# Integration of Event-Based Communication in the Palladio Software Quality Prediction Framework*

Benjamin Klatt
FZI Research Center for Information Technology
Karlsruhe, Germany
klatt@fzi.de

Christoph Rathfelder
FZI Research Center for Information Technology
Karlsruhe, Germany
rathfelder@fzi.de

Samuel Kounev
Karlsruhe Institute of Technology
Karlsruhe, Germany
kounev@kit.edu

## ABSTRACT

Today, software engineering is challenged to handle more and more large-scale distributed systems with guaranteed quality-of-service. Component-based architectures have been established to build such systems in a more structured and manageable way. Modern architectures often utilize event-based communication which enables loosely-coupled interactions between components and leads to improved system scalability. However, the loose coupling of components makes it challenging to model such architectures in order to predict their quality properties, e.g., performance and reliability, at system design time. In this paper, we present an extension of the Palladio Component Model (PCM) and the Palladio software quality prediction framework, enabling the modeling of event-based communication in component-based architectures. The contributions include: i) a meta-model extension supporting events as first class entities, ii) a mode-to-model transformation from the extended to the original PCM, iii) an integration of the transformation into the Palladio tool chain allowing to use existing model solution techniques, and iv) a detailed evaluation of the reduction of the modeling effort enabled by the transformation in the context of a real-world case study.

## 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

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


*QoSA+ISARCS'11,* June 20–24, 2011, Boulder, Colorado, USA.


## Keywords

Event-based Communication, Component-based Architecture, Performance Prediction

## 1. INTRODUCTION

In event-based component-based systems, components communicate by sending and receiving events. Compared to synchronous communication using, e.g., remote procedure calls (RPCs), event-based communication among components promises several benefits [10]. For example, being asynchronous in nature, it allows a *send-and-forget* approach, such that a component that sends an event can continue its execution without waiting for the receiving component to acknowledge the event or to react on it. Furthermore, the loose coupling of components provides increased flexibility and better scalability.

However, the event-based paradigm is more complex given that application logic is distributed among multiple independent event handlers making the control flow during execution hard to track. This increases the difficulty of modeling event-driven component-based systems for quality prediction at system design and deployment time. The latter is essential in order to ensure that systems are designed and sized to provide an adequate quality-of-service to applications at a reasonable cost.

Performance modeling and prediction techniques for component-based systems, surveyed in [16], support the architect in evaluating different design alternatives. However, most general-purpose performance meta-models for component-based systems provide limited support for modeling event-based communication. Furthermore, existing performance prediction techniques specialized for event-based systems (e.g., [20]) are focused on modeling the routing of events in the system as opposed to modeling the interactions and message flows between the communicating components.

The Palladio Component Model (PCM) [5] is a mature meta-model for component-based software architectures enabling quality predictions (e.g., performance and reliability) at system design time. Architecture models are annotated with quality attributes and a number of different analysis techniques can be applied to evaluate the quality of the modeled architecture. However, in its current version, PCM is limited to synchronous point-to-point communication and event-based communication can only be modeled using a workaround approach, as demonstrated in [24]. The modeling effort incurred by this workaround approach is very high and it provides limited flexibility to evaluate different design alternatives. In [26], we briefly sketched a basic extension of

PCM to support the modeling of event-based communication. In a follow up poster paper [25], we described an idea of using a model-to-model transformation to map the newly introduced model elements to existing PCM model elements allowing to use the available analytical and simulative analysis techniques, e.g., [5, 18, 17], while significantly reducing the modeling effort.

In this paper, we present the complete and finalized PCM extension, based on our preliminary ideas sketched in [26] and [25], combined with a refined model-to-model transformation from the extended PCM to the original PCM allowing to weave in middleware-specific components. The contributions of the paper are: i) a complete meta-model extension for modeling event-based communication supporting events as first class entities, ii) an automatic transformation from the extended to the original meta-model supporting the separation of platform-independent and platform-specific aspects of the event-based communication, iii) an implementation and integration of the PCM extension and the proposed model-to-model transformation into the Palladio tool chain, iv) a case study evaluating the benefits of the proposed transformation approach in terms of reducing the effort for modeling event-based communication in component-based architectures.

The results of the case study show that using the transformation the modeling effort can be reduced by more than 80% with prediction accuracy within 10% of the manual workaround-based approach.

The remainder of this paper is organized as follows. Section 2 introduces PCM which is the basis of our work. Section 3 presents the PCM extensions to enable the modeling of event-based communication. Next, we present the model-to-model transformation followed by an evaluation of our approach in the context of a real-world traffic monitoring case study. Finally, in Section 6, we present an overview of related work and conclude with a brief summary and a discussion of ongoing and future work in Section 7.

# 2. FOUNDATION

The Palladio Component Model (PCM) [5] is a design-oriented meta-model for component-based software architectures with a focus on quality predictions (e.g., performance, reliability). It supports automatic transformations of architecture-level performance models to predictive performance models including layered queuing networks [17], stochastic process algebras [7], queuing Petri nets [18], and simulation models [5, 4].

