Towards a Modular Palladio Component Model

Misha Strittmatter, Philipp Merkle, Andreas Rentschler, Michael Langhammer
Institute for Data Structures and Program Organisation (IPD)
Karlsruhe Institute of Technology (KIT)
{lastname}@kit.edu

Abstract: The Palladio Bench started out as a tool for designing and analyzing the performance of component-based enterprise software systems. Over the following years, numerous extensions have been introduced to increase Palladio's analysis capabilities. These include in particular support for additional quality attributes. With an eye on current research projects, we expect this trend to continue. This includes opening up Palladio for multiple technology domains, besides its current focus on software systems. The ever-growing number of modeling constructs, however, easily leads to users being overwhelmed by modeling complexity they do not even need in many cases. Similarly, developers of Palladio tooling struggle with consequences of weak modularization. We therefore propose a refactoring of the historically grown Palladio Component Model (PCM) into different modules that build upon each other. A core module is planned to serve as architecture description language (ADL). Modules for different quality dimensions enrich the ADL by specific modeling and analysis capabilities. This paper presents our vision of a modular PCM. Due to the early stage of the project, we focus on discussing requirements and challenges, and present initial solution ideas.

1 Introduction

With the Palladio approach [2], one can evaluate quality attributes of software already at the modeling stage. At Palladio's core lies the Palladio component model (PCM) that is used to describe component-based software designs along with quality-relevant properties. Initially, only performance-relevant concerns had been supported by the PCM, but more recently, concerns to predict reliability and security have been integrated as well. Ongoing research aims to support the prediction of further quality attributes, and to prepare the PCM so that other domains can be modeled. However, if we include further concerns, like security or additional structural concepts, our tooling will contain functionality, in which only a part of users is interested in. Moreover, the more concerns are integrated into a single modeling artifact, the harder it gets to understand and maintain the metamodel.

To avoid these problems we are planning to modularize Palladio's metamodel. The goal of the refactoring is to split up the metamodel into a core metamodel, which can be considered an architecture description language. Capabilities to model properties like performance, reliability, security and so on, will be contained in metamodel extensions which can be used by installing additional plug-ins. This way users can use only the functionality they are interested in. Furthermore, the PCM gets more flexible and thus can be used as integration platform for future extensions. From a technical point of view, the metamodel becomes

Proc. Kieker/Palladio Days 2013, Nov. 27–29, Karlsruhe, Germany
Available online: http://ceur-ws.org/Vol-1083/

cleaner, and further extensions of the metamodel can be included with less effort. To ensure compatibility with existing tools and legacy instances of the metamodel we plan a transformation from the modularized PCM instances to classical PCM instances.

The contribution of the paper is threefold: First, we present our vision of a modularized PCM. Second, we bring our plans up for discussion to accomplish our goal. Third, we list requirements for a modularized PCM we collected from various stakeholders. The remainder of the paper is structured as follows: In Section 2, we discuss extension scenarios of the PCM. In Section 3, we present our vision and propose solutions how the refactoring can be done. Section 4 explains technical requirements and open challenges. Section 5 concludes the paper and gives an outlook to future work.

2 Extension Scenarios

Several extensions to the PCM have been proposed over the last years, most of which made their way into Palladio's metamodel, its model editors and analysis tools. These extensions can be divided into at least three groups. Extensions of the first group enrich Palladio by additional quality attributes such as reliability [3] or security. Extensions of the second group tailor Palladio for specific types of software systems such as event-based systems [16], storage-intensive systems [15] or transactional systems [12]. Extensions of the third group use Palladio mainly as an ADL, for example to document design decisions [4], or to maintain traces between requirements and architecture artifacts [9]. Extensions can also fall in more than one group simultaneously. Tailoring Palladio to transactional systems, for instance, paves the way for including data consistency as a new quality attribute.

In what follows, we take the PCM-REL extension as an example representative of a group one extension, and analyze the metamodel changes introduced. We observe that metamodel changes in this example boil down to a small set of *extension types*. In Section 4, we use the identified extension types to discuss potential extension mechanisms.

