Modeling Event-Driven Service-Oriented Systems using the Palladio Component Model*

Christoph Rathfelder
FZI Forschungszentrum Informatik
76131 Karlsruhe
Germany
rathfelder@fzi.de

Samuel Kounev
FZI Forschungszentrum Informatik
76131 Karlsruhe
Germany
kounev@fzi.de

ABSTRACT

The use of event-based communication within a Service-Oriented Architecture promises several benefits including more loosely-coupled services and better scalability. However, the loose coupling of services makes it difficult for system developers to estimate the behavior and performance of systems composed of multiple services. Most existing performance prediction techniques for systems using event-based communication require specialized knowledge to build the necessary prediction models. Furthermore, general purpose design-oriented performance models for component-based systems provide limited support for modeling event-based communication. In this paper, we propose an extension of the Palladio Component Model (PCM) that provides natural support for modeling event-based communication. We show how this extension can be exploited to model event-driven service-oriented systems with the aim to evaluate their performance and scalability.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Architectures—*Domain-specific architectures*; C.4 [**Computer Systems Organization**]: PERFORMANCE OF SYSTEMS—*Modeling techniques*; I.6.5 [**Computing Methodologies**]: SIMULATION AND MODELING—*Model Development*

General Terms

Performance

Keywords

Event-driven communication, Performance prediction, Software architecture

*This work was supported by the European Commission (grant No. FP7-216556)

QUASOSS'09, August 25, 2009, Amsterdam, The Netherlands.

1. INTRODUCTION

In the enterprise context, business processes are often driven by real-life events, e.g., arrival of a shipment or receipt of a new order. The combination of event-based communication and Service-Oriented Architecture (SOA) provides a good basis for implementing the IT services underlying such business processes [24]. Compared to synchronous request/reply communication, event-based communication among services using a mediating communication bus promises several benefits [15]. For example, being asynchronous in nature, it allows a *send-and-forget* approach, i.e., a service that sends a message can continue its execution without waiting for the receiver to acknowledge the message or react on it. Furthermore, the loose-coupling of services achieved through the mediating communication bus enables an easy extension of the system since new services have to be connected to the bus only and not to each individual service they communicate with.

However, the above benefits come along with some new challenges for system developers and architects [15]. The event-driven programming model is more complex because the application logic is distributed among multiple independent event handlers and the flow of control during execution is more difficult to track. On the one hand, the asynchronous communication might improve the system performance, on the other hand, the introduction of an intermediary between message senders and receivers introduces some overhead. It is difficult to predict the influences of the event-based communication and the underlying message bus on the system performance especially in the early phases of system development when no implementation is available for testing. Performance modeling and prediction techniques, surveyed in [3] and [5], support the architect in evaluating different design alternatives. However, most existing performance prediction techniques for systems using event-based communication require specialized knowledge to build the necessary prediction models (e.g., [2, 8, 16]). Furthermore, general purpose design-oriented performance models for component-based systems provide limited support for modeling event-based communication.

In this paper, we propose an extension of the Palladio Component Model (PCM) [6] to support the design-oriented modeling of event-driven service-oriented systems. PCM is a design-oriented performance meta-model for modeling component-based software architectures. It allows to explicitly capture component context dependencies (e.g., dependencies on the component usage profile and execution environment) and provides support for a number of differ-

ent performance analysis techniques. The contributions of this paper are: i) an extension of PCM that provides natural support for modeling event-based communication, ii) a mapping of the introduced model extension to existing model elements in the current version of PCM and iii) a case study that demonstrates the application of the proposed modeling approach in the context of an event-based service-oriented system. Using an automated model-to-model transformation from the extended PCM to the original PCM allows to take advantage of existing analytical and simulative analysis techniques significantly reducing the modeling effort and complexity.

The remainder of this paper is organized as follows: Section 2 introduces PCM, which is the basis of this work. Section 3 presents the proposed extension of PCM and the mapping to existing model elements. Section 4 demonstrates our case study. Section 5 presents an overview of related work. Finally, Section 6 concludes with a brief summary and a discussion of ongoing and future work.