PCM includes domain specific meta-models for modeling the major aspects that have influence on the quality-of-service provided by software components, including their internal control flow, the allocation of resources, the used external services, and the usage profile. The *component repository* describes components in terms of their provided and required interfaces. Furthermore, it contains Resource Demanding Service Effect Specifications (RD-SEFFs) which describe the internal processing of a component from a high-level perspective with a focus on quality-of-service-relevant aspects. The *composition model* describes the assembled components that form the static architecture of the system. In this model, components are connected based on their required and provided interfaces. The deployment of the components is described in the *allocation model*. This links the components to resource containers that are described in the *resource environment model*. Finally, the *usage model* describes the usage behavior explicitly considering the input parameters passed to the system when invoking services.

Each of the above described models encapsulates a specific performance-relevant aspect. Thanks to this, it is possible to change individual sub-models independently of one another to evaluate different configurations. For example, one could change the resource environment and deployment without changing the component architecture or the usage profile. For further details on PCM, we refer the reader to [5].

# 3. MODEL EXTENSION

The PCM is one of the most advanced meta-model for component-based systems in terms of parameterization and tool support. As described above, 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 different performance analysis techniques. However, in its current version, the interfaces between components are limited to synchronous operation calls and no means are provided to model event-based communication. We have developed a meta-model extension to satisfy this need. As previously mentioned, a first sketch of the required meta-model extensions was presented in [26].

Our approach introduces an abstract interface and signature description to enable more explicit subtypes for various component contracts. This enables the distinction between operational and event-driven interfaces as well as signatures and event types and respectively their event handlers. The meta-model changes are described in more detail below.

## 3.1 Events

Interfaces describe the contract between two components. In an operational, synchronous communication, interfaces combine a set of signatures. These signatures describe operations that are required by one component and provided by another. In event-based communication, the contract does not describe a set of operations that can be called but rather a set of event types one component might emit and one or more components can handle. As shown in Figure 1, the meta-model extension we propose contains an `EventGroup` as a specific type of an abstract `Interface`. It represents a contract which components can either require or provide. To describe the individual events which can be produced or consumed, our approach includes a meta-model element named `EventType`. This specific type of an abstract `Signature` references only one instance of the existing PCM `Parameter` element that describes the event characteristics, and does not reference any return values because of the asynchronous invocation style. The `Parameter` element, referenced by an `EventType`, allows to specify the performance-relevant characteristics of the data included in an event. A parameter can refer to either a simple or a complex data type.

## 3.2 Roles

Interfaces in general and `EventGroups` in event-driven systems are the contracts for component interactions. The presented approach aligns this requirement with the general PCM concept for providing and requiring component functionality. It introduces an abstract `ProvidedRole` and an abstract `RequiredRole` element which describe the roles of a component within a component connection.

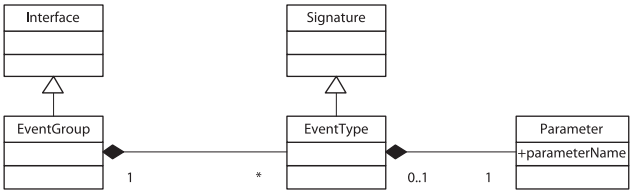


**Figure 1: Event Groups and Types**

In an event-driven interaction, the `SourceRole` is a `RequiredRole` to identify a component which might emit events of the types described by the referenced `EventGroup` and assumes a middleware that handles any further processing. The counterpart is the `SinkRole`. This is a `ProvidedRole` to identify a component providing event handlers processing the different `EventTypes` referenced by the `EventGroup`.

## 3.3 AssemblyEventConnector

Once component models are available as defined by the component developers, the system architect defines composed structures such as systems or composite components based on these components. Therefore, the architect has to specify the desired connections between sources and sinks of the same `EventGroups`.

In the existing PCM meta-model this requirement already existed for the operation calls between components. In the existing operational case, only one-to-one connections were allowed. In the new event-driven scenario, `SinkRoles` are able to handle events that are emitted by one or more `SourceRoles` and the events of one `SourceRole` can be handled by zero, one or many `SinkRoles`.

The presented approach introduces an `AssemblyEventConnector` to represent the connection between a sink and a source component. This `AssemblyEventConnector` is aligned to the design of an operational `AssemblyConnector` with the enhancement that an arbitrary number of `AssemblyEventConnectors` can start at the same `SourceRole` or end in the same `SinkRole`.

## 3.4 EmitEventAction

From a quality prediction perspective, the software architect needs to model the static structure of the components and the processing inside the components. The latter includes the point in time when events are emitted and their quality-related characteristics. According to the PCM service effect specification concept, a new action class has been designed to be placed in the RD-SEFF. A new `EmitEventAction` is introduced and able to emit events of a specific type identified by the referenced `EventType` class.

## 3.5 Event Handler

When a source emits a new event, all connected sinks are informed automatically. Each sink will individually process the event. With our approach, the architect is able to specify the quality-related behavior of the processing with the existing (RD-SEFF) facilities.

# 4. TRANSFORMATION

The meta-model changes described in the previous section enable software architects to model asynchronous, many-to-many event-based interactions in their systems. This reflects a high-level, conceptual representation of the inter-component communication as known from OMG's specification of a platform-independent model (PIM [21]).

To make use of the existing prediction facilities provided by the PCM, the model passes two modification steps as shown in Figure 2. In the first step, a transformation of the high-level model with the new event-related elements is performed to match the capabilities of the existing simulation and prediction techniques. In the second step, this platform-independent model is transformed to a platform-specific model. The latter transformation step weaves a middleware model into the platform-independent model to take the platform's quality properties into account.