The *PCM-REL* extension provided by Brosch [3] introduces reliability as a new quality dimension into Palladio. Component service invocations (`ExternalCallActions` and `EntryLevelSystemCalls`) no longer are guaranteed to succeed but may fail with a certain failure on demand (FOD) probability. For this to work, component-internal calculations (`InternalActions`) in PCM-REL may contain newly introduced `FailureOccurrenceDescriptions`. Each determines the FOD probability for a specific (software-induced) failure type. Earlier, component developers specify possible failure types and store them in the component repository. Component service invocations can recover from software-induced failures if the failing `InternalAction` is nested inside a newly introduced `RecoveryAction`. When recovery is not possible, the `retry count` – an additional attribute of `ExternalCallActions` – determines the maximum number of retries. The specifications of processing resources (`ProcessingResourceSpecifictions`) possesses the additional attributes `MTTF` (mean time to failure) and `MTTR` (mean time to repair) to specify hardware-induced failures and, respectively, the recovery from such failures. Similarly, `CommunicationLinkResourceSpecifications` possesses an additional attribute `failure`

`probability`.

From this description, we can extract the following extension types that can be found in PCM-REL:

ET1 Adding a metaclass

ET2 Adding an attribute to an existing metaclass

ET3 Adding a containment reference to an existing metaclass

Support for these extension types would be a good starting point for a modular PCM. Yet, to be able to support the whole range of extensions, they will most likely not suffice.

3 Modularization Approach

Our vision of how to refactor the Palladio Bench can be best described starting from the metamodel (the PCM). An illustration is shown in Figure 1. On the right side, the current version of the metamodel (here *PCM Classic*) is depicted. As indicated by the upper arrow labeled refactor, the current metamodel is refactored into the modular metamodel, which is shown on the left. The ellipses on the left represent metamodel modules.

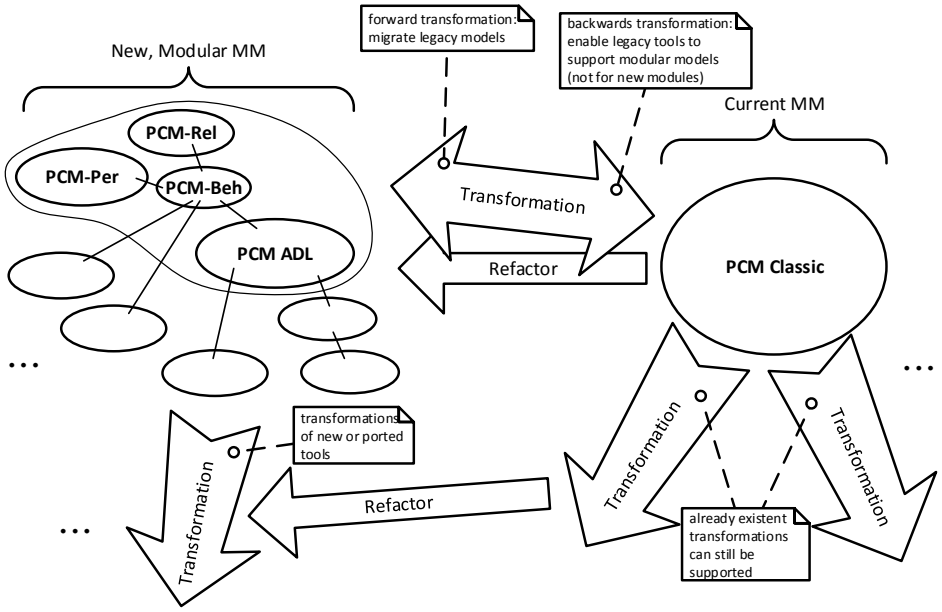

Figure 1: PCM Refactoring Illustrated

The refactoring process will entail modularization as well as the forming of an ADL. The ADL forming results in the PCM ADL module. Everything else contained in the classic

metamodel is refactored into modules which are located within the outline of the curved shape. Please note that the picture does not depict all modules, as it serves only to give an overview of our proposed approach. For example, core concepts like usage model, deployment and resource environment are not depicted. How exactly the metamodel is cut is part of future work. The modules outside of the curved shape are new extensions which will result from current research (e.g., security, code mapping [10], tracing of requirements and design decisions [9]).