2. PALLADIO COMPONENT MODEL

The Palladio Component Model (PCM) [6] is a domain-specific modeling language for modeling component-based software architectures. It supports automatic transformation of design-oriented architectural models to analysis-oriented performance models including layered queueing networks [18], stochastic process algebras [10] and simulation models [6, 4]. In PCM, architectural models are parameterized over the system usage profile and the execution environment. This allows to reuse models in different contexts for different usage scenarios and execution environments. Model artifacts are divided among the developer roles involved in the component-based software engineering process, i.e., component developers, system architects, system deployers and domain experts.

Software components are the core entities of PCM. They contain an abstract behavioral specification called Resource Demanding-Service Effect Specification (RD-SEFF) for each provided component service. RD-SEFFs describe by means of an annotated control flow graph how component services use system resources and call external services provided by other components. Similar to UML activities, RD-SEFFs consist of different types of actions:

- **InternalActions** model resource demands and abstract from computations performed inside a component. To express the performance-relevant resource interaction of the modelled computations, an InternalAction contains a set of **ParametricResourceDemands**. A ParametricResourceDemand specifies the amount of processing power requested from a certain type of resource (e.g., CPU, HDD, or network).

- **AquireAction** and **ReleaseAction** are used to acquire and respectively release a semaphore which can be used for example to model a thread pool. The capacity of these passive resources can be parameterized.

- **ExternalCallActions** represent component invocations of services provided by other components. For each external service call, component developers can specify performance-relevant information about the service input parameters. External service calls are always synchronous in PCM, i.e., the execution is blocked until the call returns.

- **SetVariableActions** are used to specify the returned value of a service or other temporary variables inside RD-SEFFs.

- **Loops** model the repetitive execution of a set of actions. The number of loop iterations can depend on the service input parameters.

- **Branches** represent "exclusive or" splits of the control flow, where only one of the alternatives can be taken. In PCM, the choice can be either probabilistic or it can be determined by a guard. In the former case, each alternative has an associated probability determining the likelihood of its execution. In the latter case, boolean expressions based on the service input parameters determine which alternative is executed.

- **Forks** split the control flow in several parts that are executed in parallel. Usually, forks are asynchronous,i.e., the original control flow continues to execute directly after the parts are forked. However, *Synchronisation-Points* can be used to synchronize the forks. In this case, the original control flow is blocked until all parts have completed their execution.

In the following, we present a set of extensions of PCM that provide explicit support for modeling event-based communication. We then show how the newly introduced model elements can be mapped to existing PCM modeling constructs. Using an automated model-to-model transformation, this enables the reuse of existing performance prediction techniques supported by PCM. Thus, the model extensions significantly reduce the modeling effort and complexity while maintaining backward compatibility with existing model analysis techniques.

3. MODEL EXTENSIONS

In the current version of PCM, only synchronous call-return communication between components is supported. As shown in [11], it is possible to model asynchronous point-to-point communication using a combination of non-synchronized fork actions and external service calls. However, following this approach asynchronous events are modeled using synchronous service calls and therefore a semantic gap between the system implementation and the architecture model is introduced. Moreover, the modeling effort of this workaround solution increases dramatically if event-based communication following the publish-subscribe paradigm is considered. To reduce this overhead and to eliminate the semantic gap, it is necessary to extend PCM with the following elements allowing to model event-based communication explicitly:

- **Events** are the central element of event-based communication. In contrast to interfaces which include method signatures, events only specify the underlying data type. For example an *UpdateStockData* event can be defined as a complex data type that includes a sender ID, a product ID, a timestamp and a number of elements added to or removed from the stock. The event data can be considered when modeling the

behavior of a component processing events of a given type.

- **Event Sources** specify that a component emits a certain type of events. For each emitted event type, the component must provide a respective event source. It is necessary to extend the RD-SEFF with a new action called **Event Action** allowing to instantiate and send events. This action is similar to an ExternalServiceCall, however, it does not block waiting for return values.

- **Event Sinks** specify that a component receives and processes certain types of events. In analogy to the event sources, each consumed event type induces a separate event sink. Only compatible event sources and sinks are allowed to be connected. Similar to provided interfaces which require the specification of RD-SEFFs for provided services, each event sink requires the specification of an **Event Handler**. Event handlers are modeled similar to ordinary component services using RD-SEFFs.

In the following, we present a mapping of the introduced model extensions to existing model elements in the current version of PCM to enable the reuse of the supported performance prediction methods.

