and apply performance completions. The design process guides performance analysts to identify, model, and validate the performance-relevant factors of middleware platforms. Software architects can instantiate the resulting parametric performance completion (i.e., the model skeletons and test-drivers) for their target platform. Parametric performance completions reduce the necessary modelling effort (for software architects and performance analysts) as well as the complexity of the software architecture models. The combination of the approach with coupled transformations includes the necessary information about low-level details and allows more accurate performance predictions.

For the future, we plan to apply the approach of parametric performance completions to capture and model the performance influence of legacy applications. Even though the source code might be available in such cases its complexity hinders a detailed design of the application. The combination of measurements and modelling proposed in this paper can help us to identify and model the performance-relevant aspects of legacy applications.

Furthermore, messaging as an essential part of today's enterprise applications needs to be fully integrated into the PCM to ease its modelling and analysis. The integration requires additional refinements and extensions of the completion. For example, software architects might want to specify a subset of the methods of an interface to be invoked asynchronously by the MOM. Finally, we have to extend the PCM towards publish–subscribe systems and their highly dynamic behaviour.

Acknowledgements

We like to thank the members of the Chair of Software Design and Quality (SDQ) at the University of Karlsruhe (TH) for their valuable discussions and thorough review of the contents of this paper.

References

[1] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, Model-based performance prediction in software development: A survey, IEEE Transactions on Software Engineering 30 (5) (2004) 295–310.
[2] M. Bernardo, J. Hillston (Eds.), Formal methods for performance evaluation, in: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM2007, in: Lecture Notes in Computer Science, vol. 4486, Springer-Verlag, Berlin, Germany, 2007.
[3] M. Woodside, D.C. Petriu, K.H. Siddiqui, Performance-related completions for software specifications, in: Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19–25 May 2002, Orlando, Florida, USA, ACM, 2002, pp. 22–32.
[4] V. Grassi, R. Mirandola, A. Sabetta, A model transformation approach for the early performance and reliability analysis of component-based systems, in: I. Gorton, G.T. Heineman, I. Crnkovic, H.W. Schmidt, J.A. Stafford, C.A. Szyperski, K.C. Wallnau (Eds.), Component-Based Software Engineering, 9th International Symposium, CBSE 2006, Västerås, Sweden, June 29 - July 1, 2006, Proceedings, in: Lecture Notes in Computer Science, vol. 4063, Springer, 2006, pp. 270–284.
[5] T. Verdickt, B. Dhoedt, F. Gielen, P. Demeester, Automatic inclusion of middleware performance attributes into architectural UML software models, IEEE Transactions on Software Engineering 31 (8) (2005) 695–711.
[6] V. Cortellessa, P. Pierini, D. Rossi, Integrating software models and platform models for performance analysis, IEEE Transactions on Software Engineering 33 (6) (2007) 385–401.
[7] M. Hapner, R. Burridge, R. Sharma, J. Fialli, K. Stout, Java message service specification - Version 1.1, http://java.sun.com/products/jms/, (last retrieved: January 2009).
[8] S. Becker, H. Koziolek, R. Reussner, The Palladio component model for model-driven performance prediction, Journal of Systems and Software 82 (2009) 3–22.
[9] R.H. Reussner, S. Becker, H. Koziolek, J. Happe, M. Kuperberg, K. Krogmann, The Palladio Component Model, Interner Bericht 2007-21, Universität Karlsruhe (TH), 2007.
[10] S. P. E. C. (SPEC), SPECjms2007 Benchmark, http://www.spec.org/jms2007/, last visit: January 21st, 2009, 2007.
[11] K. Sachs, S. Kounev, J. Bacon, A. Buchmann, Workload characterization of the SPECjms2007 benchmark, in: Formal Methods and Stochastic Models for Performance Evaluation: Fourth European Performance Engineering Workshop (EPEW 2007), in: Lecture Notes in Computer Science, vol. 4748, Springer-Verlag, Berlin, Germany, 2007, pp. 228–244.
[12] X. Wu, M. Woodside, Performance modeling from software components, SIGSOFT Software Engineering Notes 29 (1) (2004) 290–301.
[13] Object Management Group (OMG), Model driven Architecture— Specifications, 2006.
[14] Object Management Group (OMG), Unified modeling language specification: Version 2, Revised Final Adopted Specification (ptc/05-07-04), 2005.
[15] K. Czarnecki, U.W. Eisenecker, Generative Programming, Addison-Wesley, Reading, MA, USA, 2000.
[16] S. Becker, Coupled model transformations, in: WOSP '08: Proceedings of the 7th International Workshop on Software and performance, ACM, New York, NY, USA, 2008, (103–114).
[17] H. Stachowiak, Allgemeine Modelltheorie, Springer Verlag, Wien, 1973.
[18] Object Management Group (OMG), MOF 2.0 Core Specification (formal/2006-01-01), 2006.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