The lines which connect the modules represent a relation between the modules. One possible relation is: an element of one module specializes an element of the other module. Which extension mechanisms will be ultimately used, however, is yet another open question.

A transformation from the classic to the modular metamodel can migrate legacy models (forward transformation). The backwards transformation, from the modular to the classic metamodel, ensures backwards compatibility for existing tools (e.g., simulators and solvers). This eases the development process, especially with regards to release planing, as not all tools have to be adapted at once to work with the modular metamodel. Transformations and tools, which are actively developed, can be gradually ported to operate on the modular metamodel. Tools that are no longer actively maintained will remain operational. Once the modular metamodel is established, changes and new extensions will only be performed on the modular metamodel. By applying first the forward, then the backward transformation on a model, one can validate the correctness of these transformations, as well as the completeness of the modular metamodel.

4 Technical Requirements and Open Challenges

So far, we explained only the impacts of the refactoring onto the metamodel. Based on the extension scenarios pointed out in Section 2, the refactoring also impacts tools, corresponding transformations, and editors. Our goal is to provide a framework where metamodel extensions are encapsulated within plug-ins. These plug-ins also contain the functionality needed for the other layers to handle content of the metamodel extension.

This could result in the following procedure: when a new PCM model is created, the user selects the required extensions depending on the purpose of the model. The PCM ADL module will always be included. An extension is only available, if the containing plug-in is installed. Predefined configurations could be provided, which suggest specific sets of extensions to the user. For example, the performance configuration could include usage, resource environment, behavior, resource demands (an extension for the behavior extension), deployment and so forth. The extension configuration of a model can also be changed after the model was created. Adding of extensions will always be possible; removing of extensions will be more challenging, especially if the model still contains information which belongs to the extension which is to be removed. One possible and interesting feature is the hiding of information of extensions or switching between them. For example, when modeling a behavior specification, one could switch between displaying performance, reliability and security relevant information, tracing of design decision, requirement and the

documentation of applied patterns. A model can only be viewed and edited, if all necessary plug-ins are installed. Otherwise the user could be prompted to install missing plug-ins.

In the remainder of this section we discuss the requirements and challenges posed by the refactoring and the development of such a plug-in mechanism onto the layers of the Palladio Bench. These layers include the metamodel and model persistence layer, the transformation layer, the simulator layer, and the editor layer.

4.1 Metamodel

For the Palladio Bench to be able to support a plug-in mechanism poses some requirements to the metamodel layer. For the extensions presented earlier, a proper metamodel extension mechanism has to be found.

Adding a new metaclass (**ET1**) is straight forward. For the new metaclass to be instantiable within models, it either has to inherit a preexisting abstract metaclass or has to be referenced through **ET2** or **ET3**. Inheriting a non-abstract metaclass is not considered **ET1** but rather **ET2** or **ET3**, as it is a decoration of a preexisting metaclass.

Another way to achieve **ET2** and **ET3** is through the usage of EMF profiles and stereotypes [8]. A stereotype can contain additional primitive attributes, complex typed references as well as containment references. By assigning a stereotype to a model element, it is enriched by the features defined within the stereotype. Other extension mechanisms could exist, we do not make the claim to be comprehensive.

Extension by inheritance requires the metamodel to be designed in a way that accounts for variability. Abstract metaclasses have to be put in place, where **ET1** extensions are expected. The work of Heinrich et al. [5] of extending the PCM Usage Model by business process concepts is a good example for this. Amongst others, Palladio usage models are extended by new control flow actions by specializing the metaclass `AbstractUserAction`. Stereotypes do not require as much foresight, however, they should be defined in a way they can only be applied to the right entities.