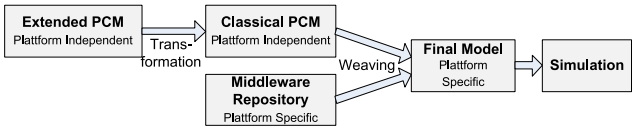


**Figure 2: Overall Transformation Process**

Surveying the high-level, event-based connection in the meta-model leads to a chain of processing steps involved in reality. Some of the steps are optional, while others are required. As our approach should support centralized as well as peer-to-peer middleware systems, the derived process chain is generic for both of them.

## 4.1 Platform-Independent Components

In reality, software communication at the protocol level is often implemented with synchronous interactions. These systems carry out their asynchronous behavior with intermediate components such as queues and hubs. In the presented approach the same strategy is applied. This concept enables the representation of the intermediate communication chain with components and processing descriptions already supported by the predictions provided by the PCM workbench.

### *4.1.1 Event Processing Pipeline*

As shown in Figure 3, a generic event processing mechanism includes six steps. First the event is processed on the sender-side. This processing is usually performed by a local library, which encapsulates the communication with the middleware server and might perform additional processing like compression or encryption. In the second step, the event is received and processed by the middleware system. Following this, the event is replicated for each receiving component. The fourth step represents the processing within the middleware which is done once for each receiving component. An example for such a processing is the filtering of events. As a last step the event is processed on the client-side. This step can again consist of decompression or decryption. It has to be mentioned, that by default the described steps do not include any processing. However, different processing activities can be integrated depending on the used event-based middleware. Such platform-specific processing activities are modeled within the middleware repository, which is later woven into the prediction model.

To facilitate a realistic quality prediction of the component communication, a set of intermediate components has been derived from the processing chain as described in the following.

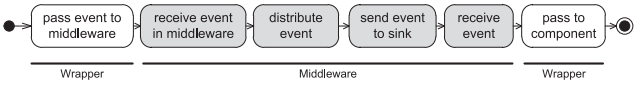


**Figure 3: Generic Event Processing**

### 4.1.2 Source Transformation

Figure 4 presents the mapping between a high-level source element and its low-level counterparts. Sources are transformed to a set of components. Each of those components is responsible for a different concern in the event processing chain.

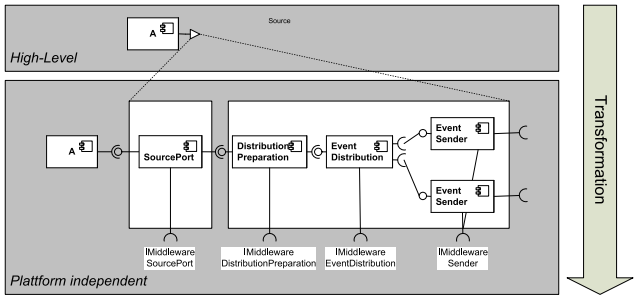


**Figure 4: Source to Intermediate Model Transformation**

The deployment of these components depends on the communication style of the underlying platform. There are two groups of components in relation to the deployment. The first group includes only the SourcePort. Individual instances of this component are deployed with each of the original source components.
The second group includes the DistributionPreparation, the EventDistribution and the EventSender. In a system with a centralized middleware node, there is only one centralized instance of each of these components deployed on the middleware resource container. If no such centralized middleware node exists, separate instances of these components are deployed with every source as done with the source port. For example, the latter concept applies to peer-to-peer event-based systems.

#### *Source.*

The source is the original component emitting the events. To model the described operational communication chain, the source is modified by replacing the event-based communication with synchronous operation calls. Strictly speaking, the component is now synchronously calling the SourcePort which takes care of any further processing.

#### *SourcePort.*

When a source emits a new event it has to be passed to the middleware. Depending on the platform, there is a wrapper component deployed with the source which takes care of the processing on the client-side. Such processing can include marshaling, queuing or similar actions. The intention of the source port component is to represent this wrapper and include any processing and quality-related demands. It is therefore deployed on the same `ResourceContainer` as the source component it belongs to.

To finally deploy the component, an `AllocationContext` is created for it in the same resource environment as the source component.

#### *DistributionPreparation.*

When a new event arrives at the middleware some processing might be required at this point. This processing takes place once per event and is independent of the number of recipients interested in it. The processing can include middleware-side marshaling, security-checks, compression and others. This varies with the underlying platform. To trigger the specific resource demands, the DistributionPreparation component is located as the first process on the middleware-related component chain.

In the case of a centralized middleware which is not running on the same infrastructure as the source component, the DistributionPreparation is deployed on the same server as the centralized middleware. This is done to analyze any resulting network traffic at the right point of the communication process.

If no centralized middleware exists, the component is deployed on the same `ResourceContainer` as the source component.

#### *EventDistribution.*

The event-based communication is often used in asynchronous and many-to-many relationships between components. On the one hand the EventDistribution component handles the replication and the decoupling on the other hand. Its internal processing triggers a new and independent control flow for each downstream sink to realize the asynchronous behavior.