Event Sinks are transformed into interfaces provided by the respective component. Each interface includes a service `OnEvent` with the respective event type as input parameter. The RD-SEFF representing the corresponding event handler is then associated with this service. Additionally, it is possible to integrate the marshalling of events as done in [11] for point-to-point connections.

The transformation of **Event Sources** and **Event Actions** is more complex and requires much more modeling effort if done manually. First, it is necessary to explicitly require an interface for each connected event sink, because PCM supports only 1-to-1 connections between required and provided interfaces. Second, every event action is substituted by a fork action with a separate forked RD-SEFF for each connected event sink. Each RD-SEFF includes an external service call to one of the required `OnEvent` interfaces and respective associated service. In the following section, we present a case study demonstrating these mappings by modeling a concrete event-driven service-oriented system.

4. CASE STUDY

Figure 1 shows a simplified event-driven service-oriented system used to track the inventory of a supermarket. When new products are delivered, they have to pass a gate equipped with an `RFIDScanner`. The latter creates an `UpdateStockData` event for each scanned product. It is possible to combine several scanned products of the same type in one event. This reduces the amount of events but increases the complexity of the `RFIDScanner`. `UpdateStockData` events are also produced by the `CashdeskService`. For each product sold, an `UpdateStockData` event is created to update the supermarket inventory.

The events produced by the `CashdeskService` and the `RFIDScanner` are consumed and processed by the three services: `LoggingService`, `InventoryManagementService`, and `OrderManagementService`. The `LoggingService` persists all

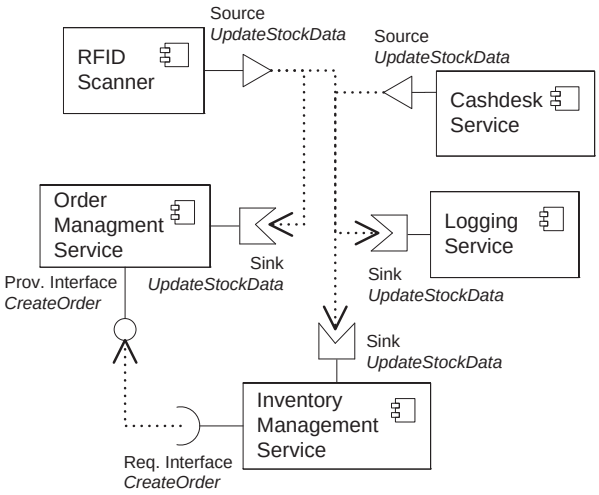

Figure 1: Example Scenario

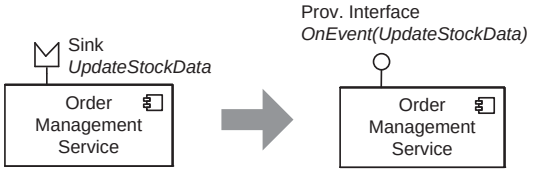

Figure 2: Transformation of `OrderManagementService`

events in a local database for 48 hours. The `InventoryManagementService` updates the supermarket inventory to reflect the flow of goods in and out of the supermarket. When goods of a given type are depleted, the `InventoryManagementService` calls the `OrderManagementService` to place an order with the respective supplier. In addition, the `OrderManagementService` receives all events sent by the `RFIDScanner` in order to register incoming order deliveries.

We now use the mappings defined in Section 3 to model this event-driven service-oriented system. We concentrate on the `RFIDScanner` and the `OrderManagementService` to demonstrate the mapping of event sources and sinks. The other components can be modeled in a similar way. The event `UpdateStockData` is realized as a PCM complex data type with attributes `senderID`, `productID`, `timestamp` and `numberOfItems`. `numberOfItems` is positive if products are added to the stock and negative if they are sold at the cash desk. The mapping of the `OrderManagementService` is illustrated in Figure 2. The event sink is substituted by a provided interface which includes the method `OnEvent` with an input parameter of type `UpdateStockData`. The RD-SEFF of the event handler can be used without changes for the respective `OnEvent` method.