4.2 Transformations

By now, Palladio comprises several transformations that map the PCM to various targets. Typical transformations translate PCM instances to various analysis frameworks or analysis models, for example the simulation and prototyping frameworks SimuCom and ProtoCom [1], queuing Petri nets (QPN) [11], and layered queueing networks (LQN) [7], but also to other targets, for instance the Java EE platform. According to our experience, maintaining all these transformations is time-consuming: changes in the PCM usually require to regain an exhaustive understanding of the code in order to adapt the transformation accordingly.

With the PCM and transformations being modularized accordingly, transformation logic for metamodel extensions is factored out into separate modules. Thereby we facilitate

understandability and reuse. We aim to break down existing transformation along two dimensions. First, each transformation part should be responsible for a certain PCM module. When the PCM metamodel evolves, principal focus can be put on migrating core concepts before spending time on less important extensions. Second, such a transformation part could be decomposed further, so that similar functionality is factored out for reuse across several target models. Reuse of transformation units has already been pursued for some Palladio transformations [1]. When changes in the PCM can be traced to a reused transformation modules, changing a single location in the code can be sufficient to migrate multiple transformations for distinct targets in one go.

Most of our transformations are model-to-text transformations. They map to text-based artifacts using Xtend as a template-based transformation language. Thus their structure is driven by the target's code structure. Each target consists of modules that override template methods and advice methods of core templates. To have self-contained conceptual extensions we must additionally cut a transformation in alignment with existing source metamodel cuts. This is possible if a core transformation module provides certain extension points, so that additional concepts can be woven in at multiple defined places, similar to aspects. Aspects can be implemented in Java using object-oriented design patterns like the decorator pattern, or with aspect-oriented programming (AOP) techniques, for instance annotations provided by Spring or AspectJ. Since Xtend is a Java-based language, we can exploit any of these facilities [14].

Palladio also incorporates model-to-model transformations. For example the PCM2QPN transformation [11] has been implemented in QVT-Operational. For this family of languages, there is yet no modularization concept known to us that is suitable for the objectives mentioned above. At the moment we are working on integrating information hiding modularity into Xtend and QVT-Operational [17] with well-defined interfaces to reduce maintenance efforts. Still, crosscutting concerns are not directly supported by these languages. At least in Xtend we can exploit existing Java techniques as mentioned above.

4.3 Simulator

Introducing new concepts on the metamodel level usually comes along with introducing new simulation behavior to reflect the added semantics. Such simulator extensions currently flow directly into the respective simulator's code base, leading to monolithic simulators. With a modular PCM, we see the chance of modularizing the simulation as well. In addition to metamodel extensions, each PCM module could contribute its particular extension to the overall simulation behavior. This creates the need for pluggable simulation modules.

Pluggable simulation modules must declare what extension points they offer, and what extensions they provide to other modules. This is exactly the idea of the plug-in mechanism available with Eclipse's OSGi implementation Equinox. Since most of Palladio's simulators are Eclipse-based anyway, a natural decision would be to take advantage of the plug-in mechanism.

The definition of extension points, however, is challenging and requires in-depth knowledge

of the current simulation and of potential simulator extensions provided in future. Furthermore, the simulation architecture must be designed with extension points in mind. As an example, EventSim [13] supports extending SEFFs by additional control flow actions; this is reflected in EventSim by means of *traversal strategies*, which have a large influence on the overall architecture.

4.4 Editors

Palladio currently uses the Graphical Modeling Framework (GMF) to provide graphical editors for the PCM. GMF uses model-driven technologies, GMF models are used to link graphical elements to abstract concepts. From these models, graphical editors are generated. Separate GMF editors exist for six different views of the PCM, each requiring its own set of GMF models. GMF Tooling and the underlying GMF Runtime both do not allow editors to be configured dynamically regarding the modeling concepts presented to users – neither by developers at design time or by users at run time.

With a modularized PCM model, however, metamodel extensions should be able to bring their own editor extensions. At runtime, editors must be able to dynamically load these extensions depending on the user's demands. There are two imaginable kinds of editor extensions, those that plug into existing PCM editors, and those providing additional editors for custom views on the added concepts.

