Palladio Workbench: A Quality-Prediction Tool for Component-Based Architectures

Christoph Rathfelder* and Benjamin Klatt*
*FZI Research Center for Information Technology
Karlsruhe Institute of Technology, Karlsruhe, Germany
Email: rathfelder@fzi.de, klatt@fzi.de

***Abstract*—Today, software engineering is challenged to handle more and more large-scale distributed systems with a guaranteed level of service quality. Component-based architectures have been established to build more structured and manageable software systems. However, due to time and cost constraints, it is not feasible to use a trial and error approach to ensure that an architecture meets the quality of service (QoS) requirements. In this tool demo, we present the Palladio Workbench that permits the modeling of component-based software architectures and the prediction of its quality characteristics (e.g., response time and utilization). Additional to a general tool overview, we will give some insights about a new feature to analyze the impact of event-driven communication that was added in the latest release of the Palladio Component Model (PCM).**

I. Introduction

The central idea of component-based software engineering (CBSE) is to compose complex software systems of basic components. The initial goal was to support the reuse of those components, but clear encapsulation of functionality and the contractual interaction with components based on their interfaces allow a better control and predictability of this single unit. However, predicting the quality (e.g., performance or reliability) of the component-based software system is much more complex than understanding the implementation of a single unit. Multiple influence factors such as the deployment of the components, the usage profile, the required external services and for sure the implementation of the component itself have impact on the systems overall quality. The Palladio Component Model (PCM) permits the modeling of component-based architectures and provides different models for those influence factors. Due to this independent models, it is possible to predict the systems behavior for variations of one or more of these factors without the need to change the others.

The overall architecture is not only influenced by the components internals but also by their way of interaction. In recent years, the event-driven paradigm gets more and more attention to build more scalable and loosely coupled systems. In event-driven systems, components communicate by sending and receiving events. Compared to synchronous communication using, for example, remote procedure calls (RPCs), event-driven communication among components promises several benefits like increased flexibility or better scalability [1]. However, the event-driven software model is more complex as application logic is distributed among multiple independent event handlers and the flow of control during execution can be hard to track.

This work was partially funded by the EU (grant No. FP7-216556)

However, current performance modeling and prediction techniques for component-based systems, surveyed in [2], do not support modeling systems using event-driven communication. Also the PCM did not in the previous versions. With the latest release published in March 2011, this has become a new feature of the Palladio Workbench. Using an automated two-stepped model transformation enables the integration of different middleware models without changing the model of the whole system. We demonstrate the application of this new feature using a real-world case study of a traffic monitoring system, which we already used for validating the prediction result's accuracy [3].

The remainder of this paper is organized as follows. In Sect. II, we introduce the PCM meta-model for component-based software architectures. Sect III gives a short overview on the tool and the underlying technology. Sect. IV presents further details of the extension for the analysis of event-based communication that we will present in the demo. The case study used on the demo is described in Sect. V before we present some tools related to the PCM in Sect. VI. We conclude with a brief summary at the end.

II. Palladio Component Model

The Palladio Component Model (PCM) is a component meta-model with focus on performance prediction. The following gives a brief overview of the PCM. A more detailed and technical description is given in [4].

The PCM is a domain specific meta-model. It describes the performance-relevant aspects of component-based software architectures, often used in business information systems. The Palladio Workbench provides tools (see Sec. III) to create and analyze PCM model instances.

The performance of a component-based software system is influenced by four factors [4]. The first is the implementation of the component itself. Second, the component performance depends on required services. Third, it depends on the deployment platform (e.g. hardware, middleware, networks). Finally, the way the component is used must be considered. In the PCM, each of these influence factors is

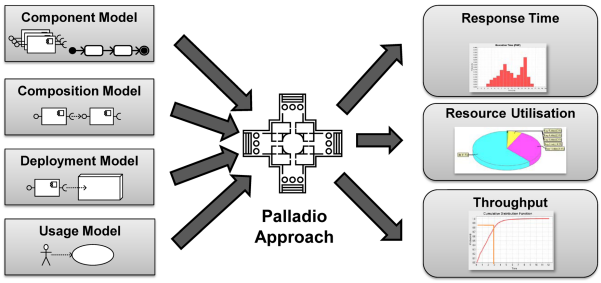

Figure 1. Palladio Overview

modeled in a specific sub-model. The composition of these models forms a PCM instance.