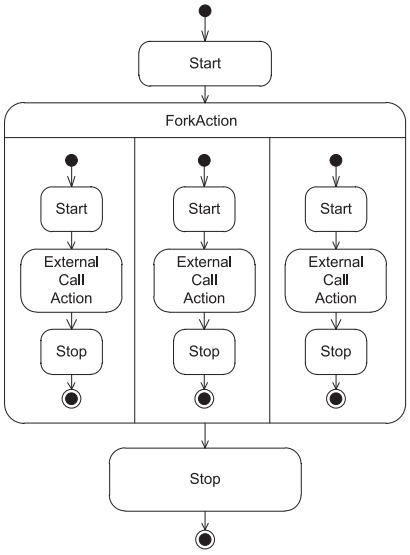


**Figure 5: Fork-Based Event Split**

As shown in Figure 5 the internal processing of the event distribution component makes use of a ForkAction element specified by the PCM meta-model. This action includes multiple ForkBehaviors to describe separate sub-flows taking care for the event replication. Without a synchronization element, this ForkAction decouples the individual sub-flows from the overall processing flow and permits the asynchronous behavior. The EventDistribution component is de-

ployed with the same concept as the DistributionPreparation.

*EventSender.*

When the middleware has performed the event replication it forwards the event to the recipients. This step might include additional processing such as compression, transformation, filtering or others. The sending itself can therefore have quality-related resource demands for every connected recipient. To reflect this in the model, separate EventSender components are placed in the chain per recipient.

### 4.1.3 Sink Transformation

The second part of the intermediate model creation is the transformation of sink elements. As with the sources, each connected sink role is replaced with a set of components. All of these components are deployed once per sink and do not depend on the existence of a centralized middleware node in the resource environment. Figure 6 provides an overview on the mapping described in the following paragraphs.

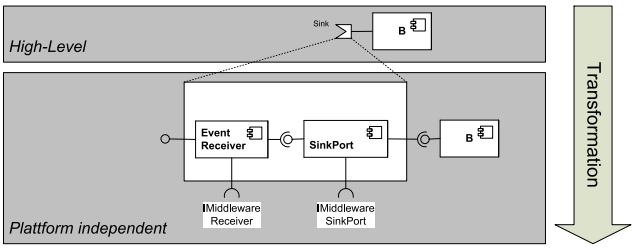


**Figure 6: Sink to Intermediate Model Transformation**

*EventReceiver.*

When an event arrives at a sink it first needs to be accepted. This might involve processing like de-marshaling or require a passive resource such as a thread pool. An EventReceiver component is placed in the communication chain and deployed with every sink component.

*SinkPort.*

Finally, the event needs to be passed to the original sink component. Depending on the platform this might have resource demands related to this final step. Even if this is often in smooth transition with the event receiving step, it is separated in this approach to not limit the flexibility for different platforms at this point.

*Sink.*

The original sink component is modified to handle the incoming operation calls instead of the emitted events. The existing RD-SEFFs of the component are now linked with the `Signatures` of the `OperationInterface` instead of the `EventTypes`. The name of the event content parameter is still the same as in the original `EmitEventAction` in the source component and does not change in the event processing chain. Due to this fact, the RD-SEFFs of the sink do not require any modifications in their internal processing and are still valid for the new incoming calls.

## 4.2 Platform-Specific Components

The available middleware products for event-based communication provide a wide range of event transmission architectures. This includes centralized message hubs, peer-to-peer systems and numerous architectures in between. The communication style and the concrete platform used for it can have a reasonable impact on the quality properties of the overall system. To decide on a platform and to find the right configuration for it, a software architect might need to select, configure, and test many different setups.

From a modeling point of view, the general event-based connection between components and the specific middleware used for the technical implementation are on two different levels of abstraction. To prevent unnecessary work, the architect does not want to change the high-level model for every platform he would like to test. To support the architect's work in this area, the presented approach enables him to use separate middleware models without any modifications of his high-level architecture. The presented approach contains a transformation that automatically weaves the additional middleware model into the event processing chain described above.

The separate middleware model has to provide predefined interfaces as presented in Figure 7. There are individual middleware interfaces for each component of the communication chain. How many components are used to provide those interfaces in the middleware is up to the architect and depends on the specific middleware product. This could include individual components for each interface or only one component for all of them. Figure 7 contains three possible alternatives for the middleware model.

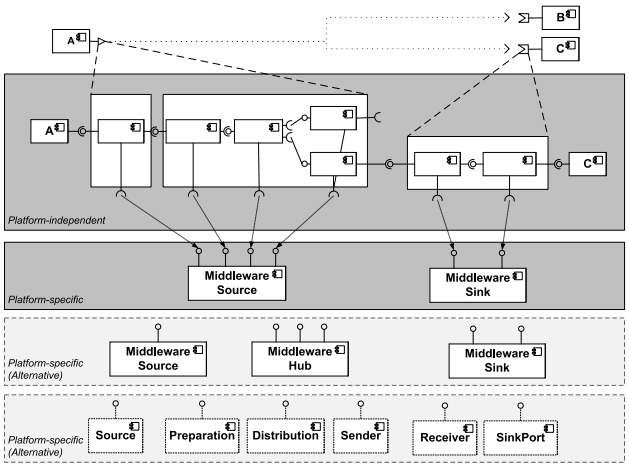


**Figure 7: Middleware Model Weaving Alternatives**

The middleware weaving is part of the transformation process developed for this approach. Figure 8 presents the required steps of this weaving sub-process.