The mapping of the `RFIDScanner` is more complex and requires some more modeling effort. For each connected event sink, the respective event interface with its `OnEvent` method has to be declared as a required interface of the `RFIDScanner` component. As illustrated in Figure 3, the service requires this interface three times. The behavior of the `RFIDScanner` is modeled as RD-SEFF. This RD-SEFF includes at least one `EventAction` which is responsible to instantiate and send the `UpdateStockData` event. Each of this `Even-`

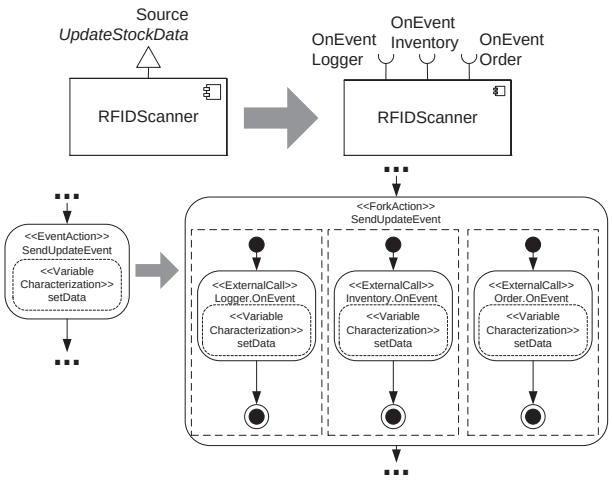

Figure 3: Transformation of `RFIDScanner`

`tAction` is substituted by a `ForkActions`. In this fork action, a separate RD-SEFF is created for each required interface corresponding to an event sink connected to `RFIDScanner`. Thus, the `onEvent` methods of `LoggerService`, `InventoryManagementService`, and `OrderManagementService` are invoked in parallel. The instantiation of the event data object is copied from the `EventAction` into each `ExternalCall`. Since the `ForkAction` does not define a synchronization point, the execution of the parent RD-SEFF continues immediately. For this reason, the resulting behavior resembles an asynchronous event dissemination although strictly speaking the semantic of the model is different.

The example we described demonstrates that the modeling effort required to model event-based systems using our proposed PCM extensions can be reduced significantly especially when event sources are connected with multiple sinks. In combination with an automated model-to-model transformation following the described mapping, it is possible to continue to use the existing performance prediction methods supported by PCM.

5. RELATED WORK

The work related to the results presented in this paper can be classified into two areas: i) design-oriented performance meta-models that can be used for modeling service-oriented systems and ii) performance analysis techniques specialized for event-based systems including message-oriented middleware.

Over the last fifteen years a number of approaches have been proposed for integrating performance prediction techniques into the software engineering process. Efforts were initiated with Smith's seminal work pioneered under the name of Software Performance Engineering (SPE) [26]. Since then a number of design-oriented performance meta-models have been developed by the performance engineering community. The most prominent examples are the UML SPT profile [22] and its successor the UML MARTE profile [23], both of which are extensions of UML as the de facto standard modeling language for software architectures. Other proposed meta-models include CSM [25], SPE-MM [27] and KLAPER [9]. Design-oriented performance models are built manually during system development and are used at design and deployment time to evaluate alternative system designs and/or predict the system performance for capacity planning purposes. A recent survey of model-based performance prediction techniques was published in [3]. In recent years, with the increasing adoption of component-based software engineering, the performance evaluation community has focused on adapting and extending conventional SPE techniques to support component-based systems which are typically used as foundation for building modern service-oriented systems. A survey of methods for component-based performance-engineering was published in [5].

We now present an overview of existing performance modeling and analysis techniques for systems based on message-oriented middleware (MOM). In [14], an analytical model of the message processing time and throughput of the WebSphereMQ JMS server is presented and validated through measurements. The message throughput in the presence of filters is studied and it is shown that the message replication grade and the number of installed filters have a significant impact on the server throughput. Several similar studies using Sun Java System MQ, FioranoMQ, ActiveMQ and BEA WebLogic JMS server were published by the same authors. A more in-depth analysis of the message waiting time for the FioranoMQ JMS server was published in [20]. The authors study the message waiting time based on an $M/G/1-\infty$ queue approximation and perform a sensitivity analysis with respect to the variability of the message replication grade.

A method for modeling MOM systems using *performance completions* is presented in [12]. Performance completions provide a general mechanism for including low-level details of execution environments into abstract performance models. The authors propose a pattern-based language for configuring the type of message-based communication. Model-to-model transformations are used to integrate low-level details of the MOM system into high-level software architecture models. In [19], an approach to predicting the performance of messaging applications based on the Java Enterprise Edition is proposed. The prediction is carried out during application design, without access to the application implementation. This is achieved by modeling the interactions among messaging components using queueing network models, calibrating the performance models with architecture attributes, and populating the model parameters using a lightweight application-independent benchmark.