The *Component Model* consists of component specifications including a model of their behavior. Components refer to interfaces (comparable to signature lists) which are provided or required. Provided interfaces specify the services a component offers. Required interfaces are external services the component needs to fulfill its purpose. The PCM provides a description language, called *ResourceDemandingServiceEffectSpecification* (RDSEFF), to specify the behavior of components as an abstraction that is quality-related.

The *Composition Model* describes the structure of the system by the composition of the components to be used. The components are connected via their provided and required interfaces to satisfy all required interfaces.

In the *Deployment Model* the components of the system are allocated to physical resources. The characteristics of the resource containers (e.g., processor speed and number of cores) are specified in the resource environment model.

The *Usage Model* contains the workload induced by the system's end-users. The workload specifies, for example, how many users invoke the system as well as their inter-arrival time between two invocations. Furthermore, it includes the characteristics of the request parameters such as the number of items in a list or the size of the input data.

As sketched in Figure 1, PCM supports the evaluation of different performance attributes, including response time, maximum throughput, resource utilization, and QoS levels. To realize these predictions, the Palladio approach provides a model that reflects the already mentioned four aspects implementation, service composition, deployment and service usage as separated sub models.

III. Palladio Workbench

The Palladio Workbench provides an advanced tooling to create PCM models and to predict quality characteristics of the software architecture. This workbench includes graphical editors aligned with the UML syntax to create the different models. As soon as a complete PCM instance is prepared, a prediction run can be configured and started. The configuration includes the type of prediction (e.g., simulation or one of the analytical solutions). Furthermore, the configuration includes a variety of prediction options (e.g., the data storage or stopping criteria) as well as the combination of sub-models to be used in the prediction. During a prediction run, the Palladio workbench provides a simulation dock with status information about the actual run. When the prediction is finished, a list of available sensors that have been automatically installed on resource containers and component interfaces are presented. For each of those sensors, various visualizations are provided and can be used to understand the quality properties of the specific parts of the architecture. Due to this integration of modeling, transformation, performance analyses and visualization, the performance evaluation of a system does not require any expert knowledge about the underlying prediction techniques.

Technically, the Palladio Workbench is based on the Eclipse platform and makes extensive use of the Eclipse Modeling Project as well as corresponding standards such as data interfaces (e.g., XMI) or infrastructure frameworks (e.g., Eclipse Bundles).

The development of the Palladio Component Model (PCM) started in 2003 at the University of Oldenburg, and since 2006 it is continued at the Karlsruhe Institute of Technology (KIT) and the FZI Research Center for Information Technology. Meanwhile, it is extensively used and enhanced in research projects as well as industrial case studies with enterprises such as ABB, IBM, SAP as well as mid-size companies. In several industrial and research case studies (e.g. [5], [6], [3]) we validated the prediction accuracy. Additionally, the applicability of the PCM approach was investigated in an empirical study [7]. The results show that the quality of the models differs less than 10% from the predictions achieved with a reference model. Furthermore, over 80% of the subjects were able to rank the given design alternatives correctly, which indicates the appropriateness of the approach itself.

Further details about the PCM are given in [4] and can be found on the project website[1].

IV. Integration of Event-based Communication

With the latest release of the Palladio Workbench, published in March 2011, a new feature to analyze the impact of event-driven communication on the whole software architecture was added. This includes new modeling as well as prediction capabilities to the Palladio Workbench.

For a semantically correct modeling of event-driven communication, we extended the PCM meta-model with new constructs (e.g., `EventSource`, `EventSink`, `EventEmitAction`) [8], [9] and enhanced the graphical editors to support them. Furthermore, as previously shown in [10], an automatic model-to-model transformation is used to transform these new elements into existing PCM modeling constructs in to permit the usage of the various existing model analysis techniques. Additional to that, the

[1] www.palladio-simulator.com

transformation is able to automatically weave configurable middleware repositories into the system architecture. Such repositories encapsulate the middleware specific characteristics and can be reused for different software architectures.

To realize the separation of platform-independent and platform-specific influence factors as mentioned above, we divided the transformation internally into two parts. The extended PCM model, that includes event-driven communication, is first transformed into a generic model representing the event transmission by a combination of `ExternalCallActions` and `Forks` as sketched in [9]. This model does not include platform-specific details such as resource demands or sizes of thread-pools. In the second step, a platform-specific middleware model, specified in a separated component model, is woven into the PCM model instance. This platform-specific component repository has to be created only once and can be reused in different contexts.