In the first step a lookup for the middleware interfaces based on naming conventions is performed.

In a second step, required roles for these interfaces are added to the appropriate components of the platform-independent component chain. If this is done, an `ExternalCallAction` is added as the first action of the service effect specifications of these components. The `ExternalCallAction` includes a `VariableUsage` and `VariableCharacteri-`

`sations` to forward the characteristics of the currently processed event to the middleware.

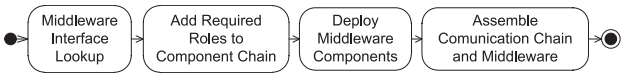


**Figure 8: Middleware Weaving Process**

As mentioned in the introduction to this paper and already applied during the deployment of the platform-independent components, the middleware components are deployed depending on the selected resource environment. If the selected environment contains a `ResourceContainer` with a name equals *"Middleware"*, instances of the middleware components providing the middleware interfaces for the DistributionPreparation, the EventDistribution and the EventSender are deployed centrally on this `ResourceContainer`. If this centralized middleware container does not exist, individual instances of these components will be deployed for every source component. With this automatic deployment concept, the architect can choose between a source infrastructure deployed as singleton or once per source. From his perspective he can simply specify a middleware container and let the transformation take care of the distribution.

As soon as `AssemblyContexts` are in place for all specified middleware components, the transformation creates the connectors between the platform-independent and the appropriate middleware components.

The specification of the middleware internals is completely up to the architect and can be modeled with the classical PCM model elements. For example, the software architect can model simple resource demands, control flows or passive resources. This separate middleware model can encapsulate enhancements in the future such as research results as mentioned by Happe [8] and others.

## 4.3 Implementation

To perform the evaluation described below, we implemented our approach in the PCM workbench. We applied the meta-model extension to the ecore-based PCM meta-model and generated the according code and tree-editors. Furthermore, we integrated the new modeling capabilities in the graphical editors using the Eclipse Graphical Modeling Framework (GMF). We implemented the model-to-model transformation with the QVTO transformation language. Finally, we extended the PCM Eclipse plug-ins to provide the middleware model selection to the user and to execute the transformation as part of the prediction workflow.

The goal of introducing the new meta-model elements to the PCM was to ease the modeling of event-based communication in the PCM while the model-transformation presented in Section 4 integrates platform-specific behavior and enables the use of existing prediction techniques. To evaluate our approach we use a real-world case study based on the traffic monitoring system developed by the University of Cambridge. After introducing this case study, we describe the different evaluation aspects following the Goal-Question-Metric (GQM) approach introduced by Basili et al. [3]. Finally we present the results of the evaluation combined with a short discussion.

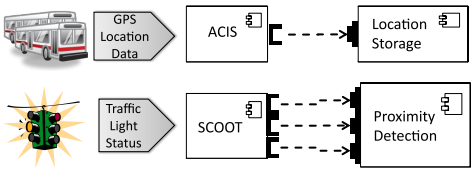


**Figure 9: Components of the TIME System**

# 5. EVALUATION

The goal of our approach is an improved modeling and prediction of event-based communication in component-based software architectures. To validate its applicability and its improvement, the new approach and the previously available Palladio version have been applied on the real-world case study presented in the following subsection. The new approach and the advantages it provides are assessed in comparison to the previously existing Palladio approach. To validate the prediction accuracy we evaluated 4 different deployment scenarios with different workload specifications. The evaluation of the modeling effort is based on a set of 5 different potential evolution scenarios as they can be observed in nearly all event-based system.

## 5.1 Case Study

The system we study is developed within the TIME project (Transport Information Monitoring Environment) [2] at the University of Cambridge. The system is based on a novel component-based middleware called SBUS (Stream BUS) [12] which supports peer-to-peer event-based communication including both continuous streams of data (e.g., from sensors), asynchronous events, and synchronous RPC. In SBUS each component is divided into a wrapper, provided by the SBUS framework, and the business logic that makes up the component's functionality. The wrapper manages all communication between components, including handling the network, registration of endpoints and event sinks, or marshaling of data.

The traffic monitoring system is used to estimate the speed of buses that are near traffic lights when they turn red. It requires information describing the current state of traffic lights alongside location information of buses. These two sources of data are, in many cases, not maintained by the same organization. This means that the application must combine data provided by multiple organizations. The implementation of this application uses four classes of SBUS components (see Figure 9) described below. Due to the middleware SBUS, it is possible to distribute these components over several computing nodes as well as centralize them on one node without any changes to the components' implementation. Finding the maximal processable event rate for a given deployment option or a resource-efficient deployment scenario that still meets all requirements regarding the event processing time or resource utilization is a complex task. Furthermore, the influence of newly integrated components on the other components is almost impossible without specialized evaluation techniques. Using performance prediction techniques eases the analysis of performance attributes for different deployment scenarios and event rates without prototypical implementations or test environments.

- Bus location provider (the "ACIS component")
  The bus location provider uses sensors to detect the locations of buses and reports any changes as a stream of events.