The following technical challenge needs to be tackled: When a module is adding modeling concepts in the form of new metaclasses to the PCM (**ET1**), these additions can require several views and thus editors to be adapted accordingly. Scattering and tangling occurs, because a concept can be scattered over multiple editors, or an editor tangles multiple modeling concepts, even at the level of code artifacts. These issues occur for both graphical and textual representations, and extensibility must be handled dynamically at runtime, for example using an Eclipse-based plugin mechanism. Adding attributes or containment references to existing metaclasses (**ET2**, **ET3**) is already in the process of being integrated into GMF editors with the help of a stereotyping mechanism based on EMF Profiles [8].

Under Eclipse, there is another prominent framework to design graphical representations, the Graphiti framework. The Graphiti framework is designed for manual programming in a declarative style, therefore it is much leaner but still more flexible, and can be conceived as a successor to GMF. It seems that Graphiti as well does not explicitly support extensible editors so far. Because its streamlined API is easier to understand, we believe Graphiti is a prime candidate to be prepared for runtime composability. Further research is required to explore techniques to make the Graphiti framework susceptible for extensible metamodels.

In the future, textual editors may complement Palladio's graphical editors as alternative views with similar expressiveness. As far as we know, textual editors can be already designed in a composable fashion, albeit only at design-time. For example, MontiCore and Spoofax/IMP [6] are designed for modular language extensibility, and the Xtext editors are extensible to lesser extent by supporting inheritance and import of grammar modules. In this area as well, further research is required to pick up the most suitable solution.

5 Conclusion and Outlook

In this paper, we presented the motivation behind the refactoring, identified different types of extension scenarios, proposed a vision of how to solve these issues and outlined the arising requirements and challenges we identified so far.

The refactoring endeavor is especially important for the future development of the Palladio Approach. We strive to broaden the application range of the Palladio Approach to cover more and more stages of the software development process. The artifacts and information created in these stages should not be captured in separate models, managed by separate tools. Rather, we aim to consolidate them into the PCM and by doing so to relate them to the component-based architecture. Architecture design, performance and reliability are some of the features already incorporated within the PCM. Tracing of Requirements, design decisions and patterns, code mapping, security characteristics and many more are currently subject to research. In addition to that, modeling concepts from new domains like energy infrastructure and public transportation will also be incorporated. To support a conceptually clean extension mechanism we propose to form an ADL as a basis and need to modularize the current content of the metamodel. We suspect such a refactoring to also improve maintainability and understandability of the metamodel and maybe even of the Palladio Bench.

We are currently in the phase of collecting requirements and developing a concept on how to cut the metamodel and how the ADL core should look like. For the future we plan to form research questions from the scope of refactoring metamodels, metamodel quality and from the refactoring of architecture simulators. An in-depth literature review is also planned to reveal similar approaches on which we can possibly build upon. To substantiate the concept of the metamodel cut and the ADL, we also need to identify as much future extension as possible, to account for variety at the right spots of the metamodel. This will be followed by an evaluation of technical solutions. Any feedback to our approach is always very welcome.

6 Acknowledgments

We would like to thank the further members of the special interest group PCM Refactoring for their valuable input: Jörg Henß, Lucia Happe, Max Kramer and Robert Heinrich. Further we thank Klaus Krogmann, Ralf Reussner and the whole SDQ Chair for many fruitful discussions.

References

[1] Steffen Becker. “Coupled Model Transformations”. In: *WOSP ’08: Proceedings of the 7th International Workshop on Software and performance*. New York, NY, USA: ACM, 2008, pp. 103–114.

[2] Steffen Becker, Heiko Koziolek, and Ralf Reussner. “The Palladio component model for model-driven performance prediction”. In: *Journal of Systems and Software* 82.1 (Jan. 2009), pp. 3–22.

[3] Franz Brosch, Heiko Koziolek, Barbora Buhnova, and Ralf Reussner. “Architecture-based Reliability Prediction with the Palladio Component Model”. In: *Transactions on Software Engineering* 38.6 (2011).