The result of the transformation is a model instance which includes platform-specific details and can be analyzed by means of analytical or simulation techniques. The transformation is performed automatically and is fully transparent for the analyst. It is only required to select the component repository that corresponds to the used middleware.

Fig. 2 shows the result of the platform-independent part of the transformation. For each Sink and Source an additional component is added. These composite components include several components that represent different steps in the middleware's event processing:

- **SourcePort** Interaction between source component and middleware.
- **DistributionPreparation** Processing within the middleware done once per event (e.g., marshaling before the distribution).
- **EventDistribution** Splitting the control flow and distributing the events to all sinks.
- **EventSender** Processing within the middleware that per connected sink (e.g., communication handshake).
- **EventReceiver** Receiver-side event processing within the middleware (e.g., receiving from the communication channel).
- **SinkPort** Pass event to the receiving component.

Each of these internal components includes an `ExternalCallAction` to trigger a dedicated interface of the middleware model.

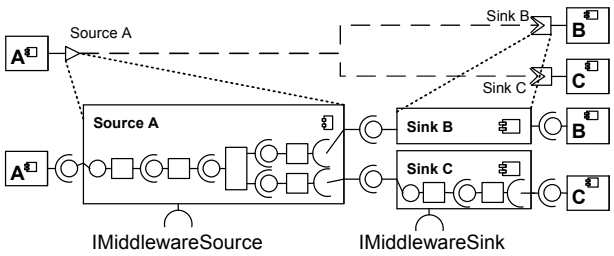

Figure 2. Platform-independent Result of the Transformation

V. Example Case Study

To demonstrate the Palladio Workbench and the performance predictions, give some insights of a case study that was used for validating the new modeling and prediction capabilities for event-driven communication ([3], [11]). The system we studied was partially developed as part of the TIME project (Transport Information Monitoring Environment) [12] at the University of Cambridge. The system is based on the SBUS middleware [13] which supports peer-to-peer event-driven communication including both continuous streams of data (e.g., from sensors), asynchronous events, and synchronous RPC.

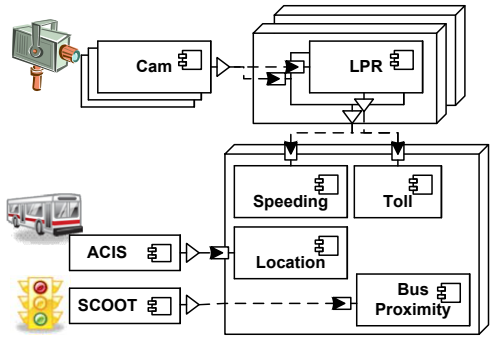

Figure 3. Case Study Overview

Our implementation of this application uses different classes of SBUS components (see Figure 3) described below. With ACIS and SCOOT, GPS tracking data of buses and status updates on traffic lights are transferred into the system. Several cameras (Cam) take pictures of the traffic. These pictures are received and processed by license plate recognition (LPR) components. 4 different classes of components are receiving and processing these events about bus tracking, traffic lights status and license plate detections. The LPR events are used to detect speeding and to calculate the toll for entering the city. The tracking information is used to calculate the position of all buses. Combined with the traffic light status this information is used to calculate if buses are reaching or waiting in front of red traffic lights. The aim of the case study was to asses different deployment and configuration scenarios.

In our demo we will show the use of the Palladio Workbench to analyze and predict the different design and deployment options.

VI. Related Tools

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 on Software Performance Engineering (SPE) [14]. Based on this work, the SPEED tool [15], [16] was developed. The SPE method was the first elaborated, practically applicable comprehensive approach for early, design-time performance prediction for software systems. However, it is a monolithic approach

which does not consider the new requirements that are introduced by component-based architectures like adaptability of the models and reuse of component models [7].

Queuing Petri nets are often used to model performance critical systems in general and are not limited to software systems. With QPME [17], Kounev et al developed a modeling and prediction tool, that already has been validated in several case studies (e.g. [18], [19]). However, due to the use of queuing Petri nets as modeling language, performance modeling and prediction requires specific knowledge.

A recent survey and complete overview of analysis techniques specific to component-based performance-engineering was published in [2].

CB-SPE [20] applies the SPE method to component-based systems by separating component performance models. However, the software architect has to specify a performance critical scenario in analogy to the not scalable and proprietary SPE method and the influence of parameter values is neglected.