- Location Storage
  The location storage component maintains the state that describes for a set of objects, the most recent location that was reported for each of them. The location state is not conceptually a stream of events so, in the evaluated implementation, it is stored in a relational database that other components may query.

- Traffic light status reporter (the "SCOOT component")
  The traffic lights in the City of Cambridge are controlled by a SCOOT system [11], designed to schedule green and red lights to optimize use of the road network. SCOOT knows the status of each traffic light and provides a stream of events identifying that a light switches to red respectively to green. It further provides an RPC endpoint to retrieve location information about the traffic lights.

- Proximity Detector
  This component receives the event stream about traffic light status changes, uses the SCOOT component's RPC facility to determine the location of the traffic light and correlates this with the location information of the buses.

## 5.2 Evaluation Goals

A software prediction process includes a wide range of tasks ranging from system modeling to the interpretation of the prediction results. In order to focus on the crucial criteria of the presented approach, a Goal Question Metric (GQM) plan, as proposed by Basili et al. [3], has been developed to assess the quality of our approach. The overall goal of the work we presented in this paper is the improvement of the modeling capabilities and the reduction of the modeling effort related to event-based communication in the PCM quality prediction framework while keeping the performance prediction accurate. This goal can be split up into three questions:

- Is the modeling effort reduced by the new approach?
- Are the prediction results acceptable?
- Is the evaluation of different middleware settings/setups simplified?

While the improvement is always evaluated in comparison to the existing version of the PCM, the metrics for the questions are surveyed in comparison to the existing PCM capabilities. We use the results of our first manual case study [24], in which we also used the presented traffic monitoring system as case study. This allows us to compare the modeling effort as well as the prediction accuracy of the manually implemented model using classical PCM elements with those of a model based on the extended PCM combined with the presented transformation.

The first question targets the reduction of the modeling effort. We count the number of required element changes and creations within the model to analyze whether the required effort is reduced or not. This metric is independent of the individual velocity of the modeler who performs the model modifications.

The second question covers the accuracy of the prediction result. While the presented approach should simplify the modeling, the prediction results should provide at least the same quality as in the first case study. The applied metrics compare the new prediction results with the measurements of the real system as well as the existing prediction results.

The last question concerns the capabilities to extend the model to predict additional systems and setups. The metrics used to answer this question include the number of required steps to change the middleware configuration and to change the middleware at all.

## 5.3 Results and Discussion

For the evaluation, we modeled the traffic monitoring system with the new modeling facilities and analyzed the same scenarios as already done and validated in the previous manual case study [26]. Afterwards, we compared the two sets of prediction results against measurements on the existing system. Furthermore, we analyzed evolution scenarios to estimate required modification efforts. Additionally, we considered different evolution scenarios which require changes on the model such as adding new components or changing connections. We analyzed the required modeling steps to implement these changes.

### *5.3.1 Reduced Modeling Effort*

The original PCM meta-model did not provide any elements specific to events and event-based communication. As shown in the previous case study [26], using a workaround enables the architect to setup performance equivalent structures, however, this modeling is semantically incorrect as it directly uses forks and synchronous interfaces to emulate asynchronous behavior. With the presented approach, we introduce new meta-model elements for explicit modeling of `EventTypes`, `EventGroups`, `Sinks`, `Sources`, `EventConnectors` and `EventEmittingActions` for the processing flow. While there is no numerical quality index for such a difference, it can be clearly stated that there is an improvement on the coverage of event-related elements. Figures 10 and 11 present the same connection between an ACIS and a LocationStorage component. While in the old approach it was not clear that this represents an event-driven connection, the new approach makes it explicit.

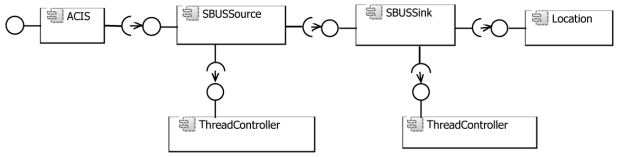


**Figure 10: Source Sink Connection - Old Case Study**

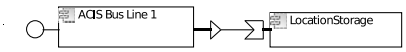


**Figure 11: Source Sink Connection - New Case Study**

In addition to the semantically correct modeling of event-based communication the new elements reduce the modeling

effort. We tracked the effort for three different scenarios. In the first scenario a complete new connection between a source and a sink was created. In the second and third scenarios, an additional sink respectively source were added to an existing connection. We tracked the effort in terms of element creations and did not consider the time consumption to create the different elements. This was done to remove the influence of the individual experience and training of the architects in the usage of Eclipse modeling tools in general and the PCM tools in particular. For a completely new connection, the required effort was reduced from 59 to only 11 new elements with the new approach. Adding an additional sink is reduced to only one new element instead of 14 with the old approach. And the effort for adding an additional source is reduced from 35 to 6 elements.

| Change Scenario | New #elements | Old #elements | Effort Reduction |
|---|---|---|---|
| New Connection | 11 | 59 | 81,3% |
| Add Sink | 1 | 14 | 92,8% |
| Add Source | 6 | 35 | 82,8% |

**Table 1: Reduction of Modeling Effort**

These results (see Table 1) are clear indicators for the reduced effort to model event-based communication between components. Even without the specific effort per event creation, the results of the metric highlight the benefits of the new approach.