Several performance modeling techniques specifically targeted at distributed publish/subscribe systems exist in the literature. In [16], an analytical model of publish/subscribe systems that use hierarchical identity-based routing is presented. The model is based on continuous time birth-death Markov chains. The proposed modeling approach, however, does not provide means to predict the event delivery latency and it suffers from a number of restrictive assumptions. Many of these assumptions are relaxed in [21] where a generalization of the model is proposed, however, the generalized model is still limited to systems based on hierarchical identity-based routing. In [8], an analytical model of pub/sub systems based on subscription forwarding is proposed. The authors provide closed form analytical expressions for the overall network traffic required to disseminate subscriptions and propagate notifications, as well as for the message forwarding load on individual system nodes. However, the same restrictive assumptions as in [16] are made about the system topology and the distribution of publish-

ers and subscribers among brokers. In [17], a methodology for workload characterization and performance modeling of distributed event-based systems is presented. A workload model of a generic system is developed and analytical analysis techniques are used to characterize the system traffic and to estimate the mean notification delivery latency. For more accurate performance prediction queueing Petri net models are used. While this technique is applicable to a wide range of systems, it relies on monitoring data obtained from the system and it is therefore only applicable if the system is available for testing.

In [2], the authors present an attempt to formally model a publish/subscribe communication system as a classical distributed computation. The authors formalize the concept of information availability and model a few properties of the computation, namely completeness and minimality, that capture the expected behavior of a publish/subscribe system from an application viewpoint. In [1, 28] a computational model of a publish/subscribe notification service is proposed, where the latter is abstracted as a black box connecting all participants in the computation. While some interesting results are presented, the proposed model is rather coarse grained and it is based on the assumption that the subscription and diffusion delays are known which is not realistic to expect. In [7], Bricconi et al. present a simple model of the Jedi publish/subscribe system. The model is mainly used to calculate the number of notifications received by each broker using a uniform distribution of subscriptions. To model the multicast communication, the authors introduce a spreading coefficient between 0 and 1 which models the probability that a broker at a given distance (in hops) from the publishing broker receives a published notification. Finally, in [13], probabilistic model checking techniques and stochastic models are used to analyze publish/subscribe systems. The communication infrastructure (i.e., the transmission channels and the publish/subscribe middleware) are modeled by means of probabilistic timed automata. Application components are modeled by using statechart diagrams and then translated into probabilistic timed automata. The analysis considers the probability of message loss, the average time taken to complete a task and the optimal message buffer sizes.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed an extension of PCM providing native support for modeling event-driven service-oriented systems. Furthermore, we described a mapping of the newly introduced modeling elements to existing elements in the current version of PCM. This allows to continue to use existing performance prediction techniques supported in PCM. We presented a case study demonstrating the application of the proposed modeling approach in the context of an event-based service-oriented system.

The proposed extensions of PCM allow a semantically correct modeling of event-driven systems. In combination with an automatic model-to-model transformation based on the proposed mapping, our approach significantly reduces the effort to build and analyze models of systems that use event-based communication. The automation of the transformation is part of our current work. Furthermore, we plan to introduce some further constructs in PCM to support the modeling of an event bus.

As part of our future work, we intend to study the performance-relevant influence factors associated with event-based communication in service-oriented systems. As a first step, we intend to focus on the influence of persistent vs. non-persistent message delivery, the number of event consumers, and the event filtering mechanisms. Based on the results, we plan to extend the models and respective transformations to consider these factors in order to increase the prediction accuracy.

7. REFERENCES

[1] R. Baldoni, R. Beraldi, S. T. Piergiovanni, and A. Virgillito. On the modelling of publish/subscribe communication systems. *Concurrency and Computation: Practice and Experience*, 17(12):1471–1495, 2005.

[2] R. Baldoni, M. Contenti, S. Piergiovanni, and A. Virgillito. Modeling publish/subscribe communication systems: towards a formal approach. pages 304–311, Jan. 2003.

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

[4] S. Becker. *Coupled Model Transformations for QoS Enabled Component-Based Software Design*, volume 1 of *Karlsruhe Series on Software Quality*. Universitätsverlag Karlsruhe, 2008.