[4] Zoya Durdik and Ralf Reussner. “Position Paper: Approach for Architectural Design and Modelling with Documented Design Decisions (ADMD3)”. In: *Proceedings of the 8th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2012), Bertinoro, Italy*. 2012.

[5] Robert Heinrich, Barbara Paech, and Jörg Henss. “Extending Palladio by Business Process Simulation Concepts”. In: *Palladio Days* (2012), pp. 19–27.

[6] Lennart C.L. Kats, Karl T. Kalleberg, and Eelco Visser. “Domain-Specific Languages for Composable Editor Plugins”. In: *Electronic Notes in Theoretical Computer Science* 253.7 (2010). Proceedings of the Ninth Workshop on Language Descriptions Tools and Applications (LDTA 2009), pp. 149–163.

[7] Heiko Koziolek and Ralf Reussner. “A Model Transformation from the Palladio Component Model to Layered Queueing Networks”. In: *Performance Evaluation: Metrics, Models and Benchmarks, SIPEW 2008*. Vol. 5119. Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 2008, pp. 58–78.

[8] Max E. Kramer, Zoya Durdik, Michael Hauck, Jörg Henss, Martin Küster, Philipp Merkle, and Andreas Rentschler. “Extending the Palladio Component Model using Profiles and Stereotypes”. In: *Palladio Days 2012 Proceedings (appeared as technical report)*. Ed. by Steffen Becker, Jens Happe, Anne Koziolek, and Ralf Reussner. Karlsruhe Reports in Informatics ; 2012,21. Karlsruhe: KIT, Faculty of Informatics, 2012, pp. 7–15.

[9] Martin Küster. “Architecture-Centric Modeling of Design Decisions for Validation and Traceability”. In: *Proceedings of the 7th European Conference on Software Architecture (ECSA ’13)*. Ed. by Khalil Drira. Vol. 7957. Lecture Notes in Computer Science. Montpellier, France: Springer Berlin Heidelberg, 2013, pp. 184–191.

[10] Michael Langhammer. “Co-evolution of component-based architecture-model and object-oriented source code”. In: *Proceedings of the 18th international doctoral symposium on Components and architecture*. ACM. 2013, pp. 37–42.

[11] Philipp Meier, Samuel Kounev, and Heiko Koziolek. “Automated Transformation of Component-based Software Architecture Models to Queueing Petri Nets”. In: *19th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2011)*. 2011.

[12] Philipp Merkle. “Predicting transaction quality for balanced data consistency and performance”. In: *Proceedings of the 18th international doctoral symposium on Components and architecture*. WCOP ’13. New York, NY, USA: ACM, 2013, pp. 13–18.

[13] Philipp Merkle and Jörg Henss. “EventSim – An Event-driven Palladio Software Architecture Simulator”. In: *Palladio Days 2011 Proceedings (appeared as technical report)*. Ed. by Steffen Becker, Jens Happe, and Ralf Reussner. Karlsruhe Reports in Informatics ; 2011,32. Karlsruhe: KIT, Fakultät für Informatik, 2011, pp. 15–22.

[14] Miroslaw Milewski and Graham Roberts. “Patterns for Handling Crosscutting Concerns in Model-Driven Software Development”. In: *10th European Conference on Pattern Languages of Programs (EuroPLoP 2005)*. 2005.

[15] Qais Noorshams, Andreas Rentschler, Samuel Kounev, and Ralf Reussner. “A Generic Approach for Architecture-level Performance Modeling and Prediction of Virtualized Storage Systems”. In: *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. ICPE ’13. New York, NY, USA: ACM, 2013, pp. 339–342.

[16] Christoph Rathfelder, Benjamin Klatt, Kai Sachs, and Samuel Kounev. “Modeling Event-based Communication in Component-based Software Architectures for Performance Predictions”. In: *Journal of Software and Systems Modeling (SoSyM)* (Mar. 2013), pp. 1–27.

[17] Dominik Werle. “Adding a Module Concept to the Model Transformation Language Xtend”. Bachelor’s Thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology, Sept. 2013.