### 5.3.2 Accuracy of Prediction Results

Beside the improved modeling capabilities when compared to the old approach, it is also necessary to provide a quality prediction which at least meets the previous accuracy level. To validate this statement, the 4 scenarios of the manual case study [26] have been analyzed again with a performance model using the presented extensions. The new prediction results were compared to those of the manual case study as well as to the original measurements that were part of the first case study. In Figure 12, we present exemplary prediction results of Scenario 4. The results show that in this scenario the prediction based on the presented approach is even more accurate than the manually created model. For all scenarios the prediction error between measurements and the new prediction model did not exceed 8.3%. For the sake of brevity we cannot present all evaluation results and refer to [14] for more details.

### 5.3.3 Testing Different Middleware Setups/Settings

In addition to the improvement in modeling and predicting existing solutions, the process to model new configurations and architectures to predict changes before their implementation should be simplified. To answer the related GQM question, the same metrics as for the second question were used.

Two different scenarios were investigated to analyze the effort required to change the middleware. In the first scenario, the configuration of the existing middleware was changed. In the second scenario, the middleware was replaced completely. As a test case for the first scenario, the SBUS middleware should be changed from a single-threaded to a multi-threaded processing. With the old approach this required 15 modifications on the model but with the new approach the

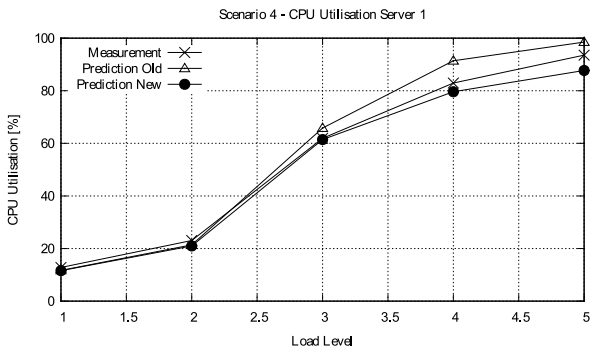


**Figure 12: Scenario 4 - CPU Utilisation**

effort is reduced to 6 modifications. In the second scenario we replaced the complete middleware. To focus on the replacement effort, we assume that the new middleware model is already in place and only the connection between the middleware components and traffic monitoring components are considered. With the old approach 22 modifications are required to release the old middleware and integrate the new one. This dramatically changes with the new approach while the only modification is to select the new middleware repository for the prediction.

## 6. RELATED WORK

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) [27]. Since then a number of architecture-level 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. Architecture-level performance models are built 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. In recent years, with the increasing adoption of component-driven software engineering, the performance evaluation community has focused on adapting and extending conventional SPE techniques to support component-driven systems which are typically used for building modern systems. A recent survey of approaches to performance evaluation of component-based systems was published in [16]. Examples of common meta-models used in this context include PCM [5], SAMM [1] and CBML [31]. Existing meta-models, however, currently do not provide support for modeling events and event-based communication using first class entities.

In the following, we present an overview of existing performance modeling and analysis techniques specialized for event-driven systems including systems based on message-oriented middleware (MOM) considering both centralized and distributed environments. Event-based communication has been of research interest in the software engineering community for years. This has led to a wide range of approaches to analyze the characteristics of event-based communica-

tions. However, most of them (e.g., [20] or [6]) are focused on analytical models for the event distribution infrastructure and do not consider the entire system architecture.

A recent survey of techniques for benchmarking and performance modeling of event-driven systems was published in [15]. In [9], an analytical model of the message processing time and throughput of the WebSphereMQ JMS server is presented and validated through measurements. Several similar studies using Sun Java System MQ, FioranoMQ, ActiveMQ, and BEA WebLogic JMS Server were published. A more in-depth analysis of the message waiting time for the FioranoMQ JMS server is presented in [19]. However, these publications only consider the overall message throughput and latency and do not provide any means to model event-driven interactions and message flows.

There are existing approaches to model high-level architectures and to add platform-specific characteristics as part of the prediction. [28] presents a framework to automatically include the impact of CORBA middleware on the performance of distributed systems but they completely ignore the influence of service parameters like the amount of data to be processed. There are some existing approaches to apply this to the Palladio Component Model. For example, Happe et. al. [8] took advantage of an approach recommended by Woodside et. al. [30] to automatically include details about a message-oriented middleware into a model. However, this approach is limited to 1-to-1-connections and does not support modeling of systems following the publish-subscribe paradigm. Kapova and Goldschmidt [13] presented an approach to use higher order transformations to generate transformations that refine the architecture model itself. However, this approach is also limited to 1-to-1-connections and does not support an encapsulated and selectable repository for the middleware-specific components.

# 7. CONCLUSION AND OUTLOOK

In this paper, we presented an extension of the Palladio Component Model, which enables a semantically correct modeling of event-based communication. Combined with the presented automated model-to-model transformation, the modeling effort for event-based communication could be reduced significantly while still supporting all existing prediction techniques such as simulation, LQNs or QPNs. The transformation is partitioned into a platform-independent and a platform-specific part. In the first part, a mapping from the new elements to a set of elements already supported by the prediction facilities has been defined, which follows a generic event processing chain as it can be found in centralized as well as in decentralized messaging systems. In the second step, platform-specific components located in a separate middleware repository are woven into the prediction model. Due to this separation, the influence of using different middleware systems can be analyzed by simply selecting another middleware repository and the system itself can be modeled independent of the underlying middleware.