[5] S. Becker, L. Grunske, R. Mirandola, and S. Overhage. Performance Prediction of Component-Based Systems: A Survey from an Engineering Perspective. In R. Reussner, J. Stafford, and C. Szyperski, editors, *Architecting Systems with Trustworthy Components*, volume 3938 of *Lecture Notes in Computer Science*, pages 169–192. Springer-Verlag Berlin Heidelberg, 2006.

[6] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82:3–22, 2009.

[7] G. Bricconi, E. D. Nitto, and E. Tracanella. Issues in analyzing the behavior of event dispatching systems. In *Proc. of 10th Intl. Workshop on Software Specification and Design*, pages 95–103, 2000.

[8] S. Castelli, P. Costa, and G. P. Picco. Modeling the communication costs of content-based routing: the case of subscription forwarding. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 38–49, New York, NY, USA, 2007. ACM.

[9] V. Grassi, R. Mirandola, and A. Sabetta. Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software*, 80(4):528–558, April 2007.

[10] J. Happe. *Concurrency Modelling for Performance and Reliability Prediction of Component-Based Software Architectures*. PhD Thesis, University of Oldenburg, Germany, 2008.

[11] J. Happe, S. Becker, C. Rathfelder, H. Friedrich, and R. H. Reussner. Parametric Performance Completions for Model-Driven Performance Prediction. *Performance Evaluation*, 2009. To appear.

[12] J. Happe, H. Friedrich, S. Becker, and R. H. Reussner. A pattern-based Performance Completion for Message-oriented Middleware. In *Proc. of the 7th International Workshop on Software and Performance*, pages 165–176. ACM, 2008.

[13] F. He, L. Baresi, C. Ghezzi, and P. Spoletini. Formal analysis of publish-subscribe systems by probabilistic timed automata. In *27th IFIP WG 6.1 Intl. Conf. on Formal Techniques for Networked and Distributed Systems*, volume 4574 of *LNCS*, pages 247–262, 2007.

[14] R. Henjes, M. Menth, and C. Zepfel. Throughput Performance of Java Messaging Services Using WebsphereMQ. In *ICDCSW '06: Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems*, Washington, DC, USA, 2006. IEEE Computer Society.

[15] G. Hohpe and B. Woolf. *Enterprise integration patterns*. Addison-Wesley, 2008.

[16] M. A. Jaeger and G. Mühl. Stochastic Analysis and Comparison of Self-Stabilizing Routing Algorithms for Publish/Subscribe Systems. In *Proc. of the 13th IEEE Intl. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, 2005.

[17] S. Kounev, K. Sachs, J. Bacon, and A. Buchmann. A methodology for performance modeling of distributed event-based systems. In *Proc. of the 11th IEEE Intl. Symposium on Object/Component/Service-oriented Real-time Distributed Computing*, May 2008.

[18] H. Koziolek and R. Reussner. A Model Transformation from the Palladio Component Model to Layered Queueing Networks. In *Performance Evaluation: Metrics, Models and Benchmarks, SIPEW 2008*, 2008.

[19] Y. Liu and I. Gorton. Performance Prediction of J2EE Applications Using Messaging Protocols. pages 1–16. 2005.

[20] M. Menth and 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*, Washington, DC, USA, 2006. IEEE Computer Society.

[21] G. Mühl, A. Schröter, H. Parzyjegla, S. Kounev, and J. Richling. Stochastic Analysis of Hierarchical Publish/Subscribe Systems. In *Proceedings of the 15th International European Conference on Parallel and Distributed Computing (Euro-Par 2009)*, 2009.

[22] Object Management Group (OMG). UML Profile for Schedulability, Performance, and Time (SPT), v1.1, Jan. 2005.

[23] Object Management Group (OMG). UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), May 2006.

[24] M. Papazoglou and W.-J. van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, July 2007.

[25] D. Petriu and M. Woodside. An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *Software and Systems Modeling*, 6(2):163–184, June 2007.

[26] C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[27] C. U. Smith, C. M. Lladó, V. Cortellessa, A. Di Marco, and L. G. Williams. From UML models to software performance results: an SPE process based on XML interchange formats. In *5th Intl. Workshop on Software and Performance (WOSP)*, pages 87–98, 2005.

[28] A. Virgillito. *Publish/Subscribe Communication Systems: From Models to Applications*. PhD thesis, Universita La Sapienza, 2003.