[19] Object Web, The fractal project homepage, 2006, Last retrieved 2008-01-06.
[20] J. Happe, H. Friedrich, S. Becker, R.H. Reussner, A pattern-based performance completion for message-oriented middleware, in: Proceedings of the 7th International Workshop on Software and Performance (WOSP '08), ACM, New York, NY, USA, 2008, pp. 165–176.
[21] V.R. Basili, G. Caldiera, H.D. Rombach, The goal question metric approach, in: J.J. Marciniak (Ed.), Encyclopedia of Software Engineering - 2 Volume Set, John Wiley & Sons, 1994, pp. 528–532.
[22] M. Woodside, V. Vetland, M. Courtois, S. Bayarov, Resource function capture for performance aspects of software components and sub-systems, in: Performance Engineering: State of the Art and Current Trends, in: Lecture Notes in Computer Science, vol. 2047, Springer, Heidelberg, 2001, pp. 239–256.
[23] H. Koziolek, R. Reussner, A model transformation from the palladio component model to layered queueing networks, in: Performance Evaluation: Metrics, Models and Benchmarks, SIPEW 2008, in: Lecture Notes in Computer Science, vol. 5119, Springer-Verlag, Berlin Heidelberg, 2008, pp. 58–78.
[24] S. Becker, T. Dencker, J. Happe, Model-driven generation of performance prototypes, in: Performance Evaluation: Metrics, Models and Benchmarks (SIPEW 2008), in: Lecture Notes in Computer Science, vol. 5119, Springer-Verlag, Berlin Heidelberg, 2008, pp. 79–98.
[25] C. Szyperski, D. Gruntz, S. Murer, Component Software: Beyond Object-Oriented Programming, 2nd edition, ACM Press and Addison-Wesley, New York, NY, 2002.
[26] B.P. Douglass, Real-time design patterns, in: Object Technology Series, Addison-Wesley Professional, 2002.
[27] H. Koziolek, J. Happe, S. Becker, Parameter Dependent Performance Specification of Software Components, in: Proceedings of the Second International Conference on Quality of Software Architectures (QoSA2006), in: Lecture Notes in Computer Science, vol. 4214, Springer-Verlag, Berlin, Germany, 2006, pp. 163–179.
[28] H. Koziolek, S. Becker, J. Happe, R. Reussner, Evaluating performance of software architecture models with the Palladio component model, in: Model-Driven Software Development: Integrating Quality Assurance, IDEA Group Inc., 2008, 95–118.
[29] H. Friedrich, Modellierung nebenläufiger, komponentenbasierter Software-Systeme mit Entwurfsmustern, Master's thesis, Universität Karlsruhe (TH), 2007.
[30] K. Sachs, S. Kounev, J. Bacon, A. Buchmann, Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark, Performance Evaluation 66 (8) (2009) 410–434.
[31] M. Menth, R. Henjes, Analysis of the message waiting time for the fioranoMQ JMS server, in: ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, IEEE Computer Society, Washington, DC, USA, 2006.
[32] D. Freedman, Statistical Models: Theory and Practice, Cambridge University Press, 2005.
[33] E. Lazowska, J. Zahorjan, G.S. Graham, K.C. Sevcik, Quantitative System Performance— Computer System Analysis Using Queueing Network Models, Prentice-Hall, 1984.
[34] S. Becker, Coupled model transformations for qos enabled component-based software design, Karlsruhe Series on Software Quality, vol. 1, Universitätsverlag Karlsruhe, 2008.
[35] K. Sachs, S. Kounev, M. Carter, A. Buchmann, Designing a Workload Scenario for Benchmarking Message-Oriented Middleware, in: SPEC Benchmark Workshop, 2007.
[36] M. Marzolla, Simulation-based performance modeling of UML software architectures, Ph.D. Thesis TD-2004-1, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Mestre, Italy, 2004.
[37] D.C. Petriu, X. Wang, From UML description of high-level software architecture to LQN performance models, in: M. Nagl, A. Schürr, M. Münch (Eds.), in: Proc. of AGTIVE'99 Kerkrade, vol. 1779, Springer, 2000.
[38] A. Di Marco, P. Inveradi, Compositional generation of software architecture performance QN models, in: Proceedings of WICSA 2004, pp. 37–46, 2004.
[39] V. Cortellessa, How far are we from the definition of a common software performance ontology?, in: WOSP '05: Proceedings of the 5th International Workshop on Software and Performance, ACM Press, New York, NY, USA, 2005, pp. 195–204.
[40] V. Cortellessa, A. Di Marco, P. Inverardi, Integrating performance and reliability analysis in a non-functional MDA framework, in: M.B. Dwyer, A. Lopes (Eds.), Fundamental Approaches to Software Engineering 10th International Conference, FASE 2007, Held as Part of the Joint European Conferences, on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24–April 1, 2007, Proceedings, in: Lecture Notes in Computer Science, vol. 4422, Springer, 2007, pp. 57–71.
[41] Y. Liu, A. Fekete, I. Gorton, Design-level performance prediction of component-based applications, IEEE Transactions on Software Engineering 31 (11) (2005) 928–941.
[42] I. Gorton, A. Liu, Performance evaluation of alternative component architectures for enterprise javabean applications, IEEE Internet Computing 7 (3) (2003) 18–23.
[43] G. Denaro, A. Polini, W. Emmerich, Early performance testing of distributed software applications, SIGSOFT Software Engineering Notes 29 (1) (2004) 94–103.
[44] R. Monson-Haefel, D. Chappell, Java Message Service, O'Reilly, 2002.
[45] G.P. Gu, D.C. Petriu, Xslt transformation from uml models to lqn performance models, in: Proceedings of the third international workshop on software and performance, ACM Press, 2002, pp. 227–234.
[46] H. Gomaa, D.A. Menascé, Design and performance modeling of component interconnection patterns for distributed software architectures, in: Proceedings of the Second International Workshop on Software and Performance, ACM Press, 2000, pp. 117–126.