We integrated the presented model extension as well as the transformation in the PCM workbench. In a real-world case study, based on a traffic monitoring system developed at the University of Cambridge, we evaluated the accuracy of the performance predictions. The prediction error did not exceed 8.5%. Considering different potential change scenarios, we estimated the required modeling effort by tracking the required changes in model elements. With the presented approach, the modeling effort could be reduced by more than 80%.

The results presented in this paper form the basis for several areas of future work. In the current version of the proposed PCM extension, filtering of events has to be modeled manually in the behavior model of the sink. We plan to further extend the PCM to specify filtering rules by using the stochastic expression language, which is already part of the PCM. These expressions combined with a BranchAction are then automatically published to the EventDistributionComponent. Additionally, we are working on a larger case study with more components and different design and deployment alternatives. With this case study, we plan to demonstrate the benefits of having a separate platform middleware repository which can be easily substituted. Furthermore, we are working on the integration of the Performance Cockpit [29] to automatically derive the platform-specific middleware repositories from existing middleware products.

# 8. REFERENCES


[1] Q-impress project deliverable d2.1 - service architecture meta-model (samm). Project Deliverable, 2008.

[2] 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*, pages 687–691, 2008.

[3] V. R. Basili, G. Caldiera, and H. D. Rombach. The Goal Question Metric Approach. In J. J. Marciniak, editor, *Encyclopedia of Software Engineering - 2 Volume Set*, pages 528–532. John Wiley & Sons, 1994.

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

[6] 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.

[7] J. Happe. *Predicting Software Performance in Symmetric Multi-core and Multiprocessor Environments*. Dissertation, University of Oldenburg, Germany, August 2008.

[8] J. Happe, S. Becker, C. Rathfelder, H. Friedrich, and R. H. Reussner. Parametric Performance Completions for Model-Driven Performance Prediction. *Performance Evaluation*, 67(8):694–716, 2010.

[9] 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.

[10] B. Hohpe, Gregor ; Woolf. *Enterprise integration*

*patterns : designing, building, and deploying messaging solutions*. The Addison-Wesley signature series. Addison-Wesley, Boston, Mass., 11. print. edition, 2008.

[11] P. B. Hunt, D. I. Robertson, R. D. Bretherton, and R. I. Winton. SCOOT—a traffic responsive method of coordinating signals. Technical Report LR1014, Transport and Road Research Laboratory, 1981.

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

[13] L. Kapova and T. Goldschmidt. Automated feature model-based generation of refinement transformations. In *Proceedings of the 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2009.

[14] B. Klatt. Modelling and prediction of event-based communication in component-based architectures. Master's thesis, Karlsruhe Institute of Technology, Germany, 2010.

[15] S. Kounev and K. Sachs. Benchmarking and Performance Modeling of Event-Based Systems. *it - Information Technology*, 5, Sept. 2009.

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

[17] 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*, volume 5119 of *Lecture Notes in Computer Science*, pages 58–78. Springer-Verlag Berlin Heidelberg, 2008.

[18] P. Meier, S. Kounev, and H. Koziolek. Automated Transformation of Palladio Component Models to Queueing Petri Nets, 2011.

[19] 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.

[20] 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), Delft, The Netherlands, August 25-28, 2009*. Springer Verlag, 2009.

[21] Object Management Group (OMG). MDA Guide, June 2003.

[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] C. Rathfelder, D. Evans, and S. Kounev. Predictive Modelling of Peer-to-Peer Event-driven Communication in Component-based Systems. In A. Aldini, M. Bernardo, L. Bononi, and V. Cortellessa, editors, *Proceedings of the 7th European Performance Engineering Workshop (EPEW'10), University Residential Center of Bertinoro, Italy*, volume 6342 of *Lecture Notes in Computer Science*, pages 219–235. Springer-Verlag Berlin Heidelberg, 2010.

[25] C. Rathfelder, B. Klatt, S. Kounev, and D. Evans. (Poster Paper) Towards Middleware-aware Integration of Event-based Communication into the Palladio Component Model. In *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems (DEBS-2010)*, Cambridge, United Kingdom, July 2010. ACM, New York, USA.

[26] C. Rathfelder and S. Kounev. (Position Paper) Modeling Event-Driven Service-Oriented Systems using the Palladio Component Model. In *Proceedings of the 1st International Workshop on the Quality of Service-Oriented Software Systems (QUASOSS)*, pages 33–38. ACM, New York, NY, USA, 2009.

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

[28] T. Verdickt, B. Dhoedt, F. Gielen, and P. Demeester. Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models. *IEEE Transactions on Software Engineering*, 31(8):695–711, 2005.

[29] D. Westermann, J. Happe, M. Hauck, and C. Heupel. The performance cockpit approach: A framework for systematic performance evaluations. In *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010)*. IEEE Computer Society, 2010.

[30] M. Woodside, D. Petriu, and K. Siddiqui. Performance-related completions for software specifications. *Software Engineering, International Conference on*, 0:22, 2002.

[31] X. Wu and M. Woodside. Performance modeling from software components. *SIGSOFT Softw. Eng. Notes*, 29(1):290–301, 2004.