With SAPS [21] a simulation-based approach is presented, that uses annotated UML diagrams to derive a simulation model. With UML, they have chosen an established modeling language with good tool support. However, as they have an SAPS specific annotation language, this annotation are not supported by the general UML modeling tools.

VII. Conclusion

In this research tool demo we introduced the Palladio Component Model with its architecture model and prediction techniques implemented in the Palladio Workbench. In addition we gave insights about a new workbench feature to analyze the impact of event-driven communication introduced in the latest PCM version. The prediction results allow to compare different design alternatives and evaluate the system performance under different load situations.

References

[1] B. Hohpe, Gregor ; Woolf, *Enterprise integration patterns : designing, building, and deploying messaging solutions*. Addison-Wesley, 2008.

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

[3] C. Rathfelder, D. Evans, and S. Kounev, "Predictive Modelling of Peer-to-Peer Event-driven Communication in Component-based Systems," in *Proc. of EPEW'10*, vol. 6342. Springer, 2010, pp. 219–235.

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

[5] N. Huber, S. Becker, C. Rathfelder, J. Schweflinghaus, and R. Reussner, "Performance Modeling in Industry: A Case Study on Storage Virtualization," in *Proc. of ICSE 2010*, 2010.

[6] F. Brosig, S. Kounev, and K. Krogmann, "Automated Extraction of Palladio Component Models from Running Enterprise Java Applications," in *Proc. of ROSSA 2009*, 2009.

[7] A. Martens, S. Becker, H. Koziolek, and R. Reussner, "An Empirical Investigation of the Effort of Creating Reusable Models for Performance Prediction," in *Proc. of CBSE'08*, vol. 5282. Springer, 2008, pp. 16–31.

[8] C. Rathfelder and S. Kounev, "Model-based Performance Prediction for Event-driven Systems (Fast Abstract)," in *Proc. of DEBS2009*, 2009.

[9] Christoph Rathfelder and Samuel Kounev, "Modeling Event-Driven Service-Oriented Systems using the Palladio Component Model," in *Proc. of QUASOSS 2009*. ACM, 2009.

[10] C. Rathfelder, B. Klatt, S. Kounev, and D. Evans, "Towards Middleware-aware Integration of Event-based Communication into the Palladio Component Model (Poster Paper)," in *Proc. of DEBS 2010*. ACM, 2010.

[11] B. Klatt, C. Rathfelder, and S. Koounev, "Integration of event-driven communication into the palladio component model," in *QoSA 2011*, 2011 (submitted for review).

[12] J. Bacon, A. R. Beresford, D. Evans, D. Ingram, N. Trigoni, A. Guitton, and A. Skordylis, "TIME: An open platform for capturing, processing and delivering transport-related data," in *Proceedings of the IEEE consumer communications and networking conference*, 2008, pp. 687–691.

[13] D. Ingram, "Reconfigurable middleware for high availability sensor systems." in *Proc. of DEBS 2009*. ACM Press, 2009.

[14] C. U. Smith, *Performance Engineering of Software Systems*. Addison-Wesley, 1990.

[15] C. U. Smith and L. G. Williams, "Performance Engineering Evaluation of Object-Oriented Systems with SPEED," in *Computer Performance Evaluation: Modelling Techniques and Tools*, R. Marie, Ed., vol. 1245, 1997.

[16] Connie U. Smith and Lloyd G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

[17] S. Kounev and C. Dutz, "QPME - A Performance Modeling Tool Based on Queueing Petri Nets," *ACM SIGMETRICS Performance Evaluation Review (PER), Special Issue on Tools for Computer Performance Modeling and Reliability Analysis*, vol. 36, no. 4, pp. 46–51, Mar. 2009.

[18] S. Kounev, "Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets," *IEEE Trans. on Softw. Eng.*, vol. 32, no. 7, 2006.

[19] S. Kounev, K. Bender, F. Brosig, N. Huber, and R. Okamoto, "Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics," in *ICST'11*, 2011.

[20] A. Bertolino and R. Mirandola, "CB-SPE Tool: Putting Component-Based Performance Engineering into Practice," in *Proc. of CBSE 2004*, vol. 3054. Springer, 2004, pp. 233–248.

[21] S. Balsamo and M. Marzolla, "A Simulation-Based Approach to Software Performance Modeling," in *Proc of ESEC 2003*. ACM, 2003, pp. 363–366.