Jens Happe is a Senior Researcher at the Forschungszentrum Informatik (FZI) in Karlsruhe in the division for Software Engineering since September 2008. He received his PhD in computer science in November 2008 from the University of Oldenburg. From April 2005, he was a member of the graduate school TrustSoft of the National German Research Foundation. He got his diploma in computer science in January 2005 from the University of Oldenburg. His interests include software architectures, model-driven quality analyses, and reengineering methods for large-scale software systems.

Steffen Becker is the Department Manager at the Forschungszentrum Informatik (FZI) in Karlsruhe in the division for Software Engineering since January 2008. Before, he graduated from the University of Oldenburg with a PhD in computer science. From July 2003 he was a member of the young investigators excellence program Palladio of the National German Research Foundation. He got his diploma in business administration and computer science combined in 2003 from the Technical University of Darmstadt. He participates regularly in conferences like the QoSA, WOSP or CBSE conference series where he gives presentations, holds tutorials, and participates in panel discussions. His interests include model- driven software development, software architectures, and model-driven quality predictions.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006

Christoph Rathfelder is a Research Scientist at the Forschungszentrum Informatik (FZI) in Karlsruhe in the division for Software Engineering since March 2007. He got his diploma in computer science in February 2007 from the Universität Karlsruhe. His interests include software architectures, service oriented architectures as well as model-driven quality analyses with focus on event-driven architectures.

Holger Friedrich studied Computer Science at the Universität Karlsruhe and at the Norwegian University of Science and Technology (NTNU) in Trondheim. He wrote his diploma thesis about modelling of concurrent, component-based software systems with design patterns. Since 2007 he works at andrena objects in Karlsruhe as software developer. His main interests are agile methods, software quality and novel programming languages.

Ralf H. Reussner is a full professor for software engineering at the University of Karlsruhe and holds the Chair for Software-Design and -Quality since 2006. His research interests include software components, software architecture and model-based quality prediction. Ralf graduated from University of Karlsruhe with a PhD in 2001 and was with the DSTC Pty Ltd, Melbourne, Australia. From 2003 till 2006 he held a Juniorprofessorship for Software Engineering at the University of Oldenburg, Germany. Ralf is organiser of various conferences and workshops, including QoSA and WCOP. As Director of Software Engineering at the IT Research Centre in Karlsruhe (FZI) he consults various industrial partners.

Please cite this article in press as: J. Happe, et al., Parametric performance completions for model-driven performance prediction, Performance Evaluation (2009), doi:10.1016/j.peva.2009.07.006